Feedback-driven usability testing for hotel software

Feedback-basiertes Usability-Testen für Hotelsoftware

by

Stephan Lukasewitz, Kevin Nöller

Master's Thesis

presented for the degree of
Master of Science in Informatics

TUM School of Computation, Information and Technology

Prof. Pramod Bhatotia, Prof. Bernd Brügge

München, 15.11.2024

This master's thesis in informatics is a collaborative effort between Stephan Lukasewitz and Kevin Nöller. The respective contributions are marked with the abbreviations 'SL' and 'KN' at the section titles. The abbreviations are not repeated in subsections if they are already indicated in the main section title.

I, Stephan Lukasewitz confirm that, any sections in this master's thesis in informatics, marked with 'SL' is my own work and I have documented all sources and material used.

I, Kevin Nöller confirm that, any sections in this master's thesis in informatics, marked with 'KN' is my own work and I have documented all sources and material used.

Acknowledgements

We would like to express our deepest gratitude to Prof. Bernd Brügge. His support and expertise had a decisive impact on this work and extended far beyond the supervision of this thesis. His course iPraktikum in 2023 provided us with a great foundation, which has been invaluable not only in the following semesters but also as the catalyst for the development of the CheckedInn system.

Special thanks also go to Prof. Pramod Bhatotia for granting us the opportunity to undertake this thesis at his chair.

Lastly, we would like to thank all hotels that participated in the study.

Abstract

From simple paperwork and supporting systems to complex and complicated software solutions - the ongoing digitalization is more present than ever in the hotel industry. Therefore, the quality of the property management software (PMS) used can not only determine the efficiency, but even the ability of a hotel to stay competitive. While more powerful systems can better support day-to-day business, they are often less user-friendly. This poses a major challenge for several reasons. Unexperienced receptionists, high staff turnover and complex processes make usability and user experience particularly relevant. The focus of this thesis is a usability study to investigate the addition of a feedback mechanism to increase the usability in an existing system. For the investigation we reengineered an existing system for booking and hotel management called CheckedInn.

The usability testing study was conducted over a period of nine weeks and five iterations performing a total of 23 usability tests. 13 hotels participated in the usability tests. User feedback was collected based on Nielsen's usability heuristics through Likert scales and natural language and classified into the categories Deployment Problems, Bug Reports, Change Requests, Feature Proposals and Usability Issues. The natural language processing was performed by utilizing ChatGPT 4o.

Over the five testing iterations, the overall satisfaction of the usability of the reengineered system yielded an increase by 18.75%. Moreover, the iterative testing approach reduced bug reports by 95.45%. Usability issues improved by 83.33% from the first to the fifth testing iteration.

We conclude that involving end users through the reengineering of the system has increased the quality of the system, in particular its usability.

Zusammenfassung

Vom einfachen Papierkram über unterstützende Systeme bis hin zu komplexen und komplizierten Softwarelösungen - die fortschreitende Digitalisierung ist in der Hotellerie präsenter denn je. Daher kann die Qualität der verwendeten Property Management Software (PMS) nicht nur die Effizienz, sondern sogar die Fähigkeit eines Hotels bestimmen, wettbewerbsfähig zu bleiben. Während leistungsfähigere Systeme das Tagesgeschäft besser unterstützen können, sind sie oft weniger benutzerfreundlich. Dies stellt aus mehreren Gründen eine große Herausforderung dar. Unerfahrene Rezeptionisten, hohe Personalfluktuation und komplexe Prozesse machen Usability und User Experience besonders relevant. Im Mittelpunkt dieser Arbeit steht eine Usability-Studie, in der untersucht wird, wie ein Feedback-Mechanismus die Benutzerfreundlichkeit in einem bestehenden System verbessern kann. Für die Untersuchung wurde ein bestehendes System für Buchungs- und Hotelmanagement namens CheckedInn überarbeitet.

Die Usability-Teststudie wurde über einen Zeitraum von neun Wochen und fünf Iterationen mit insgesamt 23 Usability-Tests durchgeführt. 13 Hotels haben an den Usability-Tests teilgenommen. Das Nutzerfeedback wurde auf der Grundlage von Nielsens Usability-Heuristik mittels Likert-Skalen und natürlicher Sprache gesammelt und in die Kategorien Implementierungsprobleme, Fehlerberichte, Änderungswünsche, Feature-Vorschläge und Usability-Probleme eingeteilt. Die Verarbeitung der natürlichen Sprache wurde mit ChatGPT 4o durchgeführt.

Während der fünf Testiterationen stieg die Gesamtzufriedenheit mit der Benutzerfreundlichkeit des überarbeiteten Systems um 18,75%. Außerdem reduzierte der iterative Testansatz die Fehlerberichte um 95,45%. Die Benutzerfreundlichkeit verbesserte sich von der ersten bis zur fünften Testiteration um 83,33%.

Es folgt daraus, dass die Einbeziehung der Endnutzer in das Reengineering des Systems die Qualität des Systems CheckedInn und insbesondere seine Benutzerfreundlichkeit erhöht hat.

Table of Contents

Acknowledgements ..... **iii**

Abstract ..... **iv**

1. Introduction (KN) **1**

2. Terminology (SL) **5**

2.1 Usability 5

2.2 Usability Testing 9

2.2.1 Heuristic Evaluation 9

2.2.2 Discount Usability Testing 13

3. Reengineering Process (KN) **16**

3.1 Approach 16

3.1.1 Inventory analysis 18

3.1.2 Reverse engineering 19

3.1.3. Forward engineering 20

3.2 Usability Testing Methodology (SL) 21

3.2.1 Participants 22

3.2.2 Study Design 23

3.2.3 Feedback Evaluation 27

4. Inventory of the current system (KN) **33**

4.1. Functional Requirements 33

4.2. Non-Functional Requirements 37

4.3 Design Rationale 40

5. Reverse engineering (SL) **41**

5.1 Reservation Scenario 43

5.2 Trace Scenario (KN) 49

5.3 Booking Manager (SL) 54

5.4 Financial Manager (KN) 58

5.5 Report Scenario 67

5.6 Requirements fulfillment evaluation 72

6. Forward Engineering (KN) **75**

6.1 Feedback type-based evaluation 76

6.1.1 Deployment problems 77

6.1.2 Bug reports 80
6.1.3 Change requests 84
6.1.4 Feature proposals 86
6.1.5 Usability issues 88
6.1.6 Cumulative analysis 91
6.2 Usability Heuristic-based evaluation (SL) 93
6.2.1 Visibility of the system status 93
6.2.2 Match between System and the Real World 95
6.2.3 Error prevention 97
6.2.4 Overall Usability 100
6.3 Threats to Validity 102
7. Conclusion (SL) 107
7.1 Summary of Findings 109
7.2 Future Work 112
Appendix 114
A Questionnaire 114
B Models 123
C Exemplary handling of study feedback (KN) 124
C1 Exemplary handling of a deployment problem 124
C2 Exemplary handling of a bug report 125
C3 Exemplary handling of a change request (SL) 127
C4 Exemplary handling of a feature proposal (SL) 129
C5 Exemplary handling of a usability issue (KN) 132
D Heuristic Evaluation 136
Bibliography 139

1. Introduction (KN)

This chapter introduces the topic of the master's thesis. To do so, an introduction to the subject domain is first given in order to clarify the relevance of the topic. The agenda, methodology and structure of the thesis are then briefly explained in order to provide a navigational overview of the following chapters.

Digitalization has become a driving force in the hotel industry, transforming operations from simple paperwork and basic systems to advanced, versatile software solutions. This change is driven not only by the need for efficiency and flexibility in a competitive market, but also by increased expectations for seamless service and real-time management [1]. As hotel software becomes increasingly complex, ensuring its usability is critical. This is further reinforced by the fact that, even a few years after the height of the Covid-19 pandemic, the hotel industry is still severely affected by its consequences [2]. Staff shortages, acquisitions from abroad and the growing popularity of alternative accommodation options are just a few of the challenges facing the industry [3]. Therefore, the need for suitable and efficient software solutions is greater than ever before. Especially at a time when contactless processes and automated workflows are becoming increasingly important, the usability of hotel software is of central importance. User-friendly and intuitive systems can not only make the day-to-day work of staff much easier, but also improve customer satisfaction. Another key factor is the often untrained or temporary nature of hotel staff, which makes a well-designed user experience crucial. When staff turnover is high or employees don't have the time or resources for extensive training, hotel software needs to be intuitive enough to ensure smooth operation from day one. This minimizes errors and shortens the learning curve, allowing staff to focus on their core tasks instead of dealing with complicated interfaces. In addition, hotel operations involve a variety of complex processes, from booking management and check-in/check-out procedures to housekeeping and customer service. Given this complexity, a clear and structured software is essential for an efficient workflow. A good overview of tasks, real-time updates and easy access to key functions are crucial to avoid confusion and bottlenecks and ensure that operations run smoothly even during busy periods. In summary, the quality and particularly the usability of the property management software a hotel uses can not only improve its efficiency, but can even determine its ability to stay competitive.
However, usability engineering is not a trivial task and therefore requires a structured approach [4]. Furthermore, usability is a widely relevant non-functional requirement of software of any kind and is therefore by no means limited to software for the hotel industry. As a result, usability has long been recognized as a fundamental topic in software

development, and extensive research has been dedicated to understanding and improving usability [5]. While usability engineering can significantly enhance the user experience, it often requires a substantial investment of time, resources, and specialized knowledge. Achieving statistically significant results in usability testing can be resource-intensive, making it difficult to implement in many projects, including the context of this thesis [6]. In such cases, established guidelines like Nielsen's heuristics for usability engineering offer a practical alternative [7]. These heuristics provide a proven framework for evaluating and improving usability without the need for extensive resources, allowing for meaningful improvements even in constrained environments. The goal of this thesis is to observe how the usability of hotel software can be systematically assessed and optimized by focusing on these well-established principles. Therefore, this thesis is not a literature review, but a field study in which the underlying principles are applied in practice and empirical research is conducted.

The foundation for this thesis is the Property Management System (PMS) CheckedInn 1.0, a system that has already been developed. This software, later often referred to as the current system, enables a variety of central functions that ensure the smooth running of daily operations. For example, reservations can be easily created and managed, making it much easier to coordinate room bookings. The system also offers the option of efficiently recording and invoicing additional services, such as wellness offers or room service orders. Another important aspect supported by the current system is the management of guest correspondence, which covers communication before arrival as well as during and after the stay. Moreover, it supports the creation of invoices, which significantly speeds up and simplifies the check-out process. Housekeeping can also be managed via the software by coordinating and monitoring tasks such as room cleaning and maintenance requests. Administrative tasks such as the management of prices, restrictions and availability, as well as the processing of internal finances and the creation of sales reports are also seamlessly integrated into the software. This not only increases operational efficiency, but also ensures a high degree of control and clarity. All this makes it clear that the functionality of CheckedInn 1.0 is well suited to the day-to-day business of the hotel industry. However, the system suffers from usability issues. Therefore, CheckedInn 1.0 serves as a suitable foundation for conducting feedback-driven usability tests for hotel software.

As this thesis is not about developing a new system from scratch, but rather revising an existing system, one is dealing with a reengineering project with a focus on usability engineering. A detailed description of the reengineering process applied in this thesis can be found in section 3.1. In the prior development of the CheckedInn 1.0 system, the focus

was on rapid innovation, which meant that mainly source code was developed, but the system was not modeled. This means that the source code space is vastly larger than the model space. In order to successfully adapt and further develop the existing system, the models of the current system must therefore first be derived from the source code. The prerequisite for this is that the functional and non-functional requirements on which the development of the existing CheckedInn 1.0 system is based, also known as the reengineering inventory, have been recorded in detail. This is done in Chapter 4 and the existing system can then be represented in the form of models. This fundamental part of the thesis is referred to as reverse engineering and is conducted in detail in Chapter 5.
Once the system has been modeled, forward engineering can then be carried out. The resulting system is referred to as CheckedInn 2.0. Since a particular focus here is on improving usability, a feedback system is introduced and usability tests are conducted in parallel. As already mentioned, the resources to carry out large-scale usability studies are not available in the context of this study. The focus is therefore on the application of Nielsen's usability heuristics. To carry out the usability studies, 13 hotels agreed to take part in the study. This offers optimal conditions for conducting discount usability testing (see section 2.2.2), as the number of hotels allows for exciting insights into the needs of industry-related users, but at the same time is not too large and thus enables close cooperation with the users. During the implementation of the feedback from the usability studies, there is a strong focus on users participating in the study as beta testers in order to tailor adjustments and new features optimally to the preferences of actual users during implementation. A detailed explanation of the implementation of the discount usability testing methodology can be found in section 3.2. Figure 1.1 illustrates the steps of reengineering based on the division into model space and source code space. While

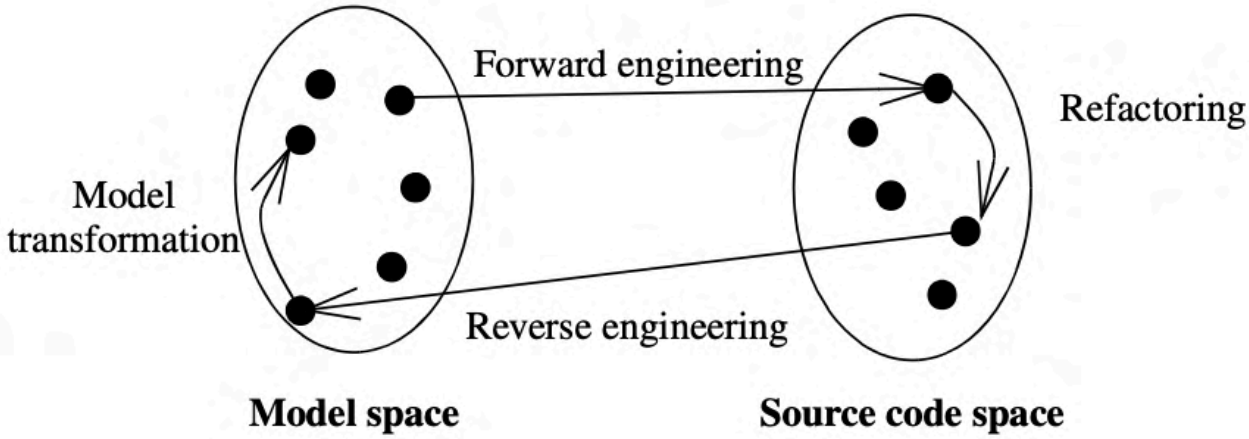


Figure 1.1: Types of transformations involved in the reengineering process [8]

refactoring and model transformation are part of the development of CheckedInn 2.0, they are not the focus of this thesis. Forward Engineering is driving the transition from CheckedInn 1.0 to CheckedInn 2.0. In addition to the usability tests, the implementation is a central component. However, as this is mainly of a technical nature, the focus is on the experimental evaluation of the usability tests.

During the experimental evaluation, it is investigated whether interesting patterns can be derived from the participants' feedback. The aim is to determine how well the improvement in usability has worked and to discuss which particular processes are successful. The aim is also to gather experience and recommendations for future and other reengineering projects, as well as for the subsequent further development of CheckedInn 2.0. The experimental evaluation is conducted in detail in chapter 6.

Following the discussion of the research objective's relevance and the thesis structure, chapter 2 provides a detailed explanation and definition of the key terms pertinent to this study.

2. Terminology (SL)

This chapter deals with the terminology relevant to this work. Chapter 2 analyses the terminology in historical and literary terms. In particular, the terms *usability* (section 2.1) and *usability testing* (section 2.2) take centre stage. These terms will be frequently mentioned, especially in sections 3.2 and 6, during the discussion and implementation of the study.

2.1 Usability

With the commercialization of personal computers in the 1970s and even before, the relevance of the product, not the user, always applied at first. With initially simple programs, this was not a problem at first. However, programs quickly became increasingly complicated to use as their technical complexity grew. As a result, technically well-designed programs failed to fulfill their actual purpose due to a lack of usability [9].
With the publication of the book The Psychology of Human-Computer Interaction by Stuart K. Card, Thomas P. Moran, and Allen Newell [10], the term Human-Computer Interaction (HCI) became popular only a short time later in 1983. At the same time, the research field of HCI sprouted up around the term, which eventually also introduced the term usability [11].
In 1988, the term user-centered design was introduced by Donald Norman in this context. This recommended that products should no longer be designed solely with a focus on functionality, but in particular with a focus on the user. User centered design advocates for a holistic focus on user experience at every stage of the design and development process, from ideation to implementation, ensuring that the final product is not only functional but also usable and understandable. [12]
This perspective was taken further three years later by Shackel, when the environment of the system was also taken into account for a good system design. This resulted, among other things, in the relevance of handling user support and his resulting definition of usability as:

"The capability in human functional terms to be used easily and effectively by the specified range of users, given specified training and user support, to fulfill the specified range of tasks, within the specified range of environmental scenarios."

In this context, Shackel was also the first to quantify effectiveness, learnability, flexibility and attitude with his metrics. Shackel suggests assigning numerical values to the terms

when setting the usability goals in the requirements specification and then checking whether the values have been achieved. [13]
The approach of a metrics-based definition was pursued further by Jakob Nielsen in 1993. He introduced usability as a clear component of system acceptance on the same level as technical functionality, which finally made usability equally relevant to technical implementation in literary terms. At the same time, he described a variety of options for testing usability based on this. These are described in more detail in section 2.2.

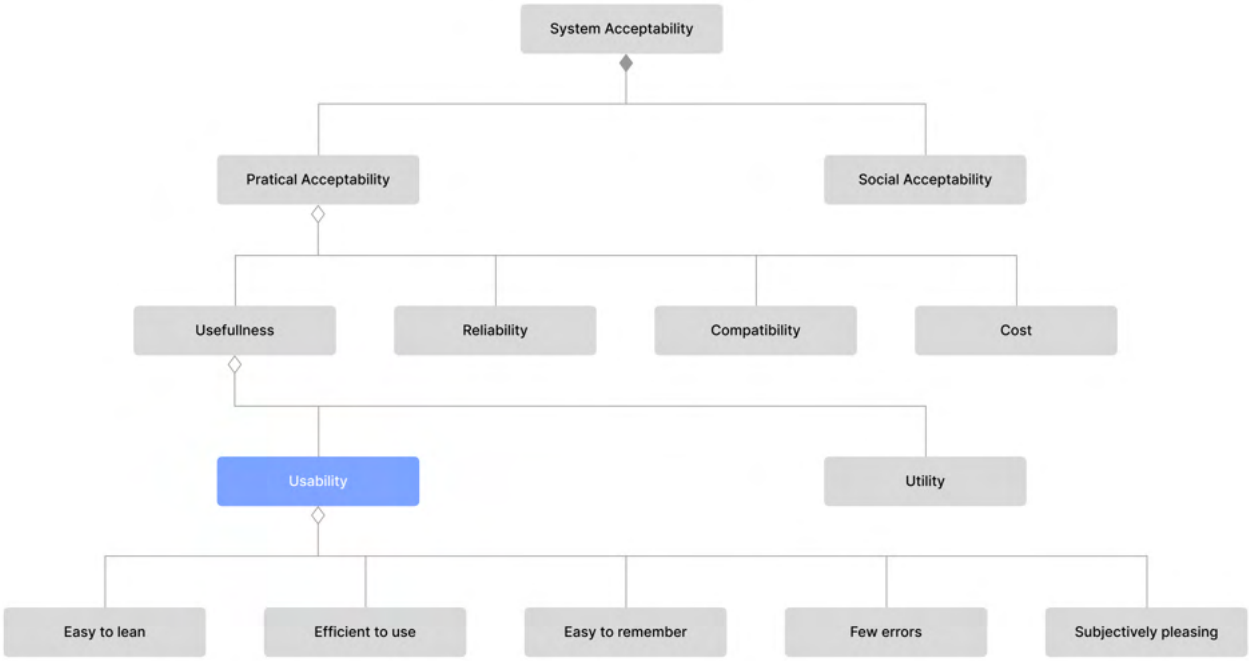


Figure 2.1: UML Class Diagram based on Nielsen's „attributes of system acceptability" [4]

Figure 2.1 shows the criteria described by Nielsen for the acceptance of a system. In addition to practical acceptability, he also addresses social acceptability. This can be, for example, people's open-mindedness towards the use of certain technologies. Practical acceptability, on the other hand, encompasses a more quantifiable set of factors. In addition to costs, compatibility and reliability, the component that Nielsen is primarily focussing on is usefulness and, in particular, usability. He categorizes these into five acceptance criteria: easy to learn, efficient to use, easy to remember, few errors and subjectively pleasing.

Learnability

Learnability is a fundamental component of usability and describes the user's ability to quickly familiarize themselves with a new system. Systems that are easy to understand generally have a steep learning curve for users. In this way, they can operate the system

quickly for the most part. This is particularly important for applications that are intended for single use and where error-free use on the first attempt is particularly desirable. However, this learning curve can also vary if users already have experience with this or similar systems. In this case, a higher level of proficiency can be assumed. When measuring learnability, it is particularly important to test with test subjects who are representative of the target group. The time it takes them to reach a satisfactory level of proficiency is then measured. In the context of an explorative learning process, it is particularly helpful to have descriptive error messages and back buttons.

Efficiency of Use

If a converging state of the learning curve is recognizable, the efficiency of use is evident. Jakob Nielsen refers to this state as steady-state. It is usually reached after a long period of use and depends heavily on the complexity of the system. It aims to measure how quickly certain tasks can be completed with the system. Given that the users are in a steady-state, testing for efficiency is a suitable way of determining the practical effectiveness of the system.

Memorability

Memorability is particularly relevant for occasional users. In other words, those who use the system occasionally but are neither complete novices nor highly experienced, frequent users. An example of this could be systems for generating quarterly reports. Memorability aims to measure how quickly a user who used the system some time ago can familiarize themselves with it again. It should be possible for these users to utilize functionalities without having to relearn how to use them correctly. The ability to teach users how to use the system over the long term without having to familiarize themselves with it again is the central challenge.

Few and Noncatastrophic Errors

Nielsen distinguishes between two types of errors: minor and catastrophic errors. In principle, anything that fails to achieve the desired goal when the user action is correct can be categorized as an error. However, not all errors have the same effect. Minor errors can be corrected by the user and therefore only take a little more time to use. Catastrophic errors, on the other hand, can limit the user experience and cause errors that are not immediately obvious. Nielsen emphasizes not only avoiding errors, but also providing users with appropriate feedback in the event of errors. Interfaces should also be designed in such a way that they move the user away from catastrophic errors. As a metric, Nielsen suggests measuring the frequency with which errors occur.

Subjective Satisfaction

Subjective satisfaction describes how pleasant and enjoyable a system is to use. This is particularly important for non-work environments. For such systems, the entertainment factor is often even more relevant than, for example, efficiency. Satisfaction can be achieved through an appealing design or good user feedback, among other things. Typical ways of quantifying the user experience include questionnaires with numerical response options. Although it would be possible to determine even more specific measurements, such as the heart rate during use, such quantifications are rarely used due to the time-consuming test implementation. Another complicating factor is that subjective perception can often be influenced when obtaining measured values. For example, the user may feel the need to provide particularly good feedback.

Four years later, this definition was also adopted in a similar way by the International Organization for Standardization (ISO). The most commonly referenced ISO definition of usability comes from ISO 9241-11:1998. The official definition of usability helped to create a cross-industry understanding of usability and its increasing popularity in the industry. The definition breaks usability down into three main factors:

Effectiveness

The accuracy and completeness with which specified users can achieve specified goals in particular environments.

Efficiency

The resources expended in relation to the accuracy and completeness of the goals achieved.

Satisfaction

The comfort and acceptability of the work system to its users and other people affected by its use.

The ISO also emphasizes that usability must always be seen in the context of use. Depending on users, tasks and equipment, the picture of good usability can vary greatly. For this reason, they also propose a guide for evaluating usability. This emphasizes the importance of early interaction with future users when implementing a system. A focus should also be placed on the emotions of users during testing. [14]

After this definition based on Nielsen was well received not only in the industry but also in science, a large number of definitions based on it were created. A very popular one was created by Quesenbery in 2002 [15].
In 2018, the ISO updated the earlier definition by refining the definitions to better align them with current usability and user experience practices. It expands the context of use and emphasizes a more holistic approach to evaluating the user experience. [16]

Due to the great acceptance and influence of the ISO definition for usability on science, industry and especially user experience designers, the ISO definition for usability is used as the basic definition in this thesis. This adoption of the ISO definition enables a consistent, structured approach to assessing usability across different systems and interfaces.

2.2 Usability Testing

This section explores the two most common approaches to evaluating usability: heuristic evaluation (2.2.1) and discount usability testing (2.2.2). Each method provides insights into the usability of systems, and their combined use can offer a comprehensive understanding of user experience.

2.2.1 Heuristic Evaluation

Heuristic evaluation aims to have the system tested by evaluators. These can be future users or other external testers. Testing is carried out by having the testers familiarize themselves with the system and assess usability based on heuristics. This method of testing is particularly practical in terms of efficiency and effectiveness.
Jakob Nielsen first presented his nine usability heuristics in 1990. These were part of his early work on heuristic evaluation, a method he developed together with Rolf Molich [17]. The original set was intended as a simplified checklist for identifying usability problems when evaluating user interfaces. About conducting a heuristic evaluation, they state:

"Heuristic evaluation is done by looking at an interface and trying to come up with an opinion about what is good and bad about the interface." [17]

4 years later, Nielsen added a tenth heuristic to this list. Its main purpose was to refine and extend the framework to better cover aspects of user interaction and support. The addition of the tenth heuristic aimed to provide usability evaluators and designers with a more

comprehensive set of tools to ensure that interfaces are not only functional and efficient, but also accessible and user-friendly. [4]

In 2020, Nielsen published a slightly revised version of the 10 heuristics for usability in an online blog article of the Nielsen Norman Group. These updates provided clearer explanations and additional examples. Further, the language was modernized to reflect current technology and design practices. The following heuristics are involved:

H1: Visibility of System Status

The system should always keep users informed about what is going on, through appropriate feedback within a reasonable time. It ensures that at any given moment, users are not left guessing what the system is doing or what has just occurred. This transparency helps to foster a sense of control and predictability. It further enhances the user's confidence and satisfaction with the interface.

H2: Match between System and the Real World

The system should speak the users' language, with words, phrases, and concepts familiar to the user, rather than system-oriented terms. Real-world conventions should be followed, making information appear in a natural and logical order. This alignment can help users understand and navigate the system more effectively, leading to a smoother user experience. Visual metaphors should be chosen based on recognizable symbols from the user's environment.

H3: User Control and Freedom

Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Undoing and redoing actions should always be possible and visually highlighted so that the users can navigate freely and easily.

H4: Consistency and Standards

Users should not have to wonder whether different words, situations, or actions mean the same thing. Platform and industry conventions should be followed.

H5: Error Prevention

Even better than good error messages is a careful design that prevents a problem from occurring in the first place. Systems should either eliminate conditions that are prone to errors or check for such conditions and present users with a confirmation option before they commit to an action.

H6: Recognition Rather Than Recall

The user's memory load should be minimized by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

H7: Flexibility and Efficiency of Use

Accelerators, unseen by the novice user, may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Users should be allowed to tailor frequent actions.

H8: Aesthetic and Minimalist Design

Dialogues should not contain information that is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

H9: Recognize, Diagnose, and Recover from Errors

Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

H10: Help and Documentation

Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, should list concrete steps to be carried out, and not be too large.

[7]

In addition to the selected heuristics, which serve as the central basis for the planned tests, numerous other aspects must be considered that are essential for a comprehensive evaluation of the system. Section 2.2.2 deals in detail with determining the number of evaluators that play a critical role in the evaluation process. It is also important to define the ideal profile of these evaluators, especially in terms of their experience and knowledge of the system under test. In addition, it is necessary to determine the optimal number of interactions between the evaluators and the system to ensure valid and meaningful test results.

Nielsen carried out a study to test how much influence the test subjects' usability experience has on the results of the tests. In this study, it was tested how many users are required to find a certain percentage of usability issues in a system. Users were divided into three groups: Users who are very experienced with usability and the system, those who have a little experience and users who have no experience with the system to be tested and good usability.

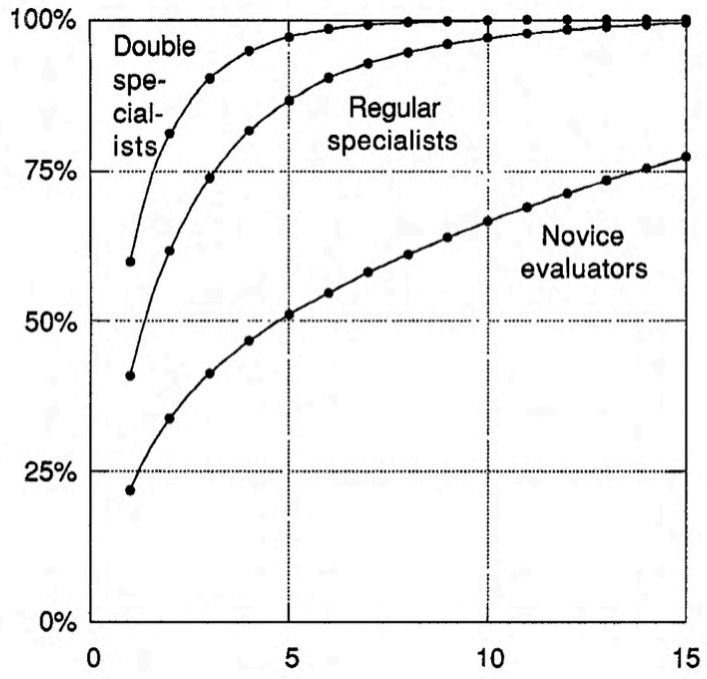


Figure 2.2: Average proportion of usability problems found as a function of number of evaluators in a group performing the heuristic evaluation [4]

Figure 2.2 shows the result of this study. It can be seen that the group of novice evaluators needed five people to find half of the usability issues. Only with 14 participants were they able to exceed the benchmark of 75%. The users with sufficient experience already reached the 75% mark in a group of four people. With eight people, 90% of the usability issues could then be found. As the number of people increases, the value converges towards 100%. For the experts, a group of two people was already sufficient to exceed the 75% benchmark. With four participants, over 90% of the issues could be found. [18]

The result shows that it makes a big difference to the effectiveness of the usability study how experienced the users already are with the system and with using software in general. It is clear that effectiveness and efficiency in testing can be increased with increasing expertise of the evaluators. In particular, this also has an impact on the cost efficiency of testing, as fewer testers are required to find a minimum number of issues.

In addition to cost efficiency, the speed with which the study can be carried out is a particular advantage. The test processes can often be completed within a short time. This makes it highly suitable for agile development environments where rapid iterations are common.

2.2.2 Discount Usability Testing

Discount usability testing was first taken up by Nielsen in 1993. The approach is grounded in the idea that usability does not have to be a luxury only large corporations can afford. Instead, it can be integrated into any design process through simplified techniques that are both economical and effective. [4]

It cannot be reduced to a concrete methodology, but is rather an aggregation of various core elements that Nielsen has expanded over several years and thus proposes effective usability testing with a high cost-user ratio. Probably the most important element of discount usability testing is the heuristic evaluation presented in 2.2.1. As already anticipated in figure 2.2 in 1992 that five test participants are a good number for carrying out a test, Nielsen expanded on this approach in 2000. In an online article, he describes that instead of extensive user testing sessions, discount usability testing often employs shorter, more focused sessions that might involve around five users. This number is based on Nielsen's research, which suggests that testing with five users tends to uncover most of the significant usability issues without diminishing returns. He explains that users often start solving tasks in a similar way, but deviate in their approach during the process. These deviations in execution would then lead to the discovery of new usability issues. He goes on to write that from a number of six test participants, the additional findings usually decrease further, which begins to reduce the cost-benefit ratio. A higher number of testers is only necessary if the artifact to be tested has a large variance of user groups. [19]

Nielsen also discusses the scenarios that should be tested. He initially suggests testing scenarios in which the level of functionality and the number of features are low. On the one hand, this allows the user to formulate clear feedback and, at the same time, the developer to make adjustments within a clearly defined scope. This not only contributes to efficiency and lower costs, but also to flexibility in the testing process. This is because the small and clearly defined scope of the scenario means that many iterations can be carried out quickly and feedback can be responded to quickly. [4]

As part of the Simplified Thinking Aloud principle, Nielsen also suggests dispensing with complex documentation processes in the testing process. For example, elaborate voice or video recordings are not necessary for carrying out discount usability tests. It makes much more sense to take notes of the test subjects' thoughts spoken aloud. This is not only more

effective for obtaining data, but also makes testing much more practicable and therefore more accessible for the people carrying out the test. This facilitates early and frequent testing. Finally, this principle also contributes to the speed and cost efficiency of the testing process. [4]

As a final component of discount usability testing, Nielsen proposes an iterative design. This is closely related to the insight that ideally a test run should be carried out in a group of 5 evaluators. Nielsen recommends spreading the budget for user testing over several smaller studies instead of investing it in a single, extensive study. If one has the funds to recruit 15 users, it makes more sense to use these funds for three studies with five users each. The iterative nature of this method allows design changes to be made after each round of testing, gradually improving the user experience.

The iterative design works according to a cycle of design, test and redesign. After the first round of testing, in which most of the obvious usability issues are identified, the product is revised based on the feedback. The revised design is then tested again to ensure that the changes were effective and to determine if any new issues have arisen. This method ensures that the design is continuously improved and that the final product is as user-friendly as possible.

Another advantage of iterative design is that it allows for deeper insights into usability. In later studies, aspects such as information architecture, task flow and alignment with user needs that may have been overlooked in earlier phases can be examined in more detail. This is particularly important as superficial issues often require a lot of attention in the initial tests and can mask deeper structural issues.

The results of each study serve as the basis for the next, which not only ensures the quality of the results of the previous study, but also creates new insights that need to be addressed in further redesigns. This ongoing process of evaluation and improvement means that the final user experience is significantly improved by three studies with five users each, compared to a single large study with 15 users. [19]

In summary, discount usability testing offers a practical and cost-effective solution to the widespread problem of usability given the constantly changing requirements for web and software development. Nielsen also emphasizes the high overall economic relevance of early and frequent testing of computer software:

[...] bad intranet Web design will cost $50-100 billion per year in lost employee productivity [...]. *[20]*

The fact that this approach makes it possible to conduct effective usability tests without having to resort to expensive and time-consuming procedures is a key reason why the following work focuses on discount usability testing. It provides the ideal basis for increasing the accessibility and effectiveness of usability practices and minimizing the productivity losses caused by poor design. Section 3.2 will describe the specific approach to usability testing used in this work. The foundations described in this chapter will thus play a key role in the intend to improve the usability of the current system.

3. Reengineering Process (KN)

This chapter issues the fundamentals and methodology of the reengineering process in this thesis. Section 3.1 focuses on the process and explains the procedures of Inventory Analysis (3.1.1), Reverse Engineering (3.1.2) and Forward Engineering (3.1.3). Section 3.2 deals with the study design of the studies conducted in this thesis.

3.1 Approach

As no new software system is built from scratch but an already existing system is incrementally enhanced, we are dealing with a *Reengineering* Project. Incrementally adding or adjusting features and components is called the *Incremental* or *Forward* Engineering step. However, the current system CheckedInn 1.0 is not modeled in detail but consists mainly of the source code base. So, to be able to further extend the existing system, the status quo must first be derived. Therefore, a *Reengineering Inventory* is derived, which contains all existing functional and non-functional requirements of CheckedInn 1.0. The subsequent step of deriving the models of the current system is called *Reverse* Engineering. Reverse Engineering primarily involves identifying and understanding the existing classes, structures and interactions within the system at a structural level. This process provides a comprehensive model of the system's current architecture, functionalities, and dependencies. By thoroughly analyzing the system’s design and behavior, a detailed blueprint that forms the foundation for future modifications is created.
The main goal of Reverse Engineering is to model CheckedInn 1.0 as it is now and understand it to be able to subsequently extend it effectively. This understanding aids in ensuring that any new features or modifications for CheckedInn 2.0 align seamlessly with the existing system, thereby maintaining overall system integrity and performance.
When incrementally engineering the system, the focus lies on optimizing its user experience and usability. By conducting usability tests, the aim is to identify issues with CheckedInn 1.0 that may affect user experience and productivity. The focus is on gathering insights from hotel management, front office employees, as well as housekeeping staff and facility managers, to understand their interactions with the system and identify usability problems. The insights gained from these usability tests are analyzed to pinpoint specific areas where the system needs improvement. By addressing the identified usability issues, it is ensured that the enhancements made in the forward engineering step not only add new functionalities but also improve the overall user experience. This user-centered approach helps in creating a system that is functionally robust, intuitive, and user-friendly.

In summary, the re-engineering approach involves a systematic process starting with Reverse Engineering to understand the existing CheckedInn 1.0 system, followed by usability testing to identify and address user experience issues. Subsequently, the system is adjusted and extended by forward engineering. This structured approach ensures that the re-engineered CheckedInn 2.0 system is well-understood, meets the clarified requirements derived from usability insights, and provides an enhanced user experience through continuous incremental improvements.[8]

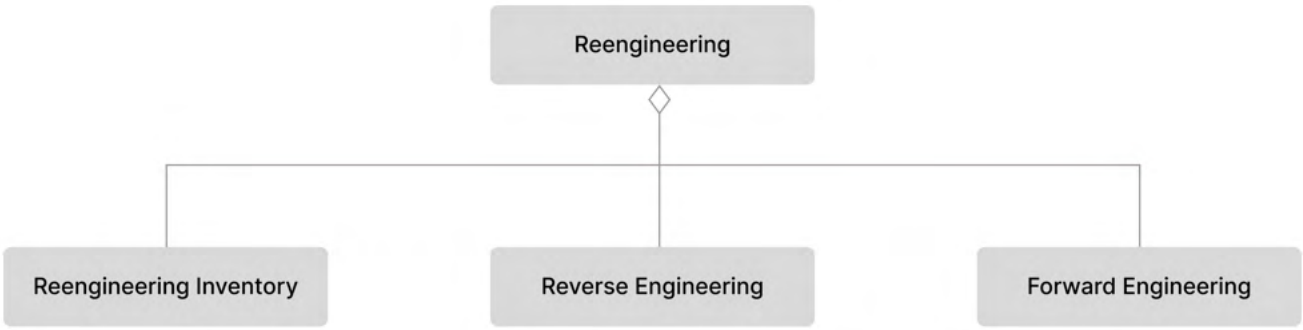


Figure 3.1: Reengineering Process (UML Class Diagram)

Figure 3.1 shows the reengineering process as an UML class diagram. Since it consists of the phases of reengineering inventory, reverse engineering and forward engineering, it is represented in the UML class diagram as an aggregation of these phases. The following chapters are structured as follows: Section 3.1 explains the reengineering process approach and its sub-phases. Accordingly, section 3.1.1 deals with the reengineering inventory phase, in which all necessary functional and non-functional requirements of the current system that are necessary for the reengineering process are captured. Section 3.1.2 explains how reverse engineering is performed, and 3.1.3 deals with how the forward engineering is performed. Subsequently, section 3.2 describes the process of discount usability testing, i.e. the phase of conducting usability studies.

The subsequent chapters 4, 5 and 6 deal with the results of the respective phase. Chapter 4 describes the derived reengineering inventory, in particular the existing requirements and non-functional requirements, Chapter 5 describes the results of the reverse engineering phase, in particular the composition diagrams, the object models and the use case models of CheckedInn 1.0. Chapter 6 describes the forward engineering. Since the actual implementation of CheckedInn 2.0 is mainly technical and the focus of this thesis lies on its usability, the focus of this chapter lies on the experimental evaluation of the usability tests.

3.1.1 Inventory analysis

The objective of the initial phase of reengineering is to document all functional and non-functional requirements of the current system CheckedInn 1.0 that are essential for the reengineering process. This involves both capturing and prioritizing these requirements. The requirements are collected through close collaboration with users and developers, as they know the current system's requirements, although these have not been formally documented previously. Additionally, all necessary artifacts, such as domain-specific constraints or relevant contexts, are also gathered and taken into account for identifying the existing requirements for CheckedInn 1.0. During this phase, several key activities take place:

1. Requirement Elicitation:
   - Conducting interviews with end-users to understand their needs and pain points.
   - Engaging with developers to extract undocumented knowledge about the system.
   - Categorizing requirements into functional and non-functional requirements.

2. Requirement Prioritization:
   - Assessing the priority of each requirement.
   - Categorizing requirements based on the scenario they belong to.
   - Collaborating with stakeholders to ensure alignment on priority requirements.

3. Artifact Collection:
   - Identifying and documenting domain-specific constraints that influence the system.
   - Recording relevant context information, such as regulatory requirements, operational environments, and user workflows.
   - Compiling all gathered data.

4. Analysis and Validation:
   - Analyzing the collected requirements to ensure they are complete, consistent, and feasible.
   - Validating the requirements with users to confirm their accuracy and relevance.
   - Updating the inventory document based on feedback and new insights.

At the end of this phase, the requirements for the current system shall be comprehensively documented. This establishes the starting point for reengineering and makes it possible to

understand which adjustments are necessary to improve the usability of the system. Then, reverse engineering will be used to create a model of the current system.

3.1.2 Reverse engineering

Deriving the model of CheckedInn 1.0 in it's current form lays the foundation for the further engineering of the system. This phase involves systematically gathering, documenting, and analyzing the architecture and model of the current system. The process of reverse engineering is structured into three main components: Understanding transformations, determining the system structure and analyzing the system behavior.

1. Understanding Transformations

The first step involves identifying the transformations the system must support. This is achieved through *Functional Modeling,* which includes the creation of scenarios and use case diagrams. Scenarios describe specific instances of system use, detailing the interactions between users and the system in particular contexts. Use case diagrams provide a graphical representation of these interactions, highlighting the system's functionality from the perspective of all involved actors. Actors are mainly three kinds of users, namely the guests interacting with the respective hotels, receptionists and hotel managers. Note that actors can also be other systems, or sub-systems. This is the case if the system interacts with other third-party systems. A prominent example is the Channelmanager (see chapter 5). To create accurate scenarios and use case diagrams, interaction with users happens through client interviews, observation of current practices and conducting of thought experiments. These activities help gather detailed insights into user needs and system requirements, which are subsequently used to optimize the usability and user experience. Section 3.2 describes how usability experiments are conducted and changes in usability are measured.

2. Determining System Structure

The system structure involves primarily creating class diagrams, that illustrate the system's static aspects, including the key objects that will constitute the system, their attributes, associations, multiplicities and operations. The main abstractions (classes) and their associations help describe how it's objects interact with each other. Thereby, these diagrams help in visualizing the system's architecture and ensuring that all necessary components are taken into account. This part is also referred to as *Object Modeling*.

3. Analyzing System Behavior

Understanding the system's dynamic behavior is done by *Dynamic Modeling*. This includes creating dynamic models and composition diagrams, which illustrate how the system operates over time. Composition Diagrams depict the way objects and subsystems of CheckedInn 1.0 interact with each other. Providers and consumers are identified, as well as the services, including their dependencies and concurrency, to illustrate the interactions between the objects. [8]

By structuring the reverse engineering step in this way, a comprehensive understanding of the current system's transformations, structure, and behavior is ensured. This facilitates the deriving of the current system’s model by reverse engineering. Subsequently it is the goal to formulate a set of activities to observe the usability of this system. Based on the derived model, the current system is extend to a robust and user-centric system called CheckedInn 2.0 by *forward* engineering. However, it is not only the models and system characteristics that need to be taken into account, but also the perspectives from which the users see our system. This means that the mental models of the users of the software should also be understood throughout the process of requirements analysis.

3.1.3. Forward engineering

After the model of CheckedInn 1.0 has been derived using reverse engineering in the previous step, the current system is to be further developed in this phase. Based on the model of the current system, the following activities happen throughout the incremental engineering phase:

1. Derive a dynamic model
   - Furthermore, a dynamic model is developed which defines the workflow of the forward engineering phase.
   - This model is event-based and is intended to enable agile further development of the system.
   - Implementing iterative cycles of development, testing, and refinement to adapt to changing requirements and feedback.

2. Injection of usability feedback subsystem
   - As the focus of this thesis lies on systematically improving the usability of the system, usability is not only the main new non-functional requirement, but even a usability feedback subsystem is introduced.

- The usability app is integrated into the system, and it's design of the usability subsystem is derived during this phase.
- The subsystem is then leveraged to conduct usability studies.

3. Deriving new functional and non-functional requirements
- Based on the results from reverse engineering, the new requirements for the enhanced system are collected.
- These can be new, complement the old requirements or revise them.
- Continuous interaction with users to refine and validate these requirements.

4. Forward engineer the system
- Next to only modeling a potential system, the aim is to actually extend and improve the current system.
- A fundamental actor in this process is the end-user, who gives feedback based on which the system is further developed.
- The feedback is evaluated using Discount Usability Testing.
- Agile methodologies are utilized to ensure continuous integration and delivery of new features.

As the process of conducting usability studies and evaluating user feedback with discount usability testing is an essential part of this thesis, it is explained in detail in section 3.2. In this section the participants, i.e. the end-users and the methodology of conducting the studies are introduced. Moreover, the concept of monitoring the user's behavior through logs to implicitly derive usability issues is also introduced.
This systematic approach lays the foundation to improve the system's user experience and transition from CheckedInn 1.0 to CheckedInn 2.0.

In chapter 6, the results of this phase, particularly the results of the usability tests, are elaborated on.

3.2 Usability Testing Methodology (SL)

As a focus of this work is to improve the usability of the current system, usability studies are conducted to gain insights about potential improvements. The subsequent chapter describes the methodology applied in the usability studies conducted. Section 3.2.1 initiates with an examination of the study participants, also detailing their diverse roles

within the testing framework. Section 3.2.2 proceeds to elucidate the study design, focusing on the mechanisms through which user feedback is gathered for usability assessment. Finally, Section 3.2.3 expounds on the methodological approaches utilized to analyze the data acquired from these studies.

3.2.1 Participants

In the initial phase of the project, the focus was on acquiring small to medium-sized hotels. This selection was strategic, as small hotels have shorter communication channels and clearer corporate structures. In addition, it is easier to reach the employees who work directly with the software in smaller establishments. These aspects are of crucial importance when conducting usability studies, which require direct interaction with the end users.

The direct involvement of the research team in the creation and implementation of the studies enables precise data collection and the generation of applicable results. The hotels included in the study are located throughout Germany, with a geographical focus on the Hamburg region. This concentration is due to the headquarters of the main client Prio Hotels in Hamburg. It offers logistical advantages and creates the opportunity to tailor the services specifically to the needs of this key client.

The targeted selection and close contact with hotels in Hamburg and other regions of Germany are aimed at creating a solid foundation for gathering user feedback. 13 hotels take part in the usability studies. Of these, six are owned by Prio Hotels GmbH, three by WM Hotels GmbH, three by the NinetyNine Hotels Group and finally one independently managed hotel.

- **Hotel Alter Wolf**, Wolfsburg [21]
- **Hotel Austria**, Stuttgart [21]
- **Hotel Condor**, Hamburg [21]
- **Hotel Design Apart**, Düsseldorf [21]
- **Hotel Fürst Garden**, Dortmund [22]
- **Hotel Home**, Dortmund [22]
- **Hotel Maximo**, Pinneberg [23]
- **Hotel NinetyNine Amsterdam Airport**, Amsterdam [24]
- **Hotel NinetyNine Augsburg**, Augsburg [24]

- **Hotel NinetyNine Hamburg-Bergedorf**, Hamburg [24]
- **Hotel Nord**, Hamburg [21]
- **Hotel Richter**, Hamburg [22]
- **Hotel Ritter**, Braunschweig [21]

All of these hotels have between 15 and 50 rooms and around 10 to 15 employees. Of these employees, around 3 to 5 per hotel are actively involved in the usability studies. Although it is primarily these selected employees who are directly involved in the studies, it is possible to collect comprehensive feedback from all hotel employees (see section 3.2.2).
The selection of test participants focuses primarily on front office employees, as they have a significant influence on the operational business and work intensively with the software on a daily basis. This makes it possible to identify relevant usability problems that have a direct impact on day-to-day work. In addition to the front office staff, there are specific functions within the software that are used exclusively by the hotel managers (see chapter 5). These functions relate among others to key operational figures such as sales figures, which are essential for strategic planning and management. To test these special functions, usability studies are therefore also carried out with the managers to ensure that this user group can also work efficiently with the software.
In addition, housekeeping staff are also included in the usability tests. Although the range of functions used by these employees is relatively small, it is crucial that these aspects of the software are also thoroughly tested in terms of their user-friendliness.
Software integration in the hotels takes place in planned phases. Once the software has been integrated into a hotel, a phase of several weeks of use follows before the usability tests begin. This gradual introduction is based on recommendations from Nielsen, who points out that users who have already gained experience with the software can provide higher quality data for the studies (see section 2.2.1). After this initial phase, each hotel has an optimal number of testers to carry out an effective iteration of discount usability testing. Details of this procedure are explained in more detail in section 3.2.2.
The initial integration of the software will take place at the Prio Hotels GmbH, ensuring a systematic and controlled introduction within a familiar corporate framework.

3.2.2 Study Design

The following subchapter describes the study design and deals primarily with the question of how feedback is obtained. Two methods are presented for this purpose.

The usability of a property management software is of very high priority due to its daily use and the high dependency of day-to-day operational and administrative business in hotels. However, the quality of such a system is not only measured by its technical performance. It is also particularly important how intuitively and efficiently the users, i.e. hotel managers, housekeeping staff and receptionists, can carry out their tasks. Gathering user feedback therefore aims to improve the usability of CheckedInn 1.0. To this end, a proactive approach in the form of a usability study is being pursued. On the other hand, the possibility is provided to give feedback directly via the software during daily use via a feedback form.

The study design for the usability tests of the current system is precisely geared towards simulating a realistic user experience and obtaining valuable feedback from end users. The tests are conducted in the form of scenario-based tasks that aim to replicate the everyday challenges and tasks that users encounter in a hotel environment.

At the beginning of the usability study, each test participant receives a task sheet containing a sequence of specific tasks that the test participant must perform as part of the test. The contents of these task sheets are tailored to the respective role of the tester. For example, a hotel manager is given tasks that focus on monitoring and managing rates, prices or finances, while receptionists are given tasks on check-in and check-out processes and housekeeping staff are given tasks on room status updates. This specific focus of the tasks ensures that the feedback from the tests is directly aligned with the respective usage patterns and needs of the different user groups of the PMS. An excerpt from a task sheet can be found in figure 3.2.

CheckedInn - Usability Study

Usability Study Scenario
Participant

1. Set the price for bed + breakfast in a double room for the next weekend at € 130 per night.
2. Create a reservation with the rate bed + breakfast in a double room in the name of Max Mustermann for Thursday to Sunday next weekend. The guest's email is max@mustermann.de.
3. On the night from Friday to Saturday, the Max would like to stay in a single room.
4. Change the address of Max and add a billing address.
5. Max would like to check in online. Prepare the online check-in. The room code is 3852 and the check-in email should be sent to the guest's email address.
6. Max has the status VIP. Let housekeeping know that they should provide the guest with a fruit basket and champagne in the room.

Figure 3.2: Excerpt from the task sheet for receptionists

During the test, participants are encouraged to think aloud, as it provides real-time insights into users' thought processes as they interact with the system (see section 2.2.2). The

facilitator present notes down this verbal feedback, which is later used to gain detailed insights into the user experience and identify specific areas for improvement.

After completing the practical tasks, the testers fill out a feedback form based on the ten usability heuristics developed by Jakob Nielsen (see section 2.2.1). Each heuristic is covered by three to four specific questions aimed at gaining a comprehensive understanding of the user's experience with the system. In addition, there are two free text blocks for each heuristic in which the testers can note down specific issues and recommendations. A sample question block on the heuristic *Aesthetic and Minimalistic Design* can be taken from figure 3.3. The entire questionnaire can be found in Appendix A.

8. Aesthetic

	Strongly disagree				Strongly agree
The interface is clutter-free, focusing only on necessary elements	1	2	3	4	5
The user interface does not distract from solving tasks	1	2	3	4	5
The look and feal optimizes space and prioritizes task-related information	1	2	3	4	5
Aesthetic elements do not interfere with the functionality of the system	1	2	3	4	5

Issues & Recommendations

Figure 3.3: Excerpt from the questionnaire for receptionists

This qualitative data provides deep insights into the practical use of the system and allows contextual suggestions for improvement to be gathered alongside the quantitative ratings from the questionnaire. At the end of the questionnaire, there is a general block with weaknesses, recommendations and aspects that the tester noticed positively about the current system. The quality and reliability of the data collected play a very important role in obtaining user feedback and increasingly pose major challenges [25]. To ensure this, all usability tests are carried out on site. The direct interaction between testers and facilitators during the tests enables immediate clarification of ambiguities and response to special incidents, which would not be possible in this form if the tests were carried out online.
However, this method of obtaining feedback also has some limitations: Firstly, time restrictions on testers and facilitators mean that the amount of feedback that can be obtained is limited. On the other hand, the procedure also restricts the scope within which feedback can be given. In terms of content through the task sheet and in terms of time

through the time window of the test. For this reason, an additional opportunity to provide feedback is offered.

Users can access a feedback form from anywhere in the software with a single click. All users were made aware of this option and the relevance of the feedback in advance in order to maximize their willingness. Problems that have arisen can be communicated to the developers via this feedback form. The completion of a free text field is mandatory. The problem that has occurred is described here. In the example in figure 3.4, for example, it is a filter functionality that is not working correctly from a technical point of view. In order to

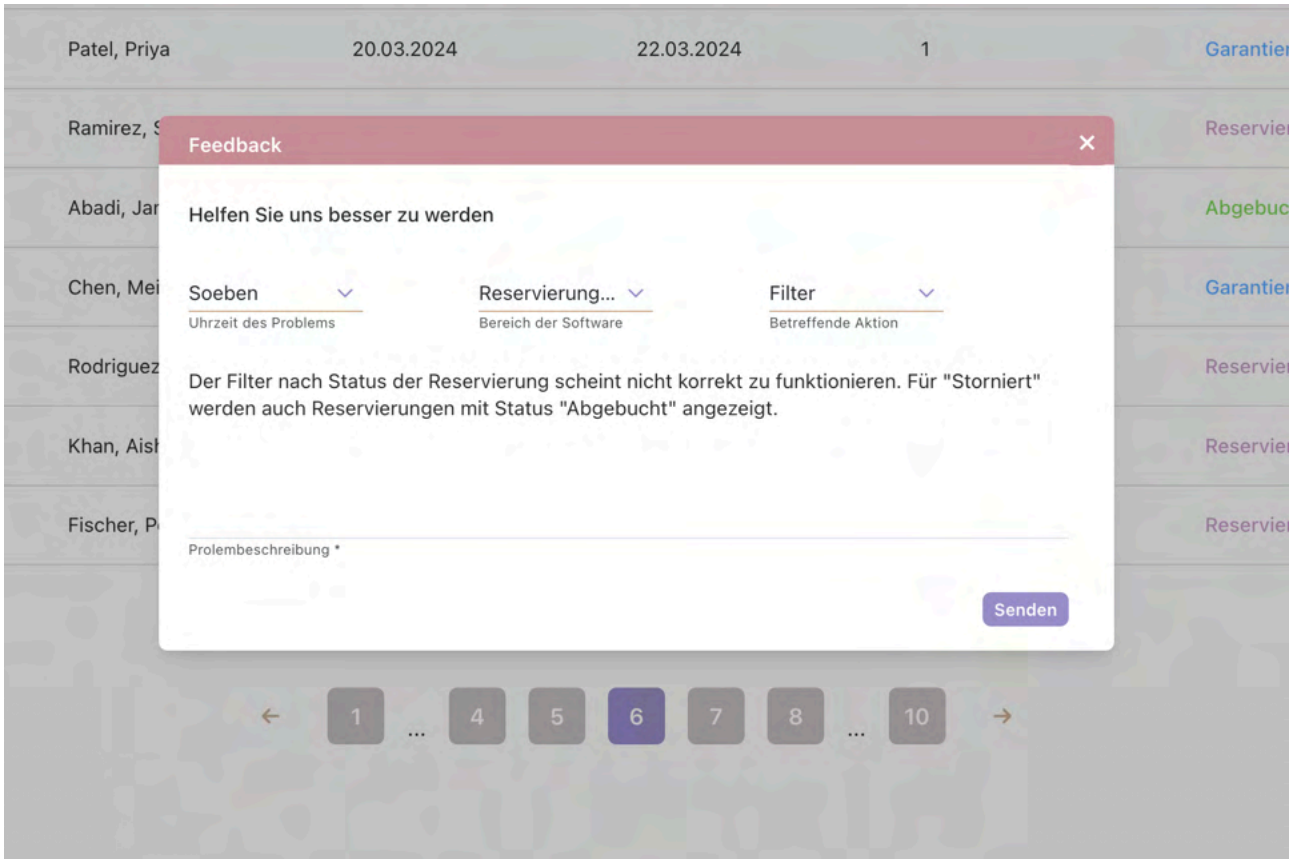


Figure 3.4: Feedback form of CheckedInn 1.0 with exemplary feedback

relieve the user of work and directly specify the relevant scope of the feedback, it is possible to do this via two drop-down menus. In the first of the two menus (middle of the three menus in figure 3.4), the relevant page can be specified, in this case the reservation overview. in the second menu (third menu in figure 3.4), the relevant actions are suggested directly. This procedure makes it easier to assign the feedback to an action and thus make it easier to understand. Finally, the user can also specify a time when the problem occurred. This provides several advantages. It makes it clearer to what extent several problems are related to each other. If there is feedback from several users at a certain time that indicates connection problems, it is immediately clearer to the developers that there is a technical problem with the server application. The biggest advantage, however, is that when the

feedback is sent, the user's logs for the specified time can also be sent. This enables the evaluators of the feedback to better understand the steps leading up to the problem. Specifying the time therefore also allows the feedback to be reported late without compromising the quality of the feedback.

These two methods offer an ideal mix of flexibility in providing feedback for the user and ensuring quality for the developers. As part of this work, data was collected over a period of nine weeks, divided into five iterations. In each of the iterations, users from two to three different hotels participated.

3.2.3 Feedback Evaluation

This section focuses on thoroughly examining the flow and, in particular, the processing of incoming feedback. The feedback system is characterized by its agility, interactivity and event-based nature. It is supported by a special object model, which is illustrated in detail in Figure 3.5. This model is complemented by the process lifecycle model shown in Figure 3.6.

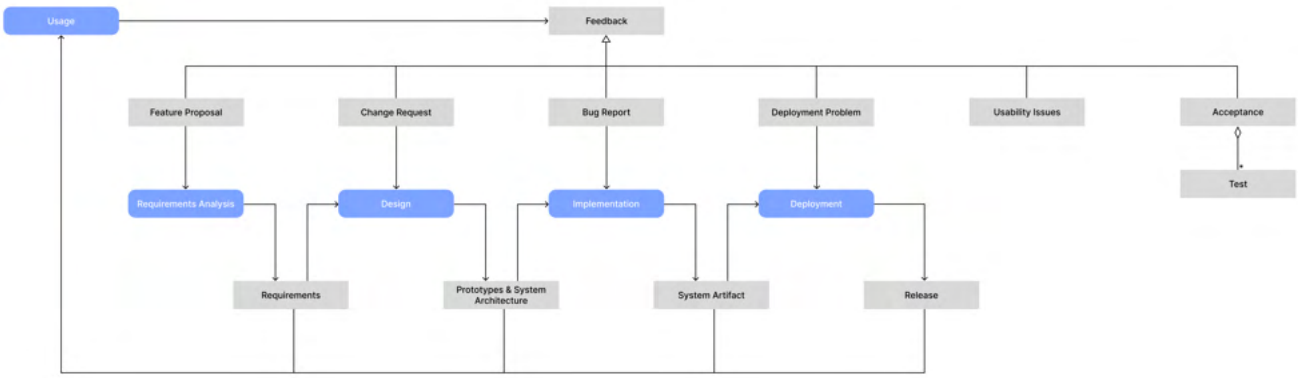


Figure 3.5: Continuous Feedback Object Model (UML Class Diagram)

The continuous feedback model, which forms the basis of the feedback process, comprises five main activities: Requirements Analysis, Design, Implementation, Deployment and Utilization. Each of these activities leads to utilization. This is a source of feedback that is used by the developers during and after each phase. In the context of these studies, this form of feedback is specifically the feedback resulting from day-to-day use. The developers receive this via the feedback form. This structure makes it possible for feedback to be generated and integrated not only after the completion of an activity, but also continuously throughout the entire development process. To ensure an efficient processing, the feedback is divided into six categories. These include:

Feature Proposal

A feature proposal describes a suggestion for new functions or a new feature. It aims at improving the functionality of the existing system.

Change Request

Suggestions for changes or enhancements to existing functionalities. These can involve modifications to features or configurations.

Bug Report

This category includes the identification and reporting of bugs or other implementation issues that affect the functionality of already existing features.

Deployment Problem

This includes any technical difficulties that may arise during the connection or deployment of the software as well as performance issues.

Usability Issue

This category includes all feedback relating to the usability of the system. This is about how user-friendly and intuitive the application is for the end user.

Acceptance

Acceptance can be achieved through testing to assess how well the product is evaluated by the user. It can be understood as an aggregation of tests with positive feedback.

Figure 3.6 shows the dynamic behavior of the feedback retrieval process. It covers both, the on site usability tests as well as the feedback via the feedback form in the software itself. The process consists of three swim lanes. For the developers, the release manager and the user respectively. During usage of the system issues can be reported via the feedback form at any time and are collected by the release manager. Depending on the type of problem reported, a specific activity is initiated. A feature proposal leads to the analysis activity in order to analyze requirements necessary to integrate the new ideas. A change request leads to a redesign of certain system components in order to implement the desired changes. When a bug report is submitted, the implementation phase is retriggered to fix the reported bug. Similarly, deployment issues may require reexamination and adjustment of deployment activities. Reported usability issues lead to further usability testing executed by the developers internally first. Another implementation phase can then be triggered.

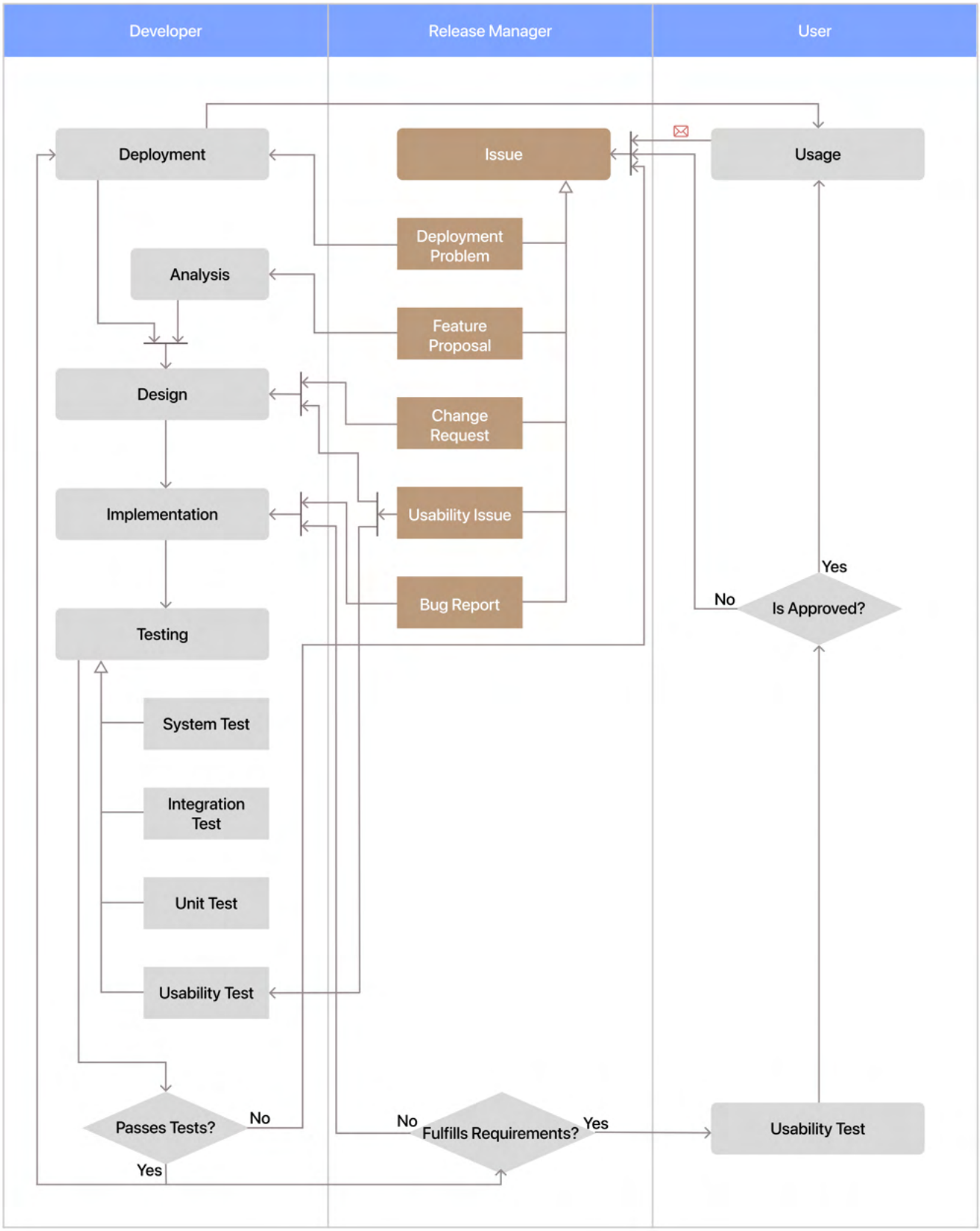


Figure 3.6: Process Lifecycle Model (Swim Lane Diagram)

Incoming feedback is classified using an algorithm that interfaces with ChatGPT 4.o. The use of large language-based models (LLMs) provides considerable flexibility in dealing with the large amounts of feedback generated by users. In order to preserve the authenticity and quality of the feedback, the LLMs deliberately refrain from automated prioritization of topics or semantic grouping. Instead, all feedback elements are evaluated and prioritized individually and according to their original form to ensure that all user contributions are taken into account appropriately.

Once the phases triggered by the feedback have been completed, a further internal test phase is carried out for quality assurance. The testing phase consists of four tests. Unit testing, integration testing, system testing and usability testing. They are performed for every release. The tests are designed to verify that the system meets all previously established requirements (regression testing) as well as any new issues that have been identified and managed in the backlog by the release manager. Some of these test cases are directly derived from user-defined use cases, which are crucial for aligning the tests with actual user needs and requirements. Others are developed based on insights gained from usability testing. Thus they ensure that the tests are profound and reflect real-world usage scenarios.

Unit tests describe the smallest unit of tests, which only test a single object with predefined methods. The main goal of unit tests is to validate the functionality of specific segments of the codebase. They should confirm that each component works independently as expected.

After successful completion of unit testing, integration tests are performed, where several components of the system are combined and tested together. This involves integrating parts of the file system with the scheduler to test the operation of the combined elements. These tests are particularly important in the context of the entire system, as they help to identify any problems that may arise as a result of interactions between the various components of the system. In this way, they ensure that the integrated system functions coherently.

Unit and integration tests are followed by the system tests. System tests are a type of software testing in which a complete and integrated system is tested against specified requirements. System tests are aimed at evaluating the behavior of the entire system under conditions that are as close as possible to real-world operating conditions. However, a complete simulation of real conditions is not always possible in the developer's environment. This makes the subsequent tests in the user's environment all the more important, as they provide information not only under different conditions, but from the user's perspective.

The last set of tests are usability tests. Unlike the previous tests, usability tests are not only conducted before a new release. They are also event-based conducted specifically in response to user feedback about usability problems. Usability tests are designed to focus on specific scenarios in which usability problems have been reported. This way, they provide the possibility of assessing the system performance under conditions that have led to user dissatisfaction.
As soon as all tests in the developer environment have been successfully passed, the deployment is carried out by the developers. In this phase, the release manager takes on the task of carefully checking whether the tested features meet the requirements. If this is not the case, the unimplemented or insufficiently developed features are further processed and implemented to ensure complete fulfillment of the requirements. If the features fulfill the requirements, the process moves to the usability testing phase in the user's environment. These are the questionnaires, conducted together with a facilitator as described in section 3.2.2. Usability tests are comprehensive and aim to cover all aspects and objects of the system. Similar to system tests, usability tests are conducted in the actual user environment but with a stronger focus on usability. This ensures that the system is tested under real-world conditions and validates its full intended purpose. After the usability tests, the facilitator plays a crucial role by entering all reported issues into a feedback form. This way, the classification algorithm gets triggered and respective activities are reinitiated based on the type of issue encountered. If no issues are reported during a usability testing iteration, the user can proceed with using the new version of the system.

Technically, feedback retrieval is solved via an external usability app (see composition diagram in Appendix B). This app provides an activity logging service in the event that a user provides feedback directly via the feedback form. The user's logs are sent to the usability app, where they are linked to the feedback message in the usability feedback subsystem and processed using an algorithm for classification. In particular, the category of feedback (feature proposal, change request, bug report, deployment problem, usability issue) is worked out here. The release manager has the option to check the classifications made by ChatGPT 4.o in order to avoid false classifications. Developers and system administrators have the ability to view, monitor, and analyze the user feedback through a dedicated administrator app. The feedback is displayed chronologically and can be filtered according to the type of feedback or the hotel from which the feedback was submitted. From here, the feedback can then be forwarded to the responsible developers.
This concludes the feedback and according release cycle, after which the process restarts with a new deployment phase for the next release. This continuous feedback loop ensures

ongoing improvements and incremental enhancements to the system, optimizing its usability and user experience over time. After the description of the methodology, the following chapter will deal with the inventory of the current system CheckedInn 1.0. In this phase, the current state of the system is examined in detail in order to create a solid foundation for future development cycles of CheckedInn 2.0.

4. Inventory of the current system (KN)

After having introduced the approach of the reengineering process in the last chapter, the respective results are presented in the following chapters. In this chapter, the reengineering inventory, which consists in particular of the functional and non-functional requirements the current CheckedInn 1.0 system fulfills, is elaborated on. As stated in section 3.1.1, this also includes their prioritization. Sections 4.1 and 4.2 list the collected and already prioritized requirements. The respective prioritization of each individual requirement are not explained in detail, but with a design rational, a general explanation of which aspects have been particularly relevant in the context of CheckedInn 1.0 is provided.

4.1. Functional Requirements

FR1: Create and manage reservations

CheckedInn 1.0 allows employees to create and manage reservations. This includes specifying a room, adding and managing (additional) services, checking-in and checking-out guests, adding special guest preferences, and adjusting all these details as needed.

FR2: Manage occupancy

CheckedInn 1.0 provides an overview of all reservations within a specific time range via a graphical user interface (the occupancy plan), giving employees a condensed overview of occupancy and allowing them to manage reservations and the status of rooms. The latter includes marking rooms as clean or dirty, which facilitates the check-in and check-out process.

FR3: Create and manage bills

CheckedInn 1.0 allows employees to dynamically create and adjusts bills for a reservation according to the guest's preferences. Moreover, it supports exporting bills as PDFs and sending them to guests by E-mail. The legal requirements for financial transactions and invoices are complied with in all these processes.

FR4: Guest correspondence

CheckedInn 1.0 ensures a seamless and convenient guest correspondence for both employees and guests. This includes keeping guests up to date with all relevant information about their reservation, allowing guests to check-in and out online from

their own mobile device, sending bills automatically via email and enabling employees to directly communicate with guests by email.

FR5: Online Check-In and Check-Out

CheckedInn 1.0 gives guests the option to check in and out online at any time using their own devices. In particular, CheckedInn 1.0 records all relevant personal information required to create the legally required documents (Meldeschein) and verifies these with the guest's digital signature. CheckedInn 1.0 also creates and stores the respective documents and links them to the corresponding reservation. Additionally, the guest receives all the information required to check in and out (such as check-in pins, check-in times) in digital form.

FR 6: Record and process payments

CheckedInn 1.0 supports the direct recording of payments. These are entered by the employee in the respective reservation and automatically stored in CheckedInn 1.0. They are then available both at the micro level within the respective reservation and at the macro level throughout the entire system in order to create cash point statements and revenue reports. Furthermore, they are a fundamental component of the consistency checks, to check whether a guest has been billed and can therefore check out. During each financial transaction, the legally applicable requirements are complied with. In particular, this includes the digital signature of each transaction using a verified technical security system (TSS).

FR 7: Handle ad-hoc tasks (traces)

CheckedInn 1.0 supports the creation, tracking, and management of ad-hoc tasks, ensuring that special guest requests or maintenance issues are completed promptly. These can be easily managed via a kanban board. Traces can be assigned to a reservation and a responsible person and provided with a due date. Responsible persons can be front-office employees, housekeeping or a facility manager. With the latter two, the traces can be accessed via the respective subsystem apps.

FR 8: Provide compressed information for daily business

CheckedInn 1.0 offers a dashboard that provides an overview of relevant information for daily business. In particular, this includes occupied rooms, arrivals, overnight stays, departures, special requests and free rooms. It also provides the most important key figures for daily business, namely an occupancy forecast and the average revenue per room.

FR 9: Generate and export lists for daily needs

CheckedInn 1.0 generates and exports (as PDF) lists for daily operations, such as housekeeping schedules, facility manager tasks, catering requirements, and arrival and departure lists to assist with front desk management. Dynamic processes are automatically incorporated into the creation of these lists, e.g. if a room is marked as clean by a housekeeping employee in their external application or an additional trace is assigned to a room, the lists automatically update accordingly.

FR 10: Manage debitor payments

CheckedInn 1.0 tracks outstanding payments from guests, sends reminders, and updates payment statuses to ensure timely collection of dues. The reminder system is divided into several stages, and the hotel manager can set the period of time after which the next stage and the corresponding steps are initiated if no payment is received. Any payments received can be entered by the employee and are taken into account accordingly in CheckedInn 1.0 in order to adjust the outstanding amount and inform the guest in case their bill has been settled.

FR 11: Manage cash points

CheckedInn 1.0 enables the management of cash points by supporting handover procedures at shift changes, (daily) closing operations, and ensuring all cash transactions comply with financial regulations. In the event of a handover or closing, CheckedInn 1.0 creates and stores internal receipts that process and document all transactions within the respective period. In the case of a cash point closing, a digitally signed version of the closing is created and transmitted to the DSFinV-K Api in order to comply with the applicable legal requirements.

FR 12: Synchronize with external booking platforms

CheckedInn 1.0 integrates with third party channel managers to connect with popular booking platforms like Booking.com and Expedia to synchronize room availability, rates and prices, ensuring reservations made through these channels are automatically updated in the hotel's system. This also includes a seamless integration into the property management system, so the employees can easily set availabilities and prices in CheckedInn 1.0 and they automatically synchronize with the respective channel managers. Additionally, CheckedInn 1.0 allows employees to conveniently set restrictions regarding the prices and availabilities of rooms. In

particular, this means they can specify a minimum and maximum duration for a guest’s stay or if no more arrivals or departures are permitted for a certain day.

FR 13: Generate reports

CheckedInn 1.0 generates detailed reports for revenues, guest analysis and compliance. This includes breaking down income by sales accounts, tax rates, payments, and time periods, providing insights into the hotel’s financial performance. Moreover, CheckedInn 1.0 also allows reports to be exported as PDF files.

FR 14: Automate company address finding

CheckedInn 1.0 facilitates the recording of company addresses through integration with a Places API. To do this, the employee enters the name of the company and receives all suggestions with matching companies. By clicking on the respective company, the address is directly adopted. Addresses can be captured either directly on an invoice or centrally stored in a guest's profile as a billing address.

FR 15: Customize the software

CheckedInn 1.0 allows users to customize various aspects, such as defining room types, services, user roles, and permissions to fit the specific needs of the hotel. This especially includes defining custom rates and services, sales account and billing rules, employee accounts, and custom templates for emails and documents. The latter in particular means PDFs for bills, reminders and offers. Moreover, CheckedInn 1.0 enables the setup of several additional services, such as the automatic generation of a QR-Code that can be used by guests to pay bills and the connection to third party channel managers (see functional requirement X).

All these functional requirements are met by the current CheckedInn 1.0 system, ensuring it can effectively manage the diverse and dynamic needs of day-to-day and administrative hotel operations. For the current system, the primary focus has been to ensure that all fundamental processes are performed effectively and efficiently. Therefore, implementing robust features that facilitate the daily operations of front office employees and hotel managers was of the highest priority.

While optimizing the overall system’s usability is the goal of this reengineering project, it is essential to maintain all previously mentioned functional requirements in the incrementally engineered system. Adjusting or extending features is not only allowed but explicitly welcomed if it enhances the system’s overall usability. However, any adjustments that

compromise existing functional requirements must be explicitly justified unless there is a change in user needs that necessitates such a requirement change.
In the following section the non-functional requirements are described in detail. They were derived in close collaboration with CheckedInn 1.0's end users to specify system qualities, which need to be fulfilled in order to effectively improve the employee's daily operations. Subsequently a justification for the choice and prioritization of the respective requirements is given.

4.2. Non-Functional Requirements

NFR 1: Security and Privacy

CheckedInn 1.0 protects the hotel's and guests' private data from unauthorized access or disclosure through appropriate security and privacy controls. This includes authentication and authorization mechanisms, role-based access control, and encryption.

NFR 2: Regulatory Compliance

CheckedInn 1.0 guarantees compliance with legal regulations and thus ensure that hotels can operate conveniently within the given laws. This refers in particular to the existing regulations regarding financial statements, financial transactions, cash point closings and data privacy.

NFR 3: Performance

CheckedInn 1.0 performs efficiently under varying loads, ensuring swift processing of bookings, check-ins, check-outs, and other transactions, even during peak seasons and high-demand periods to enable hotels to remain competitive in the face of strong competition.

NFR 4: Scalability

CheckedInn 1.0 is able to scale horizontally and vertically to accommodate increasing demands, such as additional reservations and transactions, but also new hotels as customers, without compromising performance.

NFR 5: Persistence

CheckedInn 1.0 ensures reliable storage and retrieval of data, preserving its integrity over time, so that guest histories, reservation records, operational data and especially financial documents are consistently accurate and accessible.

NFR 6: Maintainability

CheckedInn 1.0 is designed to facilitate easy updates, bug fixes, and enhancements, enabling developers to quickly implement changes and maintain optimal system performance with minimal disruption. This includes a component-oriented system development and shall enable to easily react to the ever faster evolving business environment.

NFR 7: Interoperability

CheckedInn 1.0 seamlessly integrates with relevant third-party applications and services commonly used in the hotel industry, in particular channel managers, financial regulatory software, tax statement software and customer relationship management (CRM) software. This includes APIs to connect to the respective services, as well as the functionalities to export data in desired formats for third-party organizations.

NFR 8: Responsiveness

CheckedInn 1.0 provides quick responses to user inputs, maintaining an optimal user experience for both guests and staff managing hotel operations through CheckedInn 1.0. Moreover, this also means that CheckedInn 1.0 provides a consistent and optimal user experience across various devices, operating systems, and screen sizes, ensuring that both guests and staff can access and use CheckedInn 1.0 effectively from all common end devices.

NFR 9: Flexibility

CheckedInn 1.0 is adaptable to the dynamic and complex processes and environmental conditions inherent in the hotel industry, such as managing fluctuating room availability, handling last-minute bookings, adjusting to seasonal pricing changes, catering to diverse guest preferences, and integrating new services or offers, ensuring smooth and efficient operations in a constantly changing environment. Also, the application is able to adapt to changing business requirements, and to integrate seamlessly with other systems linked to the property management software, such as external booking platforms.

NFR 10: Transparency

CheckedInn 1.0 ensures that all actions and transactions are visible and can be audited by authorized personnel, fostering a clear understanding of system operations and decision-making processes. This includes comprehensive logging of user activities, changes made to reservations, financial transactions, and administrative actions. CheckedInn 1.0 also provides detailed reporting capabilities, enabling hotel management to generate and review reports on various operational aspects, such as booking trends, occupancy rates, and financial summaries.

NFR 11: Traceability

CheckedInn 1.0 offers robust traceability features, allowing for the tracking and examination of all relevant critical operations and data changes within the hotel management software. This includes maintaining an immutable log of all transactions, modifications, and interactions, along with timestamps and user identifiers. Traceability ensures that the origins of any action or decision can be traced back to specific events or users, aiding in the investigation and resolution of issues. This is particularly important for financial operations, where compliance with regulatory requirements, internal audits, and resolving disputes is of high relevance. By providing detailed traceability, CheckedInn 1.0 supports operational integrity, security, and accountability, ensuring that any situation can be thoroughly analyzed and understood.

These are the most important non-functional requirements that the current CheckedInn 1.0 system fulfills. Other non-functional requirements, such as "Correctness" or „Effectiveness“, i.e. those necessary for a functioning software system, were also taken into account but are not mentioned explicitly here. All of the above non-functional requirements should not suffer from the improvement of CheckedInn 1.0’s usability. This explicitly means that all current non-functional requirements must continue to be fulfilled by the new CheckedInn 2.0 system during forward engineering.

As can be seen in the non-functional requirements, usability was not a non-functional requirement in the development of CheckedInn 1.0. Therefore, CheckedInn 1.0 also suffers from usability issues. The subsequent goal is to introduce usability as a fundamental non-functional requirement of the new CheckedInn 2.0 system and thereby improve its usability.

4.3 Design Rationale

Both the identification and prioritization of the functional and non-functional requirements were conducted in close collaboration with the end-users. The focus of CheckedInn 1.0 has been on capturing the fundamental workflow of daily hotel operations for all its actors, which is clearly reflected in the requirements and their prioritization.

This approach ensures that the core needs of the hotel staff and management are met efficiently, without overburdening the system with extensive, non-essential functionalities. The current system was designed to handle essential operations such as bookings, check-ins, check-outs, and financial transactions, providing a reliable and robust platform for managing day-to-day activities. This is reflected in particular in the functional requirements. In addition, the non-functional requirements reflect the characteristics that the system must fulfill. As a lot of personal and critical data is involved, such as guest information and financial data, the persistence and stability of these functions is of high priority. Moreover, there are many legal regulations that must be complied with. Therefore, regular compliance is also one of the most important non-functional requirements. Last but not least, as the hotel business is highly dynamic, non-functional requirements such as flexibility and responsiveness have to be taken into account in order to meet the challenges that employees face on a daily basis. As these requirements had not yet been formally documented to such an extent, this process was again carried out in close cooperation with the end users of the system.

5. Reverse engineering (SL)

Once the requirements have been derived in the reengineering inventory, the next step is to derive the model of the current system. To do this, use case models, object models and composition models are derived as described in 3.1.2. The following subsections detail the reverse engineering process, segmented into semantically related parts of the software. Each area is first illustrated using use case diagrams, class diagrams and composition diagrams and then technically reconstructed. These then serve as the basis for further developing the CheckedInn 2.0 system in Chapter 6 and carrying out the usability tests described in 3.2. Section 5.5 examines whether all the requirements derived in Chapter 4 are fulfilled by the use cases of the scenarios presented in this chapter. For this purpose, a representative use case is named for each scenario that fulfills the respective functional and non-functional requirements.

In the context of reverse engineering, the system was systematically analyzed and represented using an analysis object model. This model, depicted in Figure 5.1 as a UML Class Diagram, provides a visual representation of the system's structure, consisting of a set of classes. The classes are modeled together with their attributes, methods and associations. To understand the current system in detail the model is explained in the following. The key abstraction of the CheckedInn 1.0 system is the hotel class (colored in blue).

The fundamental flow of all actions regarding the management of a guest's stay evolves around the `Hotel` class. Another important abstraction is the `User` class, consisting of the subclasses `HotelManager`, `Employee`, `Guest` and `SystemAdministrator`. Each of these subclasses represents a different type of user within the system. While the Hotel Manager is responsible for the administration of a hotel, the employee is responsible for the operational processes of a hotel. The employee class again inherits the subclasses receptionist, housekeeping employee and facility manager. While the receptionist is responsible primarily for front-office tasks, the housekeeping employee takes care of room cleanliness and maintenance, and the facility manager oversees the upkeep of the hotel’s infrastructure, including repairs, safety checks, and facility improvements. A System Administrator, on the other hand, does not belong to a hotel, but manages the entire system. He alone holds the authority to create and modify essential details of the hotel, such as the room count. As the associations between hotel, hotel manager and employee make clear, a hotel can have several users and the users can also belong to several hotels.

Figure 5.1: CheckedInn 1.0 Object Model (UML Class Diagram)

Central to the guest experience is the `Reservation` class, which captures essential information such as `arrival`, `departure`, `price`, and `status`. Each reservation is linked to one or more `Room` objects, which detail the room `category`, room `name`, and the `isClean` attribute, indicating whether the room is clean and ready for guest occupancy. The relationship between `Reservation` and `Room` is modeled as a many-to-many association, acknowledging that a room can be associated with multiple reservations over time, and a single reservation can involve multiple rooms. The separate access for guests allows them to check in and out themselves.

Additionally, a reservation also requires a `Rate`, which is a predefined bundle of services, such as accommodation with breakfast. The `Rate` class includes attributes like `name`, `defaultPrice`, and `includesAccommodation`, specifying whether the rate covers overnight stays or other services (e.g., conferences). Each `Rate` is composed of multiple `Service` objects, each characterized by its `name`, `category`, and `price`.

The `Service` class plays a pivotal role in the system, with each service being assigned to exactly one `Bill`. In turn, the `Bill` class, which together with `Payment` represents another crucial part of the reservation scope, is linked to a single reservation. However, a reservation can have multiple bills. Each bill contains details such as `amount`, `recipient`, `tseId` (for legal compliance), and associated payments and services.

Finally, a reservation can include multiple guests. This is described by the many-to-many association between `Guest` and `Reservation`. Guests are differentiated by the attribute `isFellowTraveler`, which indicates whether the guest is a fellow traveler or the one who is booking the reservation. In the case of fellow travelers, only the first name and last name are required, and no address is assigned.

The last part of the fundamental workflow within a hotel is the internal billing and aggregation of sales. The `SalesAccount` class serves as the primary entity for this process, aggregating various services. Each `SalesAccount` groups services and assigns them a `tax` rate and a `datevId`, facilitating the export of sales-related data to Datev, the leading software for tax management and statements.

5.1 Reservation Scenario

The reservation scenario describes the comprehensive process from creating a reservation to executing all essential steps to ensure its completion. This crucial stage not only lays the foundation for managing reservations but also sets the stage for executing subsequent guest-related operations. Furthermore, the scenario encapsulates the critical processes necessary to facilitate and enhance guest stays within the system's operational framework. Figure 5.2 shows the reservation scenario visualized in the form of use cases and their

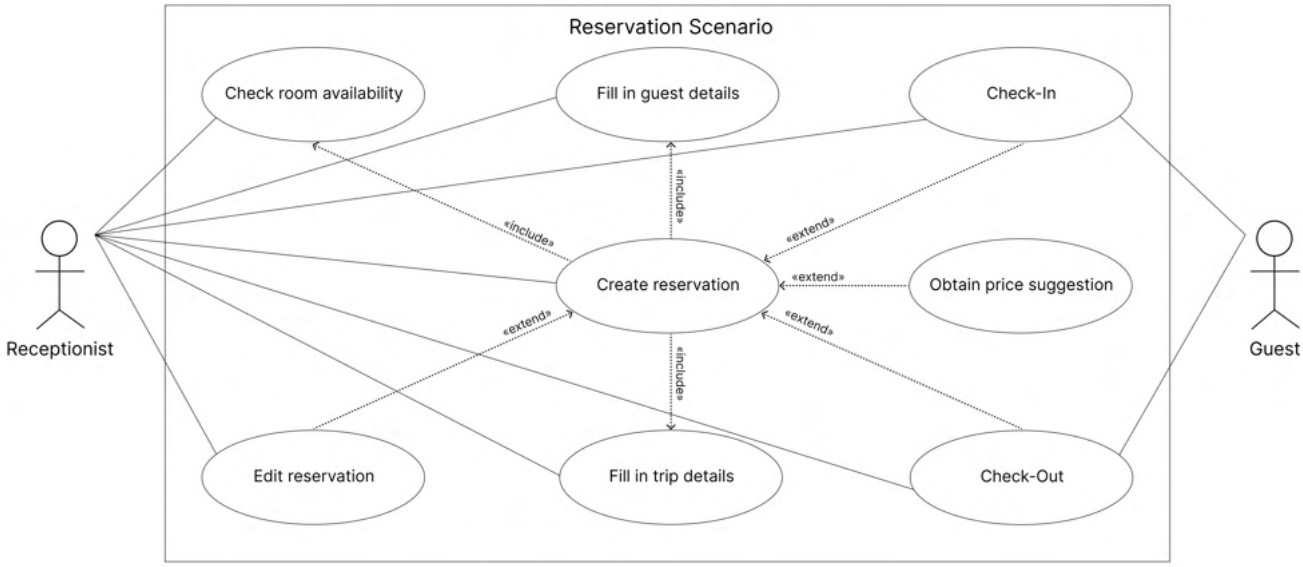


Figure 5.2: Reservation scenario for CheckedInn 1.0 (UML Use Case Diagram)

dependencies. It serves as a comprehensive visual representation of the workflow and the associated roles within the reservation process of a hotel management system.
The diagram shows two main actors involved in the reservation process: the receptionist and the guest. Each actor interacts differently with the system depending on their role and requirements.

1. `Receptionist`
   This actor represents the hotel front office staff responsible for managing reservations. The receptionists carries the right to create and access all reservation related information. He interacts directly with the system to ensure that all guest details are entered correctly and that the reservation details are up to date.

2. `Guest`
   The guest is generally not involved in the processes in terms of using the system. However, the hotel has the option of making certain actions accessible to the guest in order to automate processes.

Figure 5.3 illustrates the objects involved in the comprehensive object model of the current system. The actors in the use case diagram are linked to the inheritances of the user class, and all involved objects are subsequently described in relation to their respective use cases.

Figure 5.3: Reservation scenario excerpt of the CheckedInn 1.0 Object Model (UML Class Diagram)

The core use case in the diagram is `Create reservation`, which is central to the reservation scenario. This use case is the primary function performed by the Receptionist upon the Guest's request to book a stay at the hotel. Several other use cases are directly connected to this central activity. In particular, this use case addresses FR 1 and NFR 1. Since this also includes assigning rates and rooms, which have been customized by the hotel manager, this use case also references FR 15.

The `Check room availability` use case, which references FR 2, is a preliminary step in the reservation creation process where the Receptionist checks the availability of rooms as per the guest's request. This is crucial to ensure that the Guest's needs can be accommodated on the desired dates. This process is also essential on the part of the hotel to ensure that there are no overbookings and thereby reference NFR 5. Overbookings are to the financial disadvantage of the hotel. Once a room has been confirmed as available and was selected for the reservation to be created, the receptionist proceeds to `Fill in the`

`guest's details` within the system. Capturing accurate personal information is essential not only for reservation or legal purposes but also for personalizing the guest's stay, ensuring that all preferences and requirements are noted and met. To further personalize the service, the receptionist enters specific details about the guest's stay in the `Fill in trip details` use case. These include special requests, the status of the reservation, the number of people (pax) and optionally label the guest as a business traveler. Whether the visit is for business or leisure, understanding the nature of the trip helps in customizing the guest experience, enhancing their satisfaction with tailored services. The selection of a rate is also particularly important in this context. This includes the services offered to the guest and is therefore the most important determinant of the total price, alongside the room category and length of stay. As can be seen in figure 5.3, a rate has a default price. This is set when creating a rate and is multiplied by pax and the length of stay. With all this, these use cases address especially NFR 3.

The CheckedInn 1.0 system also includes a pricing component. This is used exclusively for pricing and overview for certain pax, rate and room category combinations. If an entry exists for the combination selected by the receptionist when creating the reservation, the suggestion to overwrite the defaultPrice with it is received as part of the `pricing suggestion` use case. Thereby NFR 9 is referenced.

Once the reservation has been created, the receptionists obtains the flexibility in managing reservations through the functionality that allows to `Edit the reservation`. Whether adjusting dates, room types, or any other details, this capability is essential for accommodating changes in the guest's plans or correcting any discrepancies in the reservation details. For providing a clear overview of all reservations the status is of high importance. It categorizes reservations into `reserved` (default status), `charged` if the guest has already paid, `option` if it is uncertain whether the guest will travel, `cancelled with payment` if the selected rate does not offer free cancellation and finally `cancelled free of charge`. Updating this attribute is also part of editing a reservation. With this in particular FR2, as well as NFR 3 and 9 are addressed.

The most obvious part of the flow for the guest describes the `Check-In` and `Check-Out` use cases. The check-in is relevant for the hotel in order to authenticate the guest against their reservation. The overview of all checked-in guests can also be used to track who has already arrived and who has not yet arrived. Check-Out, on the other hand, is important as this process must be used to track whether all services purchased by the guest have been properly billed. The guest's departure is also recorded in the system at this stage. This process is usually handled by the receptionist. However, as many hotels are increasingly automating interactions with guests, CheckedInn 1.0 also offers guests access. As soon as the reservation has been created, the guest will receive an e-mail with the necessary

access data for the guest application. Guests can then check themselves in and out without the receptionist having to intervene. These use cases reference FR 5, and NFR 8.

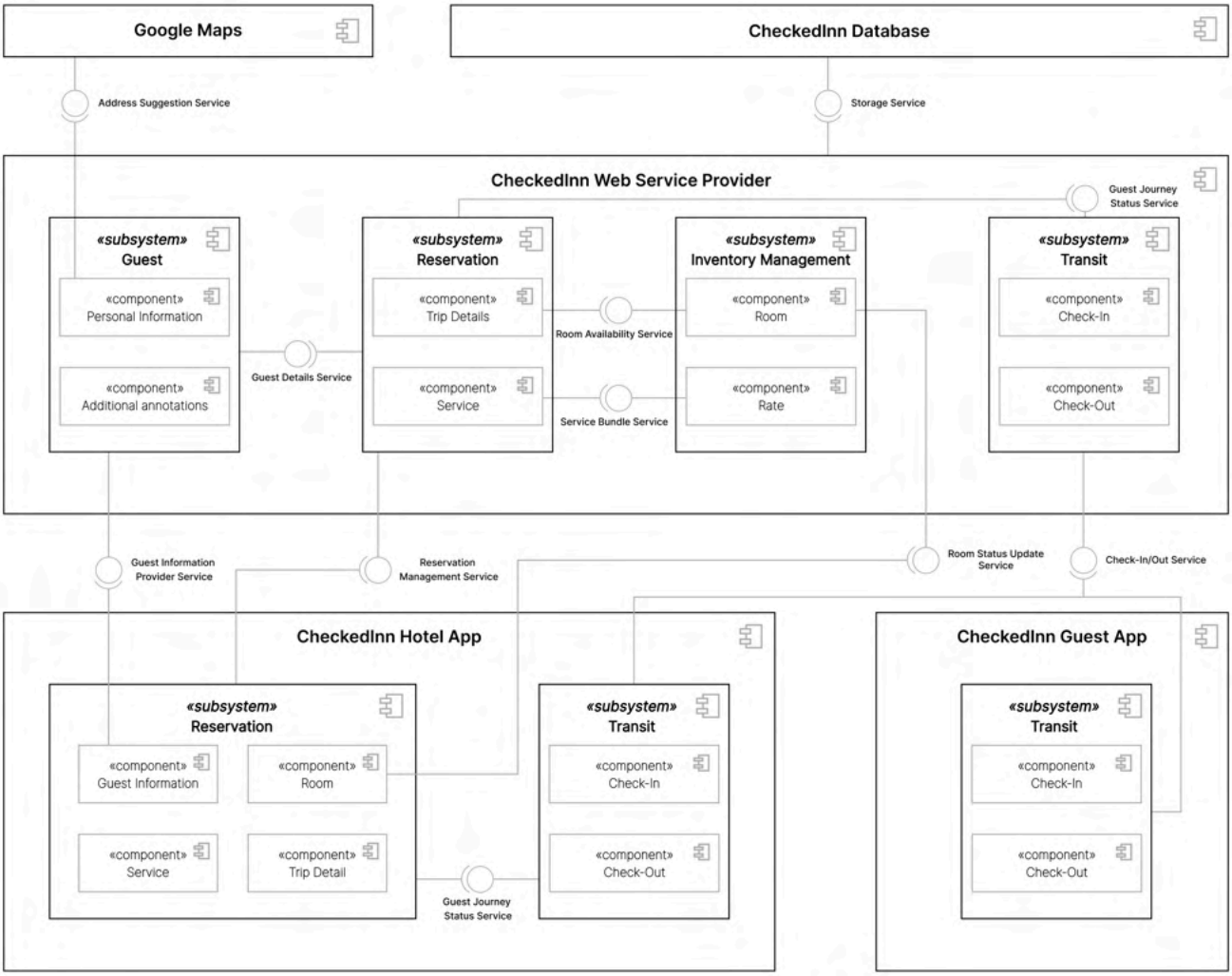


Figure 5.4: Reservation scenario excerpt of the CheckedInn 1.0 subsystems (UML Composition Diagram)

Figure 5.4 outlines the subsystem decomposition for the Reservation Scenario. The depiction is refined to correspond with the use cases described in figure 5.2. The subsystem decomposition of the entire system can be found in Appendix B.

The system is initially divided into five separate physical nodes: `Google Maps` as an external service, the `CheckedInn Database`, the server - represented here as the `Web Service Provider` - and finally, the two applications: the `Hotel App` for the hotel's receptionists and the `Guest App` for guests.

When creating a reservation, the availability for a selected period must first be checked as described by the Check room availability use case. For this purpose, the Hotel inventory subsystem offers a room availability service to the reservation subsystem. The hotel inventory subsystem manages the data that a hotel needs for its day-to-day business. In the context of reservations, this particularly affects the rates and the rooms with their properties and restrictions as for example the cleanliness status. The reservation subsystem in the web service provider, on the other hand, forms the logical processes for creating and updating a reservation, particularly with regard to the key data of the reservation and the services purchased by the guest. To set prices, the Rate component in the hotel inventory subsystem offers a Service bundle service that can be used to suggest rates and supplement them with the fixed default prices.

As part of the Fill in guest details use case when creating a reservation, there is the option to have addresses suggested based on partial inputs. This is particularly useful for hotels with many business customers due to longer address names. The Address Suggestion Service, which is provided by Google Maps and then displayed in the Guest Information component of the hotel app, is used for this purpose.

For the local management of reservations, the web service provider reservation subsystem provides a Reservation management service to the reservation subsystem in the hotel app. This service can not only be used to create, but also to Edit reservations as in figure 5.2 described. For example, if a receptionists wants to rebook a reservation for another guest, suggestions for other guests staying at the hotel can be made using the Guest Details Service. This is provided by the Guest Subsystem.

Especially on the day of arrival, it is important for the receptionist to be informed about the status of the room in order to check whether the housekeeping has already cleaned the room. For this reason, the room component of the inventory management subsystem offers a room status update service. This is consumed by the room component in the hotel app. As soon as the correct cleanness status of the room is ensured, the use case described as Check-In can begin.

This process is controlled via the Transit subsystems, which manage guest stay status data within the web service provider. The Reservation subsystem can access these data through the Guest Journey Status Service. Given that check-in and check-out functions embody distinct functionalities and require different prerequisites, they are represented as separate components within the Transit subsystem. This subsystem is also incorporated into both the Hotel App and the Guest App. Through the Check-In/Out Service provided by the global Transit subsystem, both hotel staff and guests can independently perform check-in and check-out procedures via their access to the system.

All data processed in the Web Service Provider is stored persistently in the `CheckedInn Database` via the `Storage Service`.

5.2 Trace Scenario (KN)

This section deals with another elementary building block that is needed to comprehensively model the day-to-day business of the hotel industry; the trace scenario[1]. Since the operational business is characterized by its dynamic nature, it often happens that unforeseen tasks arise spontaneously and have to be dealt with. Therefore, a powerful tool that facilitates the handling of ad-hoc tasks (traces) is of great importance.
Figure 5.5 shows the Trace Scenario, with it's associated use cases and actors as a UML Use Case diagram.

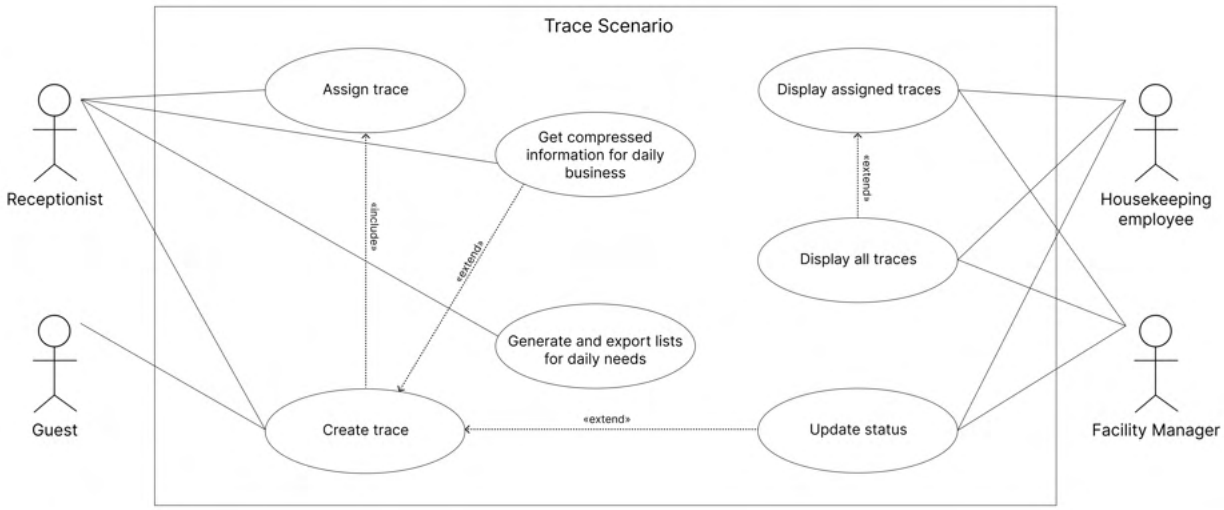


Figure 5.5: Trace scenario (UML Use Case Diagram) for CheckedInn 1.0

1. `Receptionist`
   In this scenario, the receptionist is responsible for managing the creation and assigning of traces. Therefore, he is provided compressed information about the traces for each day by the system, as well as the functionality to export them.

2. `Guest`
   Although the guest does not use the current system directly, he can also in this scenario

[1] Trace is the industry-standard term for an ad hoc operational task in the hotel industry.

interact via an interface. This allows them to create traces concerning their stay directly. This contributes to the digitalization and optimization of processes.

3. `Housekeeping employee`
   The housekeeping employee is an actor who is responsible for carrying out the operational tasks of cleaning the hotel rooms. To facilitate these tasks, access to and management of the respective tasks is extensively digitized by the current system.

4. `Facility manager`
   The facility manager is responsible for carrying out all operational tasks relating to the maintenance of the hotel and its rooms. Sudden defects also fall within his area of responsibility. His tasks are also digitized by the current system and thus made easier.

Figure 5.6 depicts a excerpt of the object model for the trace scenario as a UML class diagram. In the following, the use cases in connection with the participating objects will be discussed in more detail.
The first step in the trace scenario is the creation of a trace, which is represented by the use case `Create trace`. That this lays the foundation for further use cases is also illustrated by the include relations of `Assign trace` and `Update status` to `Create trace`. Traces can be created by both the receptionist and the guest. While the receptionist creates a trace like all tasks in the CheckedInn Hotel App, the guest uses the separate CheckedInn Guest App for this purpose. When creating a trace, the necessary information such as the title, an optional description and a due date are entered. As the basis for creating traces serves relevant information on the current day's situation. The receptionist receives this via the `Get compressed information on daily business` use case. All relevant data for ad hoc tasks of the current day, such as specific requests from guests or the number of breakfasts to be prepared, are provided in the form of a dashboard. In the use case diagram this is expressed by the include relation to `Create trace`. In addition to digitally assisted work environments or situations, these are often also required in printed form. The use case `Generate and export lists for daily needs` therefore describes, among other things, the creation of lists that contain such information about breakfasts, arrivals and departures. As a printed version of this information is essential for many hotels, it is possible to export the generated lists as a PDF and then print them.
Once a trace has been created, it can then be assigned to a responsible employee. This is described by the use case Assign trace. The employee responsible can be a housekeeping employee or a facility manager. The responsible employee receives a notification that the respective task has been assigned to them and is responsible for completing it. The

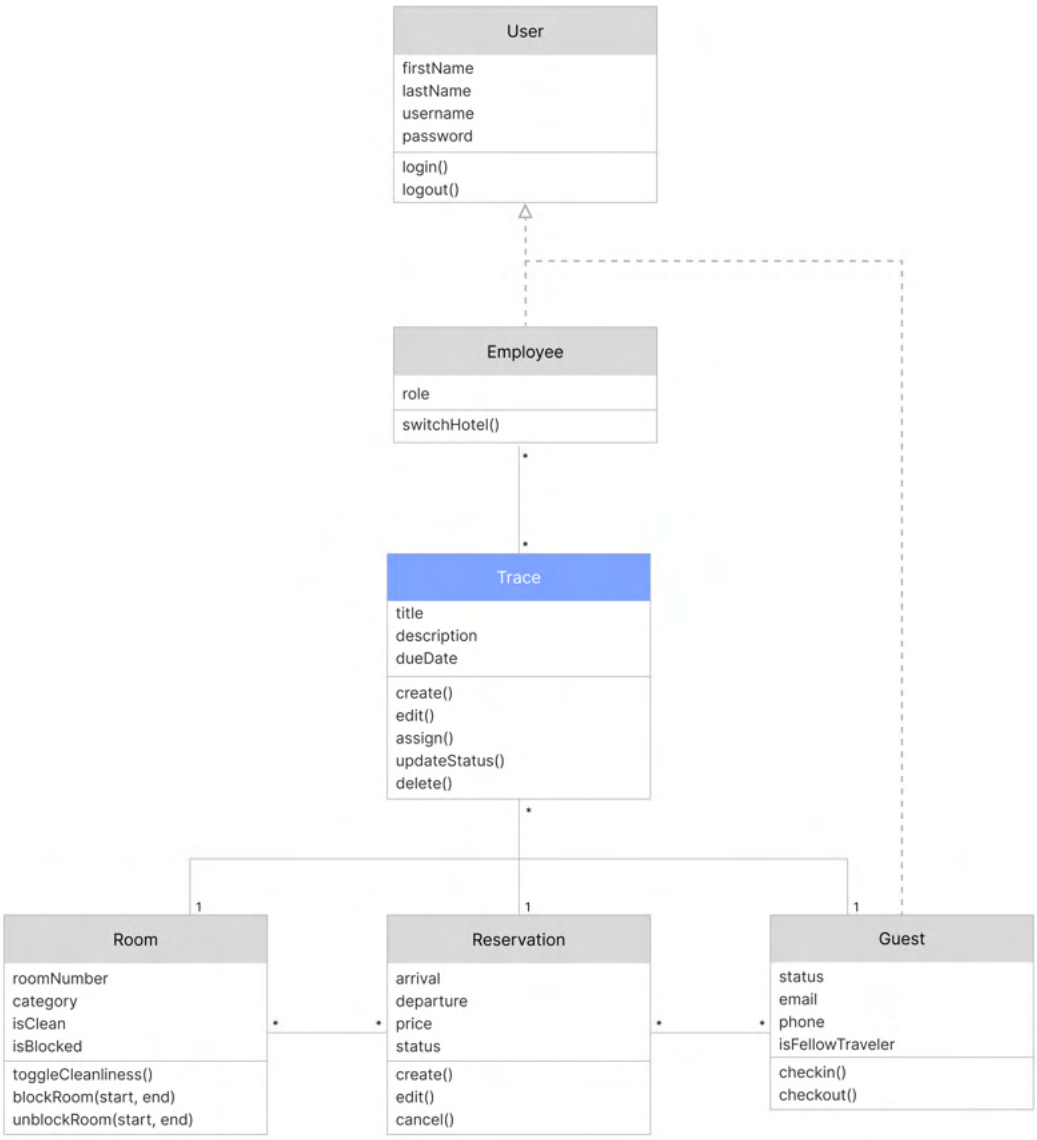


Figure 5.6: Trace scenario excerpt of the CheckedInn 1.0 Object Model (UML Class Diagram)

respective employee can view all the traces assigned to them in his respective app. This is described by the use case Display assigned traces. Optionally, this is extended by the use case Display all traces, in which all current traces are displayed to the employee. The purpose of this is to ensure that the current system does not take away the momentum of ad-hoc task execution and does not prevent coordination between employees as to which tasks are completed by whom. If a trace is being processed or has been completed, the status can be updated accordingly by the employee responsible. This is described by the use case Update status. The trace is then also updated in the system so that all other employees can see how the status of the respective trace has changed. This avoids potential duplicate processing.

Now that all use cases have been described, the subsystems of CheckedInn 1.0 relevant for the trace scenario shown in Figure 5.7 as a UML composition diagram are explained. The central subsystem in the trace scenario is the `Trace` subsystem in the `CheckedInn Web`

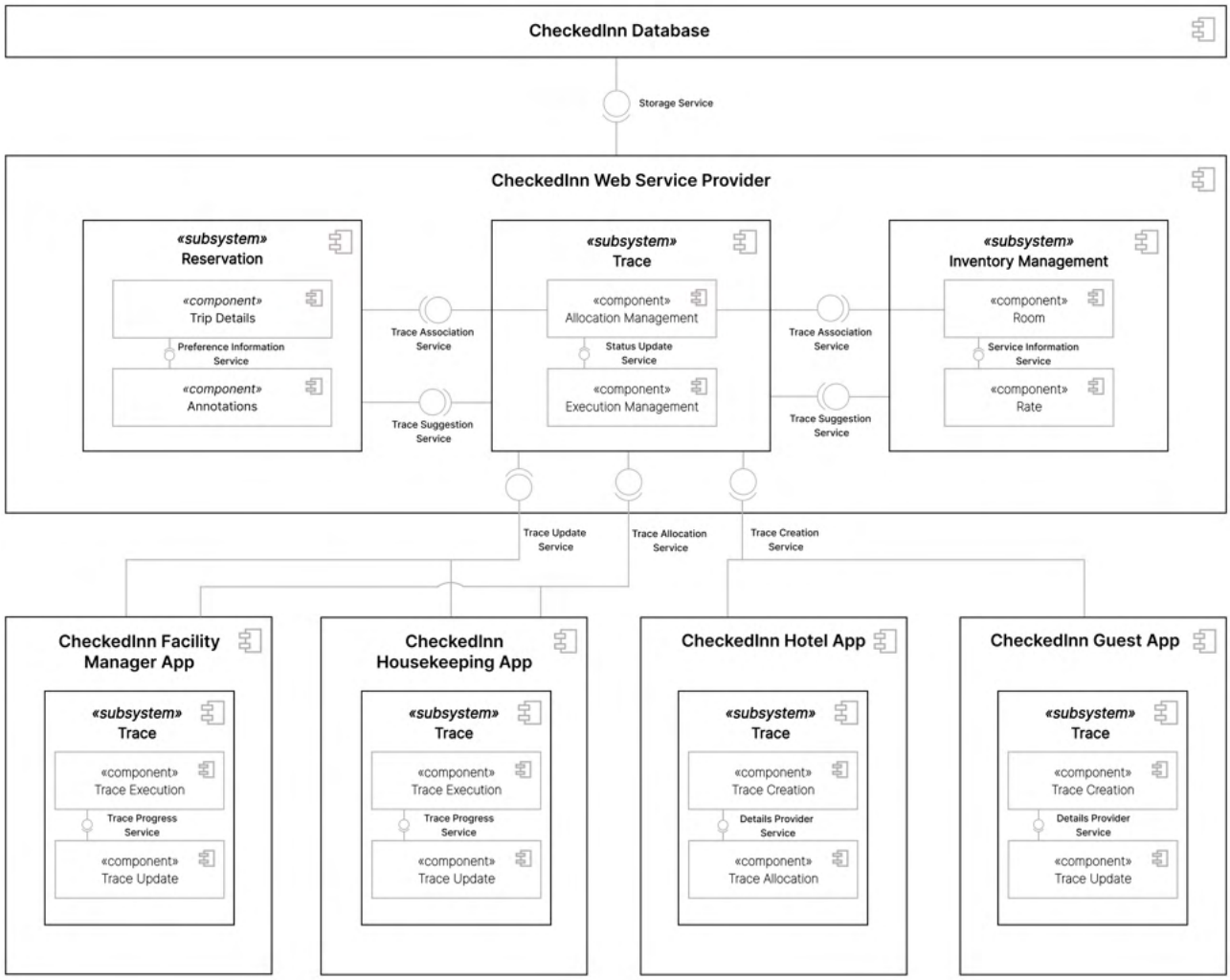


Figure 5.7: Trace scenario excerpt of the CheckedInn 1.0 subsystems (UML Composition Diagram)

`Service Provider` and in the `CheckedInn Facility Manager`, `CheckedInn Housekeeping`, `CheckedInn Hotel` and `CheckedInn Guest App`. The `CheckedInn Database` system is again used for the persistent storage of data. In the `CheckedInn Web Service Provider`, the trace subsystem consists of the `Allocation Management` and `Execution Management` components. `Allocation Management` is used to assign traces to the responsible employees using the `Trace Allocation Service`, which is provided to the `CheckedInn Facility Manager App` and the `CheckedInn Housekeeping App`. In addition, the component provides the `Trace Association Service` to the `Reservation` subsystem and the `Room` component of `Inventory Management` and thus links traces to the respective reservation or room. To obtain suggestions for traces to be created, the

trace subsystem consumes the Trace Suggestion Service from the reservation and inventory management subsystem. To generate suggestions for traces, the Annotations component provides the Preference Information Service to the Trip Details component in the reservation subsystem. This aggregates guest-specific information and suggests possible traces. In inventory management, the rate component provides the Service Information Service to the Room component. Information on booked services is used to generate suggestions for any traces that may be incurred.

The next important component is the management of traces by actors in their respective CheckedInn app. The administrative tasks are carried out in the CheckedInn Hotel App and partly in the CheckedInn Guest App. In both apps, the trace subsystem has a Trace Creation component. In this, parameters such as description, due date and, if applicable, an assignee are defined. The trace subsystem then consumes the Trace Creation Service from the trace subsystem of the CheckedInn Web Service Provider. This service generates the traces consistently for all systems that access the traces. Trace allocation is then carried out in the CheckedInn Hotel App. This is provided by the Trace Creation component Details Provider Service, which transmits relevant information for the allocation of traces. In the CheckedInn Guest App, on the other hand, the Trace Update component ensures that traces can also be marked as completed directly by the guest themselves. This simplifies the workflow and further optimizes business processes. To make these updates possible, the Trace Creation component provides the Details Provider Service to the Trace Update component. This provides the guest with all the necessary information to update traces accordingly.

The last participating subsystems are the CheckedInn Housekeeping and the CheckedInn Facility Manager App, which are responsible for the trace execution and corresponding status updates. For this purpose, the trace subsystem in the CheckedInn Web Service Provider provides the Trace Allocation Service to both subsystems, which ensures that information on the assigned traces is available in the respective apps. In the trace subsystem of both apps, the execution of traces is tracked with the Trace Execution component. The progress of the processing of a trace is provided to the Trace Update component via the Trace Progress Service. If the status of a trace changes from open to in progress or completed, this is registered by the Trace Update component. The trace subsystem then uses the Trace Update Service to provide the new status to the trace subsystem in the CheckedInn Web Service Provider. The corresponding status update is then processed by the Trace Execution component.

5.3 Booking Manager (SL)

This section deals with the role of the Booking Manager[2] in the current system. As the hospitality industry evolves, the need for a robust and efficient channel manager becomes increasingly essential. This tool is critical for managing various booking channels, which may include online travel agencies, direct bookings through the hotel's website, and other third-party services. The Channel Manager serves not only as a bridge between the hotel and these platforms but also as a strategic tool that maximizes room occupancy and optimizes revenue. Figure 5.8 shows the channel manager scenario, all actors and channel manager related use cases visualized as an UML use case diagram.

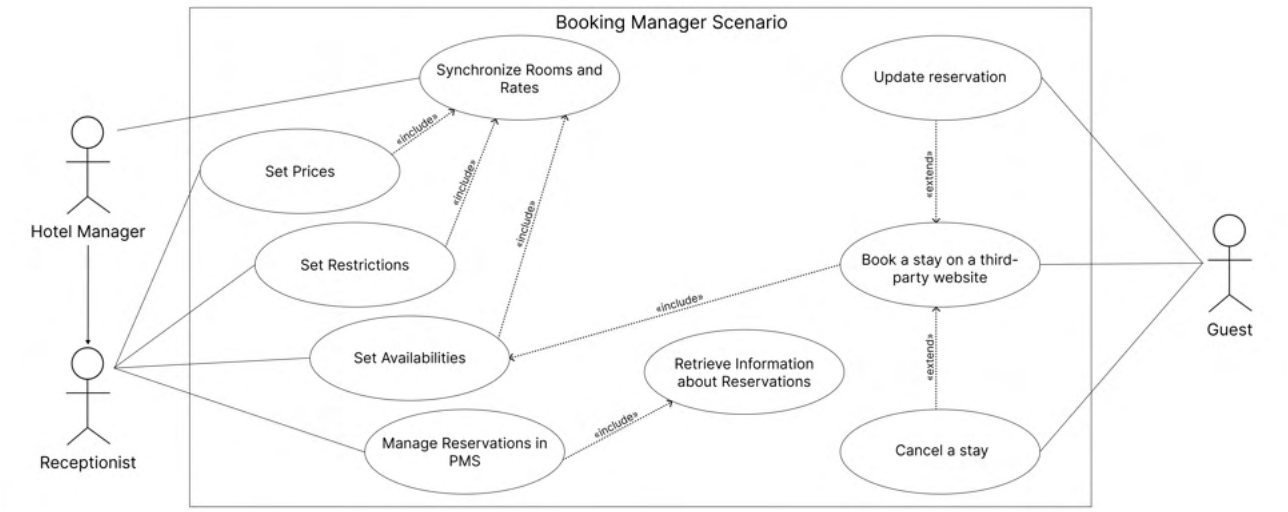


Figure 5.8: Booking Manager scenario for CheckedInn 1.0 (UML Use Case Diagram)

1. `Hotel Manager`
   The Hotel Manager is central to the operational and strategic management of the hotel's accommodations and pricing strategies. This actor interacts with the system to set prices, restrictions, and availabilities. However, these permissions also extend to his staff. It is the hotel manager, specifically, who is solely responsible for the synchronization of rooms and rates with the channel manager.

2. `Receptionist`
   The receptionist has more limited capabilities compared to the hotel manager. Their permissions and daily use cases particularly include managing prices, restrictions, and

[2] The industry-standard term for a Booking Manager is *Channel Manager*. In the following, these terms are used synonymously.

availabilities.

3. `Guest`
   The guest interacts with the system indirectly, typically through third-party websites. Guests have the ability to book a stay, update their reservation, or cancel a stay through these external platforms. Each interaction by the guest triggers updates within the Channel Manager, which then synchronizes this information across all relevant systems to ensure up-to-date management of reservations and availability.

Figure 5.9 displays the objects that participate in the full object model of the booking manager scenario. The actors from the use case diagram match the inheritances of the user class, and each participating object is explained in detail within the context of their respective use cases.

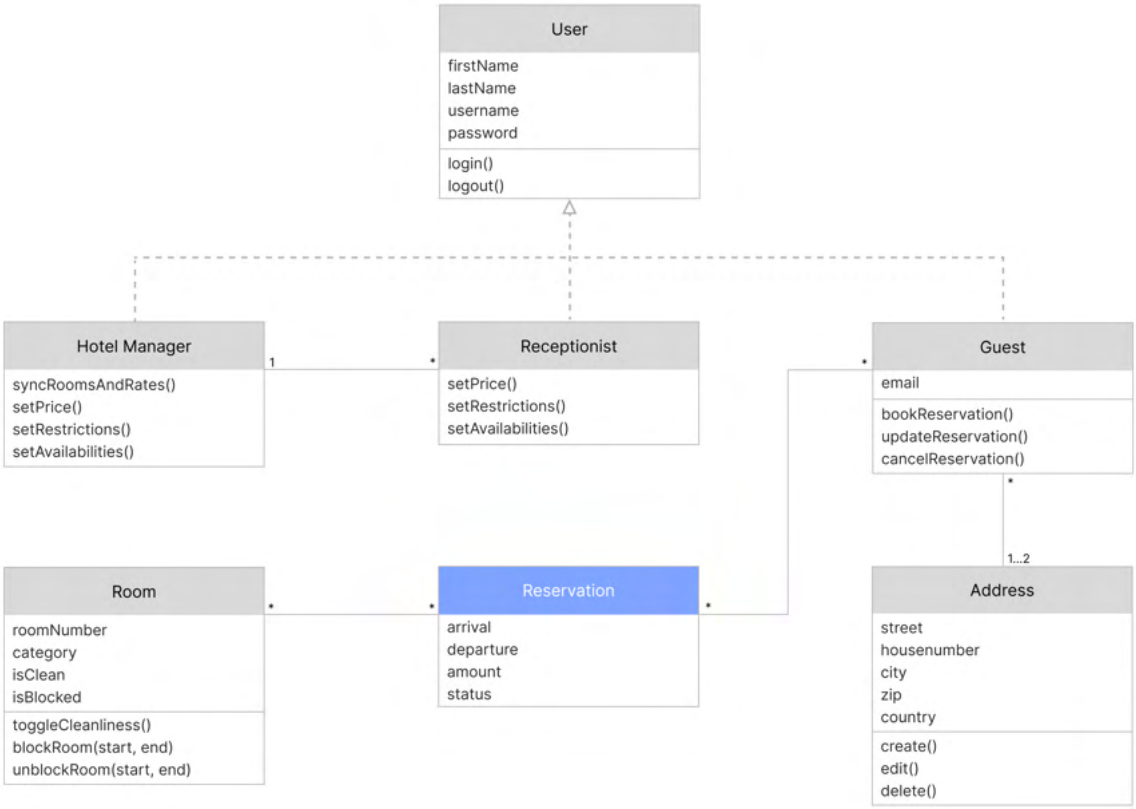


Figure 5.9: Booking Manager Scenario excerpt of the CheckedInn 1.0 Object Model (UML Class Diagram)

The `synchronization of rooms and rates` with third-party booking platforms forms a backbone of the Channel Manager's operations. This synchronization ensures uniformity in the information presented across all marketing channels, thus eliminating discrepancies

and maintaining the hotel's brand integrity. It is through this process that the channel manager updates all connected systems with the latest room statuses and pricing, ensuring that every platform reflects consistent and current data. This use case can only be handled by the hotel manager. This use case, but also the other use cases in this scenario, references especially FR 12, as well as NFR 7.

At the core of its functionalities, the channel manager enables the hotel manager as well as the receptionist to dynamically `set prices` based on a multitude of factors including seasonal demand and special offers, which is fundamental for attracting guests and maximizing revenue. Additionally, the both can impose specific booking restrictions such as minimum or maximum stay requirements and booking windows. These parameters are critical for managing guest flow and expectations effectively. It is represented by the `set restrictions` use case.

Another essential feature provided by the channel manager is the management of room availabilities, described as the `set availability` use case. This function ensures that the number of rooms available for booking is always accurate, thereby preventing overbooking and guaranteeing that potential guests receive real-time availability data. This accuracy is pivotal for maintaining operational efficiency and guest satisfaction.

Mostly the receptionist also interacts frequently with the current system to update or `manage reservations` as guests book, change, or cancel their stays. This interaction is crucial as it aids in providing personalized guest services and enhances operational efficiency. For integrating reservations from third-party websites into the PMS, employees and managers alike also benefit from the capability to `retrieve information about reservations`, which lays the foundation for guest arrivals, organizing housekeeping schedules, and managing other logistical requirements.

Guest interactions with the system typically occur through third-party websites where they can `book stays`, `update reservations`, or even `cancel` them. Each guest action - whether booking a new stay, modifying an existing reservation, or canceling it - triggers an update in the Channel Manager, which then seamlessly reflects these changes across all systems to ensure data consistency and accuracy. Next to FR 12 and NFR 7, the guest use case additionally also address NFR 8.

Figure 5.10 shows the subsystems of CheckedInn 1.0 relevant for the channel manager scenario. The channel manager currently utilized within CheckedInn 1.0 is Dirs21. `Dirs21` provides a total of four services. Initially, they offer the `Sync Inventory Service`, which enables the hotel manager to synchronize the inventory, specifically rooms and rates, with the channel manager. This service is consumed by the `Inventory Management` subsystem within the `Web Service Provider`, which, in turn, offers the `Trigger Synchronization`

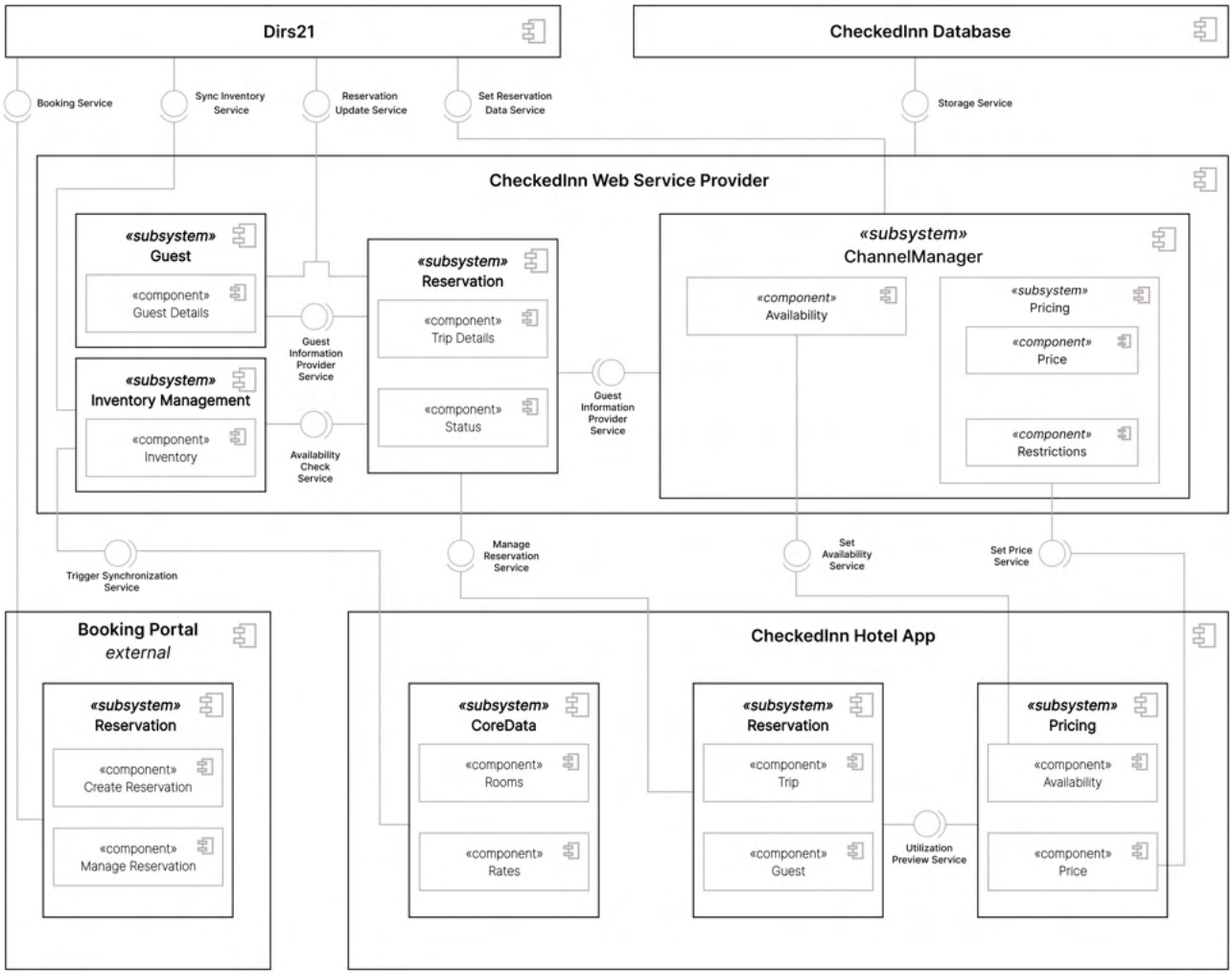


Figure 5.10: Booking Manager scenario excerpt of the CheckedInn 1.0 subsystems (UML Composition Diagram)

`Service` to the `Hotel App`. From there, the hotel manager can alter these details at any time. In this way, Dirs21 is kept informed about the number of rooms a hotel has in each category and the rates typically offered.

Furthermore, the `Booking Service` represents the interface to various booking portals, such as booking.com. While these portals are not directly included in the current system, they are essential to the process. Consequently, they have been abstracted and modeled with a `Reservation` subsystem, which includes components for creating and managing reservations. Through this subsystem, guests can now create and manage their reservations. This includes modifying details such as the number of people or the duration of stay, or even canceling the reservation entirely.

To fully enable guests to make reservations via external portals, it is essential to first establish day-specific and room category-specific availabilities and pricing. Typically, availabilities are synchronized automatically. When a reservation is made for a specific

room and date combination, the availability in Dirs21 decreases automatically. However, availabilities can also be manually adjusted within the Hotel App. For instance, availability can be set to zero to prevent further bookings on a specific day due to staff shortages. These actions are then synchronized with the Channel Manager subsystem in the Web Service Provider via the Set Availability Service.

Pricing can also be variably designed. A useful determinant for this is the utilization rate. If a particular day already has high pre-existing utilization, this indicates high demand, which often leads to an increase in prices. A Utilization Preview Service, offered by the Reservation subsystem, provides forecasts of room usage and is consumed by the Pricing subsystem. Within the Pricing component, prices are then established and synchronized with the Web Service Provider using the Set Price Services. Once this synchronization is complete, the data can then be transmitted via the Set Reservation Data Service provided by Dirs21.

Every five minutes, the Web Service Provider queries Dirs21 via the Reservation Update Service for new reservations. The response received includes all changes to reservations or new reservations that have occurred within the last five minutes. If a reservation is altered, it can be adjusted within the Reservation subsystem. If a new reservation is created, it is first checked locally through the Availability Check Service, provided by the Inventory Management subsystem, to ensure that the reservation aligns with the hotel's available capacities. Subsequently, a new guest is registered within the Guest subsystem, complete with all their personal information, which is stored in the Guest Details component. Following this, a new reservation is created in the Reservation subsystem and linked to this guest, facilitated by the Guest Information Provider Service.

If a hotel employee wishes to manage the reservation, this can be done within the Reservation subsystem of the Hotel App. Through the Manage Reservation Service, offered by the Reservation subsystem in the Web Service Provider, data is transmitted to the Hotel App. From here, all data concerning the guests and their stay can be accessed and modified.

Again, all data processed in the Web Service Provider is stored persistently in the CheckedInn Database via the Storage Service.

5.4 Financial Manager (KN)

The following section elaborates on the financial manager scenario, detailing the respective use cases and relevant subsystems in CheckedInn 1.0. The financial manager can be grouped into the following sections which all mark fundamental components of the

essential operations that receptionists perform daily: registering payments, bill creation and management, cash point handover and closing and management of outstanding payments (also referred to as debitor payments). All these processes are crucial for a hotel, as it creates transparency about the financial situation of a hotel, particularly generated revenue, facilitates the tracking of outstanding payments, and ensures compliance with legal regulations. In particular the billing system is equally significant for guests since it provides clear documentation of their stays and expenses, offers information about any outstanding payments and their settlement, and supplies necessary receipts for business trips. These aspects are essential for maintaining transparency and trust between a hotel and its guests. Therefore, an efficient and accurate financial system is indispensable for both operational efficiency and customer satisfaction for CheckedInn 1.0. In addition, the financial manager scenario is the one with the highest legal requirements. The KassenSichV from 2017 is a German regulation that specifies that every cash transaction must be digitally registered and signed in a verified digital technical security system (TSS)[3]. In addition, the DSFinV-K prescribes that all cash point closing statements must meet certain requirements[4]. These can be summarized to the fact that all business transactions that have led to the current financial statement must be presented transparently and that all digitally signed transactions must also be provided in order to make the cash statement traceable. Therefore, the finance manager not only ensures guest satisfaction and can increase the success of a hotel, but is even crucial for the survival of a hotel, as it cannot operate legally if the legal requirements are not met.

Figure 5.11 portrays the financial managers scenario with all its actors and use cases, visualized as an UML use case diagram. Similar to the previous section, where the reservation and the channel manager scenarios were described, first an explanation of the actors and their role in the financial manager scenario is given. While actors can be similar to previous sections, their roles can still differ in this scenario. Afterwards, the use cases are explained in detail.

1. `Hotel Manager`

   Similar to the channel manager scenario, the Hotel Manager is central to the hotel's compliance with legal requirements regarding the financial management. This actor interacts with the system primarily to check the compliance by managing the TSS transactions and thus ensure adherence to NFR 2 and NFR 7. The hotel manager is also able to perform all other financial operations, but these permissions also extend to the hotel employee.

[3] For a detailed description of the KassenSichV see here

[4] For a detailed description of the DSFinV-K see here

2. `Receptionist`

   The receptionist is performing all operational tasks associated with the financial manager scenario. In particular, this means he records payments, creates bills and performs cash point closings or handovers. This actor is also responsible for the ad-hoc guest correspondence. The regulatory compliance is a black-box for the hotel employee, i.e. the actor cannot retrieve digitally signed transactions.

3. `Guest`

   The guest interacts with the system indirectly, typically by accessing an bill for the reservation that has been sent to them by email. They can then use the corresponding document to view information about their stay and, in the case of outstanding payments, settle these directly using the information in the bill document.

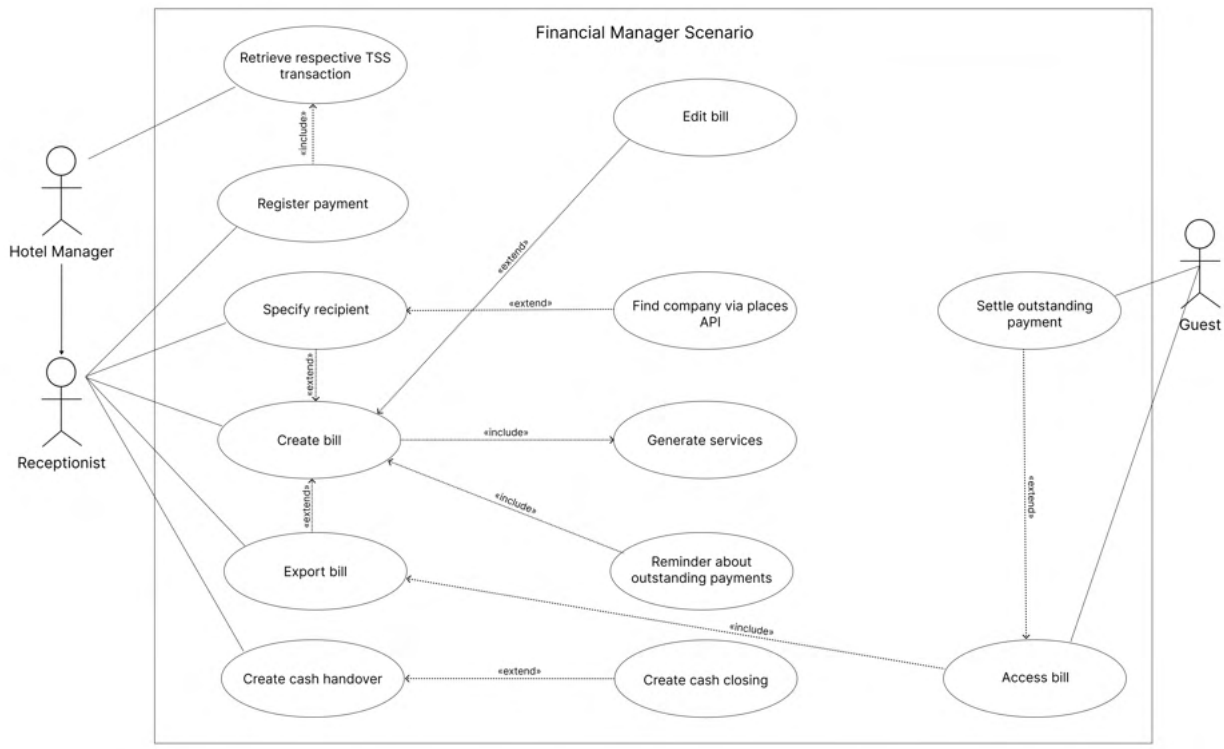


Figure 5.11: Financial Manager scenario for CheckedInn 1.0 (UML Use Case Diagram)

Figure 5.12 shows the participating objects of the complete analysis object model of the current system. The actors of the use case diagram correspond to the inheritances of the user class and all relevant objects are explained subsequently in connection with the respective use cases.

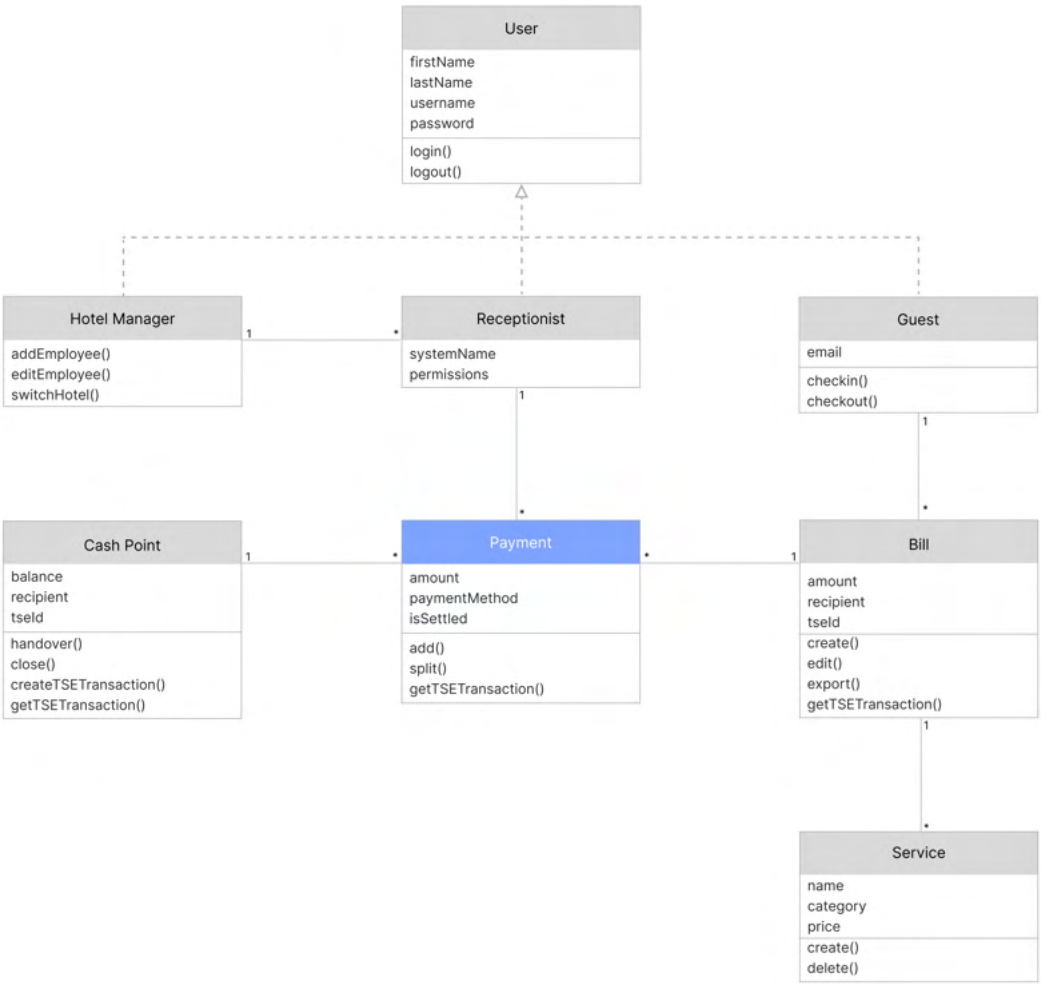


Figure 5.12: Financial Manager scenario excerpt of the CheckedInn 1.0 Object Model (UML Class Diagram)

Essential for all financial operations is the creation of services to be billed. This is usually done by the receptionist (but can also be done by the hotel manager) by creating a reservation, specifying a rate and adding additional services if necessary (see section 5.1). This is explicitly summarized here in the use case `generate services` in order to bundle the dynamic process.

Another fundamental building block is payments made by the guest. These are described by the use case `register payment`. The receptionist enters the payment method, the amount and optionally an internal note and the payment is stored in the system. It is then available for all processes with which it is associated. Once a payment has been registered, the hotel manager can view the digitally signed transaction and its metadata for the payment from a verified TSS (fiskaly[5] for CheckedInn 1.0). This is represented by the use case `retrieve respective TSS transaction`, whereas the include relation to

[5] https://www.fiskaly.com

`register payment` indicates that a payment must first have been recorded. These use cases represent FR 6 and NFR 2.

Once the services have been generated and payments registered, the receptionist can create a bill, which is described by the use case `create bill`. The fact that `generate services` forms the basis for billing is emphasized by the *include* association to this use case. The basic procedure for creating a bill is to match the services consumed by the guest with the payments made. In order to complete a bill and store it in the system, the balance of services and payments must be zero. In addition, a recipient must be specified as the billing address. The creation of a bill is designed to be as dynamic as possible in order to be able to accommodate individual guest requests. It is therefore possible to bill all services of a reservation on different receipts with potentially different billing addresses. This is particularly useful for the frequent case of business guests who consume additional private services. When billing, they often request the accommodation on a business bill and the additional services on a private bill. This use case references FR 3 and NFR 2.

Closely related to the `create bill` use case is the use case `export bill`, which is made clear by the extend association. This includes the export of the digital invoice in the system to a PDF, as well as sending the invoice as an email. When exporting as a PDF, all relevant data is transferred to the customizable template of the respective hotel. In particular, this includes the billing address, as well as all services and payments of the respective invoice. In addition, there is a QR code on the invoice, which can be used to check compliance with the legal requirements for the digital signing of the current invoice. In the event that all or some of the services consumed are paid for using the "on credit" payment method (i.e. there are outstanding payments), there is also a QR code on the bill which can be used to settle the outstanding payments directly. When exported as a PDF, the respective PDF is also automatically stored in the system and linked to the respective reservation. The bill is automatically sent to the guest by email after the guest has checked out. If requested, it can also be sent manually by the receptionist (or hotel manager) to the guest. During this use case compliance with NFR 1, 2, 5, 8 and 10 is crucial. Regarding functional requirements the use cases represents in particular FR 3, and 10 (by creating the debitor payments).

A prerequisite for being able to export an invoice is that the recipient has been specified. This in turn requires that an invoice has been created and is represented with the use case `specify recipient` and its extend relation to `create bill`. To simplify the frequent process of entering a company address, the use case `find company via places api` was realized. Instead of having to google a company's address or have the guest recite it in detail, the system allows the receptionist to search for the company in question using a query. In most cases, the name of the company and perhaps the city in which the address is

located are sufficient to find the correct result. With a simple click, the address of the company is then automatically adopted as the billing address for the invoice. As this process affects both the hotel employee and the guest, saving time not only improves the efficiency of the hotel's internal processes, but also the guest's satisfaction. This use case references FRs 3, 14 and NFRs 5, 7 and 9.

The previous use cases regarding billing all related primarily to the period during a guest's stay. The following use cases are mostly relevant when a guest has already checked out. In the event that a guest has outstanding payments because they have not settled the open amounts before their checkout, a debitor payment is created internally in the system. As already mentioned in connection with the use case `export bill`, the guest receives all relevant information for settling their outstanding payments on the bill. To do this, they must of course have access to the invoice. This is represented by the use case `access bill` and, as the bill must first be sent to the guest, this use case has an include association with `export bill`. The use case `settle outstanding payment` is therefore optional and extends `access bill`. In the event that a guest does not settle their outstanding payments on time, the use case `reminder about outstanding payments` exists, which in turn is executed by the receptionist. As part of a reminder system, the receptionist sends payment reminders to the guest at set intervals as long as their outstanding payments have not yet been settled. This process can be carried out automatically by the system or manually by an employee. These use cases in particular regard FRs 4 and 10 and NFRs 2, 5 and 8.

In rare, but no less relevant cases, an invoice may need to be edited. This is represented by the use case edit bill. As this use case is optional and is based on an existing invoice, it has an extend relation to create bill. There are usually two possible reasons for editing an invoice. Either the billing address was entered incorrectly or it is to be changed retrospectively. In this case, the original invoice can simply be adapted. The other common reason is that the content of the bill needs to be adjusted because, for example, services should not be included on the company bill for guests traveling on business and therefore need to be moved to a separate bill. In this case, the applicable regulations do not allow the content of the original invoice to be simply adjusted. Therefore, a counter-bill (cancellation bill) must be created for the original bill, which neutralizes it and the desired changes can be made in new bills. The use case edit bill therefore potentially also includes the use case create bill and export bill. These use cases reference NFRs 2, 5 and 9.

In addition to recording payments and billing, another fundamental component of the finance manager scenario is cash point management. A cash point describes a technical component in which all on-site payments are recorded (also known as point of sale transactions). Due to legal regulations, all cash point statements must be synchronized with

the DSFinV-K government api. This means that a digitally signed version, which breaks down the financial statements using the underlying business cases, is created for each cash statement. For these, persistence, transparency and traceability play a major role. As a result, Fr 11 and NFRs 1, 2, 5, 7, 10 and 11 are addressed in particular in the use cases create cash handover and create cash closing. The use case create cash handover describes the process of creating a balance sheet of the current cash balance with the respective transactions. This usually takes place between shift changes. The system breaks down all transactions since the last handover, including information on the amount, the payment method and the associated bill. This forms the inventory recorded in the system. The receptionist now has the task of determining the actual cash balance. This includes the cash on hand as well as the balances of the EC terminals. The system offers a simple interface in which the respective stocks are recorded. The system then indicates whether there is a difference between the balance recorded in the system and the balance recorded by the receptionist, thus contributing to the consistency of the hotel's financial data. In order to be able to carry out the cash handover, the receptionist must ensure that all the information on the balance sheet is correct. This means that a potential discrepancy in stock levels can be justified by special circumstances such as a cash withdrawal. The employee then selects another receptionist to 'take over' the cash point. Before starting their shift, this receptionist must check the details of the transferring receptionist and confirm that they are correct. The system thus ensures that the hotel's financial data is consistent, persistent and transparent. This means that NFRs 2, 5, 9, 10 and 11 in particular are referenced. In the system, the data for a cash point handover is saved digitally and also exported and saved as a PDF. The use case `create cash closing` forms a subset of `create cash handover`. This means that all steps of a cash handover are also carried out. This is illustrated by the extend relation to `create cash handover`. In addition, however, the cash balance is set to zero after a cash closing. For this reason, this is often carried out daily in order to empty the cash point as part of the daily closing and to take stock of the income. In addition to the comparison between the cash balance recorded by the system and the actual cash balance, the receptionist is also informed here of how much cash they need to remove from the cash register in order to restore the initial balance (for change). When closing the cash point, a digitally signed version with a breakdown of business cases must now be transmitted to the DSFinV-K government api. This use case therefore additionally addresses NFR 2.

As the use cases of the financial manager scenario have now been described in detail, the participating subsystems of the current system are explained in the following. Figure 5.13

shows the participating subsystems of the financial manager scenario as a UML

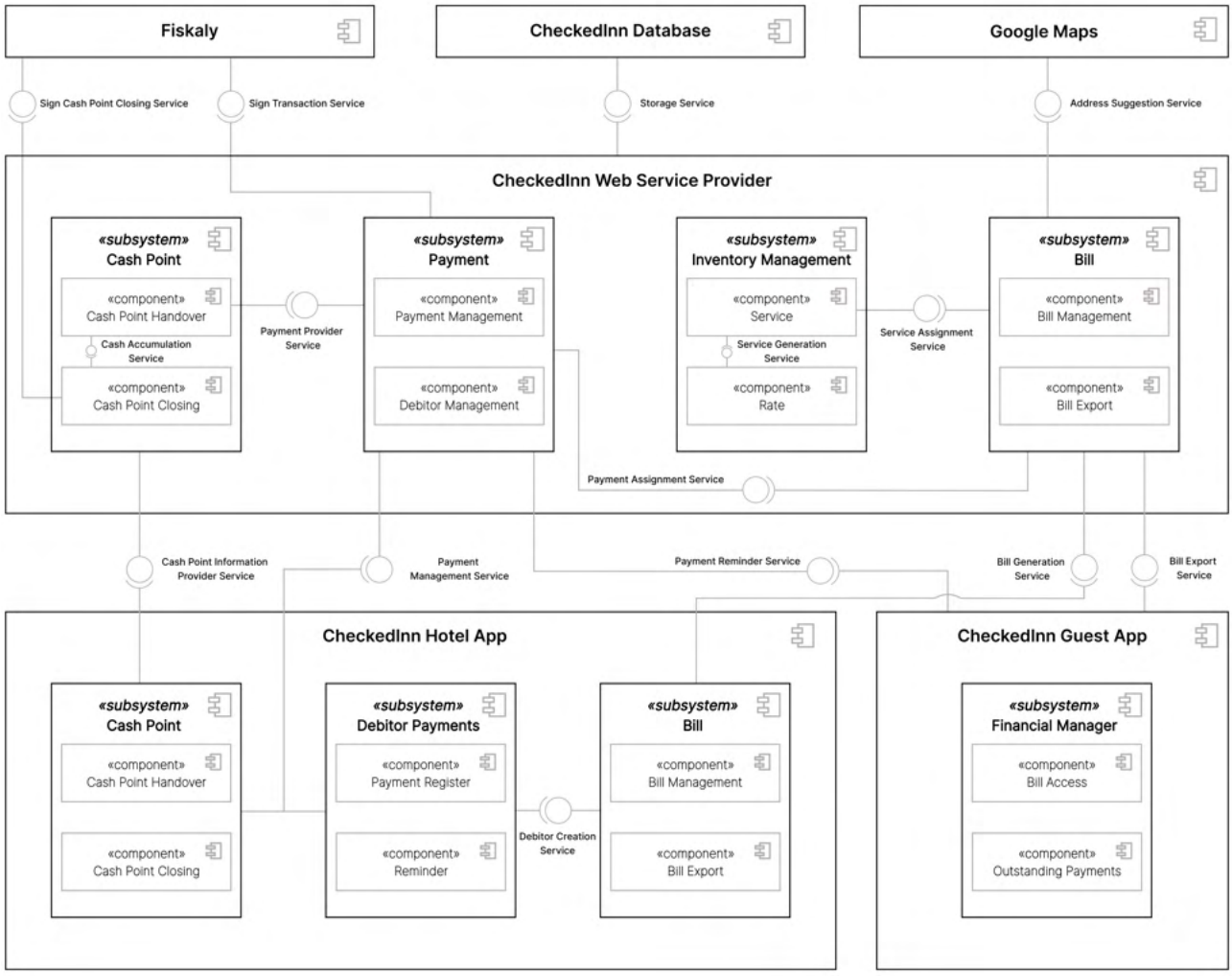


Figure 5.13: Booking Manager scenario excerpt of the CheckedInn 1.0 subsystems (UML Composition Diagram)

composition diagram.

As for previous scenarios, all data processed in the `Web Service Provider` is stored persistently in the `CheckedInn Database` via the `Storage Service`. The technical security system used for the current system is the `Fiskaly`[6] api. This external system provides services such as `Sign Cash Point Closing` to the `Cash Point Closing` component in the `Cash Point` subsystem of the `CheckedInn Web Service Provider`. As pointed out earlier in this section, a cash point closing is a subset of a cash point handover. Technically this is realized by the `Cash Accumulation Service`, which is provided from the `Cash Point Handover` component to the `Cash Point Closing` component. To create a cash point handover the `Payment Provider Service` is provided

[6] Fiskaly is one of the few certified vendors for fiscalisation software in Germany. See here.

by the Payment subsystem. This service delivers all relevant information regarding all registered payments in a given time period. To effectively perform Payment Management, the Payment subsystem utilizes the Sign Transaction Service, which is the second service provided by the external Fiskaly system. Next to payment management, the Payment subsystem is also responsible for Debitor Management (outstanding payments) and for assigning payments to the respective bill. Therefore, the Payment Management Service is provided to both, the Cash Point and Debitor Payments subsystems inside the CheckedInn Hotel App. Together with the Cash Point Information Provider Service, which is provided by the Cash Point subsystem inside the CheckedInn Web Service Provider, and the Debitor Creation Service, which is provided by the Bill subsystem inside the CheckedInn Hotel App, this allows the receptionist to conveniently create cash point handovers and closings, as well as to manage outstanding payments. While the Cash Point Information Provider Service mainly delivers persistently determined data to create cash point handovers and closings, the Debitor Creation Service is responsible to automatically track outstanding payments in the case that a guest's bill was issued with the payment method „on credit". In order to carry out the assignment of payments and services described in connection with the use case diagram in this section, further services are required. On the one hand the Payment subsystem provides the Payment Assignment Service, and on the other hand the Inventory Management subsystem provides the Service Assignment Service to the Bill subsystem in the CheckedInn Web Service Provider. This allows services consumed to be settled with payments made, thus bringing the balance of a bill to zero. The Rate component within the Inventory Management subsystem provides the Service Generation Service to the Service component. This service creates all services that belong to the respective rate and creates them for billing. To successfully create a bill, the Bill component uses the Address Suggestion Service of the external Google Maps system. This provides suggestions for billing addresses based on queries entered in order to facilitate the billing of corporate customers. Furthermore, the Bill subsystem provides the Bill Export Service to the CheckedInn Guest App. This enables the automatic export of bills directly to the guest. To additionally enable the manual export of bills by the receptionist, the Bill subsystem also provides the Bill Generation Service to the Bill subsystem in the CheckedInn Hotel App. The aspect of managing outstanding payments is completed by the Financial Manager subsystem in the CheckedInn Guest App. While the Bill Access component allows the guest to retrieve their exported bill directly, the Outstanding Payments component enables the payment of outstanding invoices. The information contained in the invoice is used for this purpose. In the event that a guest does not settle their outstanding payment within the specified period, a multi-stage

reminder system is used. For this purpose, the `Debitor Management` component provides the `Payment Reminder Service` to the `CheckedInn Guest App`. In addition, the receptionist can manually view the reminder level via the `Reminder` component of the `Debitor Payments` subsystem.

5.5 Report Scenario

After the previous sections have already explained important scenarios for the day-to-day business of the hotel industry, this section explains the last fundamental building block, namely the report scenario, in detail. To do so, the use cases together with the relevant objects of the scenario are explained first and subsequently, the relevant subsystems are elaborated on.

While the previous scenarios primarily involved operational activities, the report scenario is about administrative and strategic tasks. This is also evident from the fact that the previous scenarios involved several actors, such as the hotel employee and the guest, whereas the hotel manager is the only actor in this scenario. In specific terms, the report scenario is about creating reports that provide insights into the situation of a hotel and create added value. The revenue report is of particular importance here. However, the scenario also includes other reports, such as the general manager report, which provides an insight into all relevant key figures of the hotel, or analyses of guest characteristics and behavior, as well as relevant reports for regulatory purposes, such as the DSFinV-K or GOBD export[7].

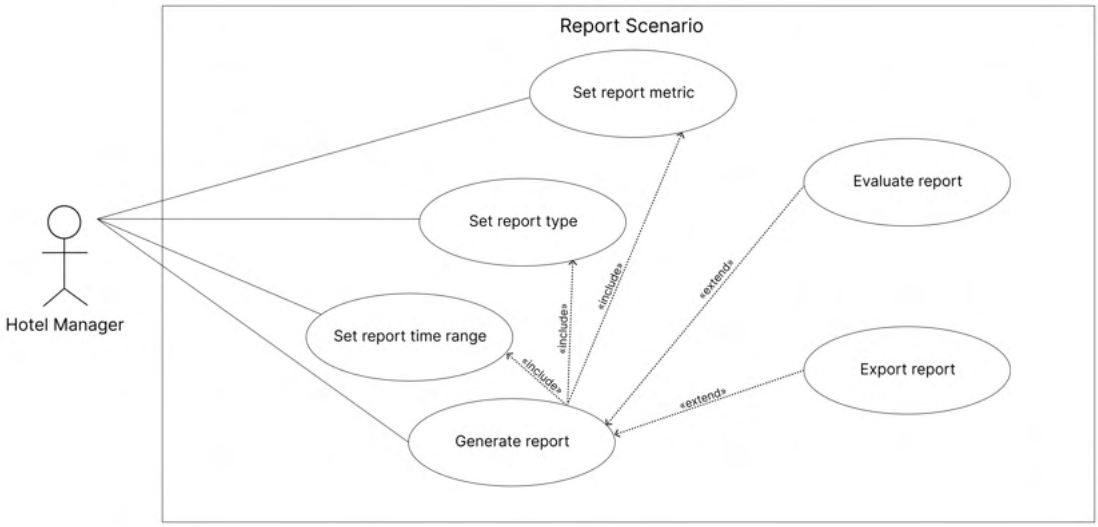


Figure 5.14: Report scenario (UML Use Case Diagram) for CheckedInn 1.0

[7] These two reports are explained later in the section.

Figure 5.14 shows the report scenario with its actor, the hotel manager, and the use cases as a UML use case diagram. Again, a brief description of the role of the hotel manager in the context of the scenario is also given here. The use cases are then explained in more detail.

1. `Hotel Manager`
   The hotel manager is solely responsible for the hotel's administrative and strategic decisions. It is therefore important to gain insight into the hotel's situation in terms of various metrics. To do this, the hotel manager determines the parameters for the reports to be created in such a way that the reports provide the highest added value. In this scenario, he simply selects the respective parameters for the reports to be generated. The reports are then automatically generated by the system.

As already mentioned, the active task of the hotel manager is primarily to select the right parameters so that the generated reports provide the highest added value. The reports are then created automatically by the system based on the set parameters. This does not mean that the creation of reports is trivial. Quite the opposite is true: the internal complexity and intricacy is merely hidden from the user in order to make the system easily accessible and to manage the complicated processes internally.
Setting the parameters is described by the three use cases `Set report type`, `Set report metric` and `Set report time range`. The fact that these form the foundation for creating a report is made clear by the include relations to the use case `Generate report`. The ability to tune those parameters addresses NFRs 8, 9 and 10. In the use case `Set report type`, the hotel manager sets the type of report to be generated. There are mainly three specific types in which reports can be categorized, namely `Revenue Reports`, `Guest Analysis` (Reports) and `Compliance Reports`.[8] This can be seen in in Figure 5.15, which portrays the participating objects as a UML class diagram.
The revenue reports evaluate the hotel’s financial situation. Therefore, issued payments and bills are taken into account to generate the respective report. Guest analysis reports analyze specific metrics of the hotel’s guest and compliance reports document the information necessary to comply with legal regulations. Each report type can be further divided into a metric (or sub-type). Choosing the specific metric is described by the use case `Set report metric`. For revenue reports, these metrics are, for example, whether sales are calculated based on the total of services billed or the total of payments received. Both can indicate a different financial situation, as invoiced services already include

[8] The aforementioned general manager report is a combination of all three, providing a comprehensive overview of the hotel’s situation. This is particularly important if a hotel is to be sold.

Figure 5.15: Report scenario excerpt of the CheckedInn 1.0 Object Model (UML Class Diagram)

outstanding payments as revenue, while incoming payments do not. With guest analysis reports, the number of metrics is significantly larger, as many key figures can be analyzed. The most important ones include the length of stay of guests, whether they are traveling for business or private reasons and their purchasing behavior. The report metrics for the compliance reports are based on the applicable legal requirements and can vary depending on the location of the hotel. The regulations that apply nationwide are the DSFinV-K export already explained in section 5.3, which specifies a standardized format for the cash point export, and the GOBD export, which is the legally compliant export of tax-relevant data in accordance with the principles for the proper keeping and storage of books, records and documents in electronic form and for data access (GOBD) for submission during tax audits.

Once these parameters have been set, the hotel manager has to select the time period for which the respective report is to be created. This is described by the use case `Set report time range`. Subsequently, the `Generate report` use case can be initiated, in which the respective report is created by the system based on the parameters set. By this FR 13 is addressed. While compliance with NFR 5 is of great importance for all reports, compliance with NFR 1 is particularly important for guest analysis reports. `Generate report` is optionally extended by the two use cases `Export report` and `Evaluate report`. `Export report` describes the export of the report in various formats. For revenue reports and guest analysis, the main export format is as a PDF. For compliance reports, however, the system enables export as a CSV file as well as direct transmission to the respective authority to which the report must be submitted. This in particular addresses NFR 2 and 7. Evaluate report describes the evaluation of reports with the aim of utilizing them to generate added value. This involves manual analysis of the report by the hotel manager, but is also supported by the system through the provision of metrics that break down the content of a report into relevant key figures[9].

Figure 5.16 shows the relevant subsystems for the report scenario visualized as a UML composition diagram. In the following, the subsystems and their respective services will be explained in more detail.

The central subsystem of this scenario is the `Report` subsystem in both the `CheckedInn Web Service Provider` and the `CheckedInn Hotel App`. In the `CheckedInn Web Service Provider`, the report subsystem consists of the `Report Generation` and `Report Export` components. For report generation, the report subsystem consumes services from the `Guest`, `Bill` and `Payment` subsystems in the `CheckedInn Web Service Provider`. While the `Guest` subsystem with the components `Personal Information` and `Purchasing Behavior` provides all relevant information for the guest analysis reports with the `Guest Information Service`, the `Bill` subsystem provides information from created and exported bills. These are particularly useful for revenue reports based on billed services, as this information originates from the invoices. In order to do this, the bill subsystem with its `Bill Management` and `Bill Export` components provides the `Bill Accumulation Service`. The `Payment` subsystem in turn provides all information regarding received and outstanding payments with the `Payment Accumulation Service`. Based on this, revenue reports can be generated on the basis of payments received, as well as reports on outstanding payments. In order to be able to provide this service, the payment subsystem in turn contains the two components `Payment Management` and `Debitor Management`. The data provided by these services is again stored persistently in the `CheckedInn Database` system. To create a report, the `Report Generation`

[9] For the revenue reports, these are, for example, the total revenue or the total outstanding payments.

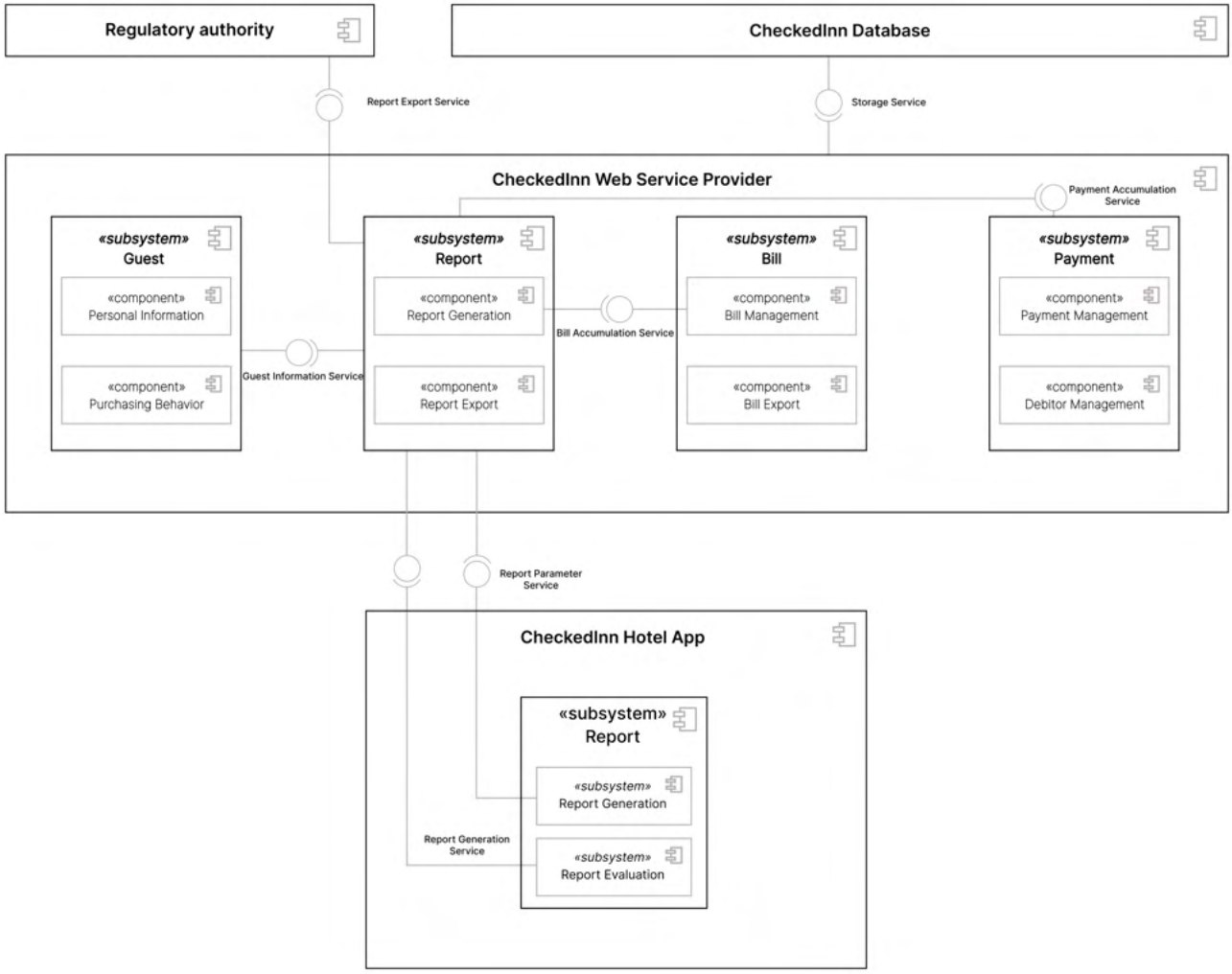


Figure 5.16: Report scenario excerpt of the CheckedInn 1.0 subsystems (UML Composition Diagram)

component additionally requires the parameters specifying which report is to be created. These are set in the report subsystem in the `CheckedInn Hotel App`. To do this, the `Report Generation` component provides the `Report Parameter Service` to the report subsystem in the `CheckedInn Web Service Provider`. This service provides all the necessary information regarding the type, used metrics and the time period of a report to be generated. Once the report subsystem has consumed this service, the report can be created with the `Report Generation` component. The report is then exported by the `Report Export` component. The target systems are once the external systems of the authorities to which the reports must be sent. For this purpose, the report subsystem provides the `Report Export Service` to the external system `Regulatory authority`. Furthermore, the generated report is made available in the `CheckedInn Hotel App`. To do this, the `Report` subsystem in the `CheckedInn WebService Provider` provides the `Report Generation Service` to the `Report Evaluation` component of the report

subsystem in the `CheckedInn Hotel App`. Once the report has been generated using this service, it can then be evaluated by the hotel manager.

5.6 Requirements fulfillment evaluation

Now that all the fundamental scenarios of the current CheckedInn 1.0 system have been elaborated in detail, a mapping between the functional and non-functional requirements from chapter 4 and the use cases of this chapter is carried out. The aim is to clearly show which requirements are all fulfilled and which may require more attention in the course of forward engineering CheckedInn 2.0. In addition, this mapping can be used during forward engineering to ensure that adjustments do not result in requirements that have already been met by the current system no longer being met.

Table 5.1 shows which use cases from the respective scenario fulfill which functional requirements. Similarly, Table 5.2 shows the mapping between non-functional requirements and use cases. For each requirements, at least one representative use case of a scenario is listed, given that there is a use case that fulfills the requirement. A requirement may also be fulfilled by several use cases within a scenario. However, as a detailed explanation of which requirements use cases fulfill has already been provided in the respective section of this chapter, the focus here is on being able to easily visually deduce which requirements are fulfilled by which scenarios.

Requirement	Reservation Scenario	Trace Scenario	Booking Manager	Financial Manager	Report Scenario
FR 1	`Create reservation`				
FR 2	`Check room availability`				
FR 3				`Create bill`	
FR 4				`Export bill`	
FR 5	`Check-In`				
FR 6				`Register payment`	
FR 7		`Create Trace`			
FR 8		`Get compressed information for daily business`			

Requirement	Reservation Scenario	Trace Scenario	Booking Manager	Financial Manager	Report Scenario
FR 9		Generate and export lists for daily needs			
FR 10				Settle outstanding payment	
FR 11				Create cash closing	
FR 12			Book stay on a third-party website		
FR 13					Generate report
FR 14				Find company via places api	
FR 15	Create reservation				

Table 5.1: Evaluation of the functional requirements fulfilled by the scenario use cases

As can be seen from Table 5.1, all functional requirements for the current system are fulfilled by at least one use case from one of the scenarios. It is also worth noting that no functional requirement is fulfilled by use cases from distinct scenarios. This clearly shows that the modular component-oriented system principle from NFR 6 was taken into account and adhered to during the development of CheckedInn 1.0. Consequently, during the development of CheckedInn 2.0 with forward engineering, it must be ensured that the functional requirements continue to be fulfilled by at least one use case. Any changes that result in this no longer being the case must be explicitly justified.

Requirement	Reservation Scenario	Trace Scenario	Booking Manager	Financial Manager	Report Scenario
NFR 1	Create reservation			Export bill	Generate report
NFR 2				Create bill	Export report
NFR 3	Edit reservation				
NFR 4					
NFR 5	Check room availability			Create cash closing	Generate report

Requirement	Reservation Scenario	Trace Scenario	Booking Manager	Financial Manager	Report Scenario
NFR 6					
NFR 7			Synchronize rooms and rates	Find company via places api	
NFR 8	Online Check-Out		Book stay on a third-party website	Access bill	
NFR 9				Edit bill	Set report type
NFR 10				Settle outstanding payment	Evaluate report
NFR 11				Create cash handover	

Table 5.2: Evaluation of the non-functional requirements fulfilled by the scenario use cases

Table 5.2 shows that all non-functional requirements from section 4.2 are also fulfilled by the current system. Unlike the functional requirements, these are fulfilled almost without exception by use cases from multiple scenarios at the same time. This in turn makes it clear that the requirements for quality characteristics of the current system were taken into account when implementing each requirement. It is noticeable that NFR 4 (scalability) and NFR 6 (maintainability) are not fulfilled by any individual use case. This is because they cannot be explicitly fulfilled by a use case, but describe qualities of the system as a whole. The fact that these are fulfilled in the use of the entire system and in the interaction of all components was verified by means of system tests.

6. Forward Engineering (KN)

The following chapter discusses the results of forward engineering in detail. As already described in section 3.1.3, forward engineering consists mainly of the steps of usability testing, the identification of new requirements for CheckedInn 2.0 and the actual development of CheckedInn 2.0. As the focus of this thesis is on feedback-driven usability testing, instead of a detailed discussion of the technical realization of CheckedInn 2.0, this chapter presents a comprehensive experimental evaluation of the results from the usability study of CheckedInn 1.0. To still provide an impression of the practical realization of feedback during the usability tests, exemplary handling of feedback can be found in Appendix C. There, the implementation of feedback is illustrated for each of the five classes.

Over a period of nine weeks and five iterations[10] feedback was systematically collected, analyzed and integrated into the system to ensure continuous improvements and to adapt the system to the actual needs of the users. An online form, accessible at any time, was used to collect feedback. This enabled users to provide continuous feedback. At the same time, structured usability tests were carried out at regular intervals of one to two weeks. These tests were carried out on the initiative of three to five employees from two to three different hotels, who were selected according to the principle of discount usability testing, which is described in detail in section 2.2.2. System changes made, particularly those resulting from specific change requests, were evaluated as part of beta testing. This testing involved people who had originally provided feedback, including future users of the system. These tests served to thoroughly test and validate the changes before their official release. The iterative approach described above allowed feedback from previous iterations to be effectively integrated into subsequent versions of the system.

Figure 6.1 shows the amount of classified feedback collected in each iteration. It can be seen that less feedback was given over the course of the iterations. However, as this is not very informative, a more detailed evaluation of the results is carried out based on two metrics. Section 6.1 provides an in-depth analysis of the test results by clustering the feedback by the five classes and describing and evaluating the quantity of individual feedback within the different testing iterations. This systematic analysis makes it possible to identify recurring problems and user patterns that require targeted adjustments to the design of the system. Section 6.2 is dedicated to the specific evaluation of the questionnaire based on heuristics. It analyzes how the evaluations of the system have developed in the context of the individual usability heuristics over the entire test and

[10] specifically from August to October 2024

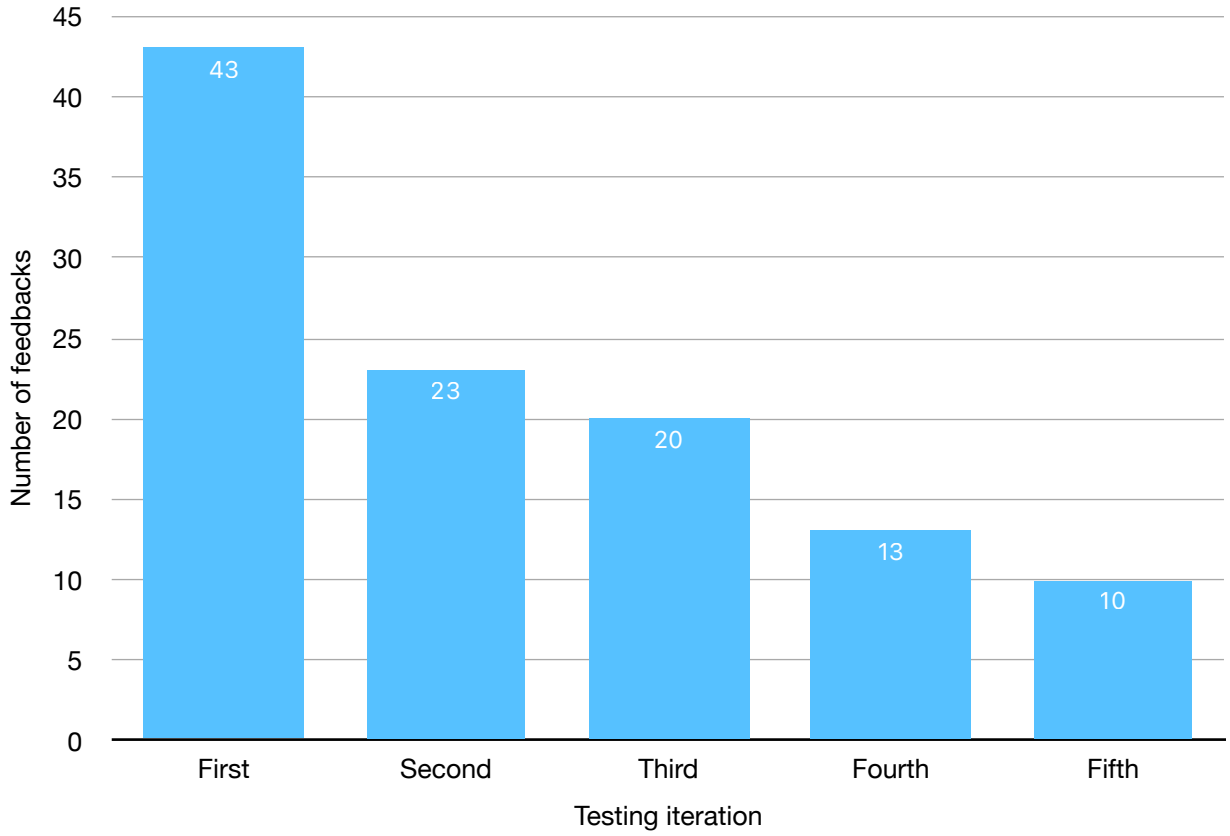


Figure 6.1: Number of feedbacks collected by testing iteration

development period. This analysis is crucial in order to identify the areas in which the system has made progress and in which areas there is still room for improvement.

Finally, the results are summarized and critically examined in an overall assessment in section 6.3 Threats to Validity. While correlations between the implementation of feedback and changes in usability or system performance are shown in Sections 6.1 and 6.2., section 6.3 examines and questions the actual causal relationships and takes a closer look at the change in user behavior of operating software over a longer period of time. In addition, the relevance of these findings for future developments and improvements to the software is emphasized in order to create a basis for further research and implementation approaches.

6.1 Feedback type-based evaluation

As already described in the introduction to this chapter, the evaluation of the experiment is based on two metrics. In this section, the results are evaluated based on the feedback type. In addition to a graphical illustration, the results are also analyzed, their implications for the course of the experiment are explained and recognizable patterns are derived.

6.1.1 Deployment problems

In this section, the deployment problems encountered in the various usability testing iterations are analyzed in detail. Figure 6.2 illustrates the number of deployment issues on the Y-axis, while the respective testing iteration is shown on the X-axis.

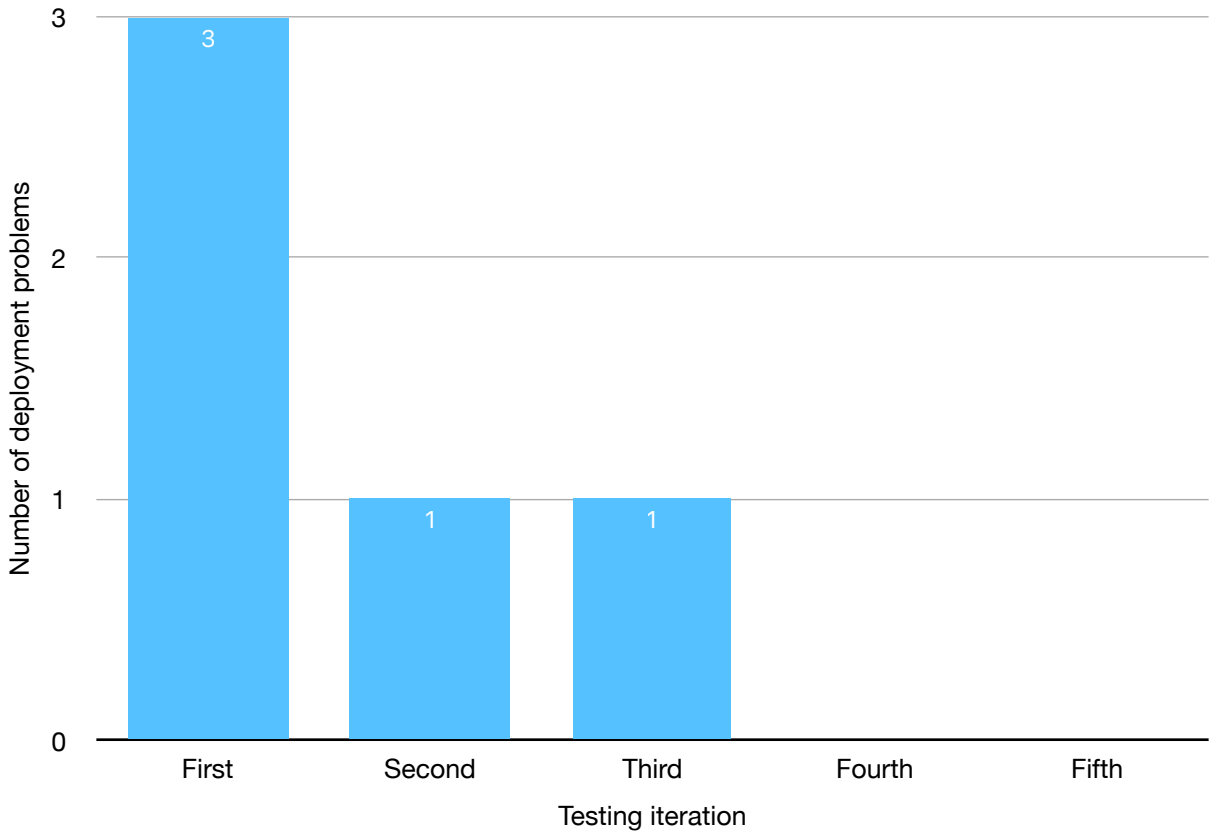


Figure 6.2: Number of deployment problems by testing iteration

Figure 6.2 shows that three deployment problems were reported as feedback in the first usability testing iteration. These then decreased in the second iteration, so that only one deployment problem was reported. In the third usability testing iteration, one deployment problem was also reported, so that these remained constant. In the fourth and fifth usability testing iteration, however, no more deployment problems were reported. Deployment problems are particularly related to the technical infrastructure, including the CI/CD pipeline. It is reasonable to conclude that most deployment issues occurred during the first release, as the system was previously used and tested exclusively in the development environment. The problems reported during the first test iteration had their origin in the configuration of the deployment pipeline and resulted in the system being inoperative. In particular, the problem was due to a faulty setup when transferring the system from the development environment to the deployed version in the user environment. The fact that a deployment problem was again reported as feedback in the second usability test iteration

could be due to the fact there were no fundamental changes to the technical infrastructure, but minor adjustments were made. These led to another deployment problem, which did not prevent the system from being fully usable, but did limit its functionality. In the third usability test iteration, another deployment problem was reported. In contrast to the previous problems it did not concern the availability of functions, but increased latency times. Compared to the previous iterations, these were in some cases significantly higher and considerably affected the usability of the system, even if functionality was not completely restricted. The reason for this lay in the implemented change requests and feature proposals from the first two iterations, which led to a deterioration in system performance in terms of latency. This problem was resolved subsequently in the course of the experiment by, among other things, running asynchronous processes in parallel, which restored performance[11]. During the subsequent adjustments and extensions of the system, there was no longer any increased latency due to extended functionalities. This could be due to the fact that the factors that previously resolved the increased latency were taken into account in the subsequent adjustments. In addition, the technical infrastructure, including the CI/CD pipeline, was not significantly changed in the further course. These two aspects could explain why no further deployment issues were reported during the fourth and fifth usability test iteration.

Overall, it can be concluded from the decreasingly occurring deployment problems that the system stabilized over the course of the experiment and that development and operation became more robust. During the implementation of change requests and the development of feature proposals, it was not certain whether deployment problems would occur again at the beginning of the next usability testing iterations. The fact that no deployment problems occurred after the third release despite the adjustments and enhancements therefore also indicates that the CI/CD pipeline was implemented in a stable manner for adjustments to the system.

Besides the number of deployment problems aggregated by testing iteration, there were also noteworthy findings regarding the times after a new release at which the deployment problems occurred. Figure 6.3 shows the times at which deployment problems were reported as a timeline. The date is shown on the x-axis with a weekly scale. The two-week releases, which always took place at the beginning of a usability testing iteration, are shown as blue dots. The times at which a deployment problem was reported are shown as blue triangles below the respective release. The time after the release is indicated both visually by the distance to the point of the release and textually by the time in hours after which the deployment problem was reported.

[11] See for example here

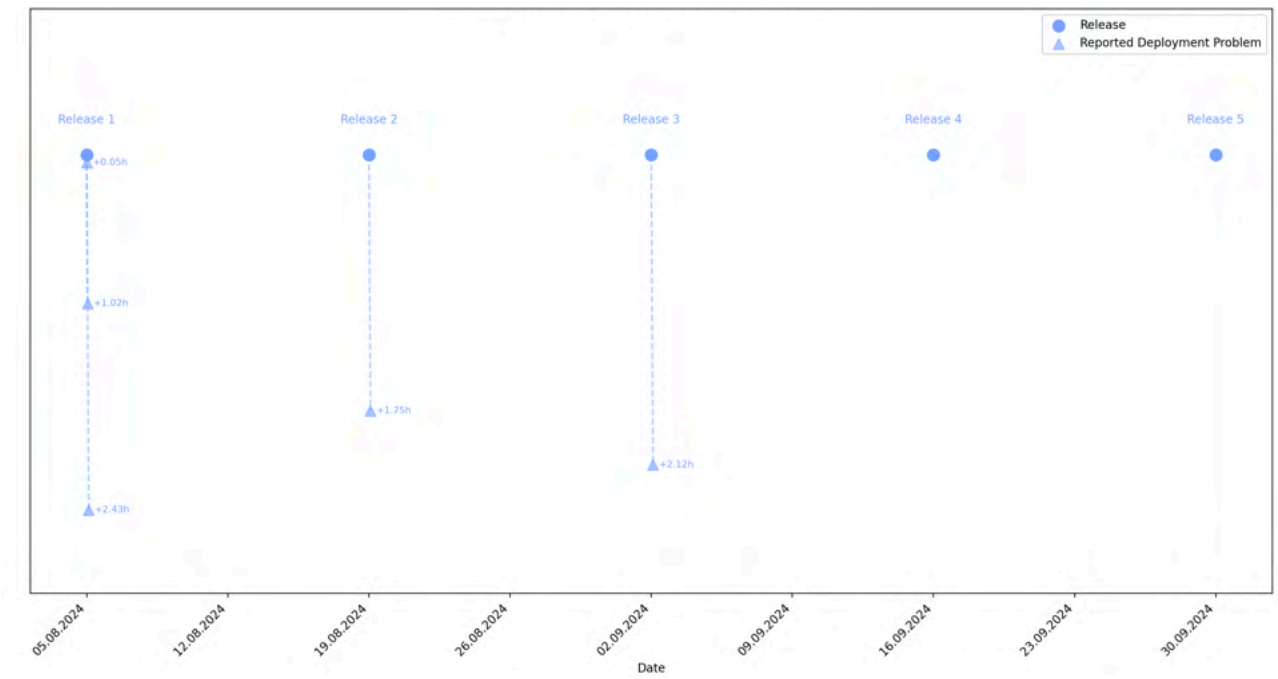


Figure 6.3: Reported Deployment Problems timeline

When analyzing the deployment problems after a release, a number of interesting things were found. In particular, the first release had a problem immediately after deployment that limited the availability of the entire system. Due to this error, users were unable to log into the system, resulting in a complete loss of accessibility. After this initial issue was resolved, further deployment issues occurred at approximately one hour intervals. With subsequent releases all reported issues also occurred within the first few hours of each release.

This could be explained by the fact that deployment problems are often related to errors in the technical infrastructure, such as the CI/CD pipeline or the setup of the production environment. This could also be the reason why they were identified and reported within a short period of time for all releases. It was particularly noticeable with the first release that all problems occurred at short intervals after the previous one and always only after the previous one had been fixed. This highlights the fact that deployment problems can immensely restrict the functionality or even prevent the general availability of the system and should therefore be fixed immediately. The time intervals after a release of 1.75 and 2.12 hours respectively after which deployment problems occurred in the second and third releases indicate that these did not restrict functionality as fundamentally as in the first release. A more stable infrastructure can be inferred from the fact that no deployment problems occurred in the last two releases. Lessons learned from this could be, among other things:

1. **Test the deployment process**

   The technical infrastructure should be intensively tested for the transition from the

development to the production environment.

2. **Supervise new releases**
   New releases should be actively supervised and accompanied, as errors can often occur shortly after the release and potentially massively restrict the further use of the system.

3. **Release when user activity is minimal**
   If possible, new releases should take place at times when the number of users using the system at that time is minimal. In this way, the system can be tested for errors in a risk-minimized manner and without massively affecting users' operational activities. For hotels this, for instance, may mean that new releases do not take place during the morning shift, but rather at night. Although the number of users is not zero here, it is significantly lower than during the morning shift, as this is when, among other things, most check-ins take place.

6.1.2 Bug reports

After the deployment problems have been analyzed, in this sub-section, the bug reports encountered in the various iterations are explained in detail. Figure 6.4 shows the number of bug reports by iteration.
By far the most bug reports were issued during the first iteration. With 22 reported bugs, the number is three times as high as in the following testing iteration and also more than twice as high as in the third iteration, in which the second most bugs were reported. There were hardly any bugs in the last two usability testing iterations. Thus, a significant downward trend can be observed in the number of reported bug reports, with a renewed increase during the third usability testing iteration.
Several conclusions can be drawn from this. Before the first usability testing iteration, the system was only tested by the developers in the development environment. As a result, the processes were not tested in their entirety and by no means all edge cases were carried out. This could explain why, relatively speaking, so many bugs occurred in the first iteration. The high number of bugs in the first iteration was then fixed before the start of the second iteration. As this was very time-consuming, in addition to deployment problems and bug reports, hardly any other feedback could be processed. This meant that change requests, feature proposals and usability issues from the first iteration could only be implemented to a large extent in the second iteration. Consequently, the system at the start of the second iteration was largely the same as the system from the first iteration, but with the bugs fixed.

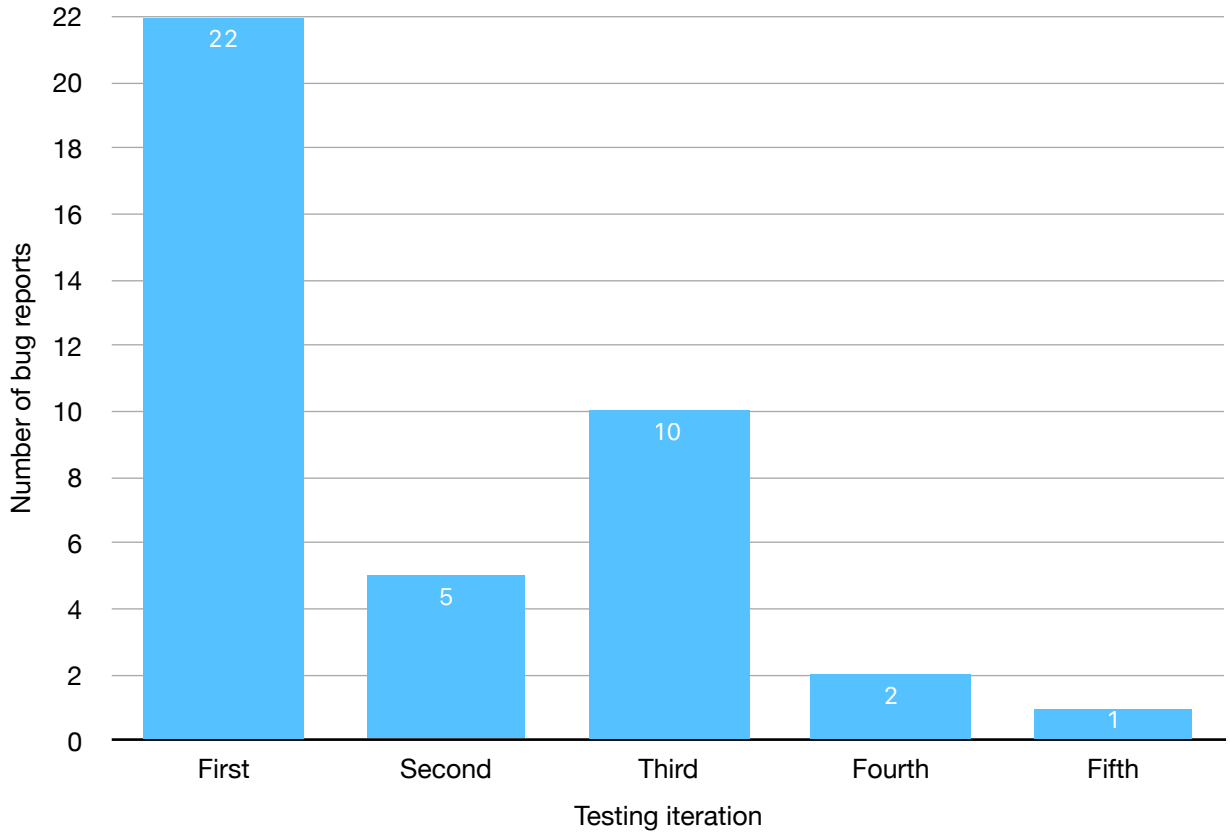
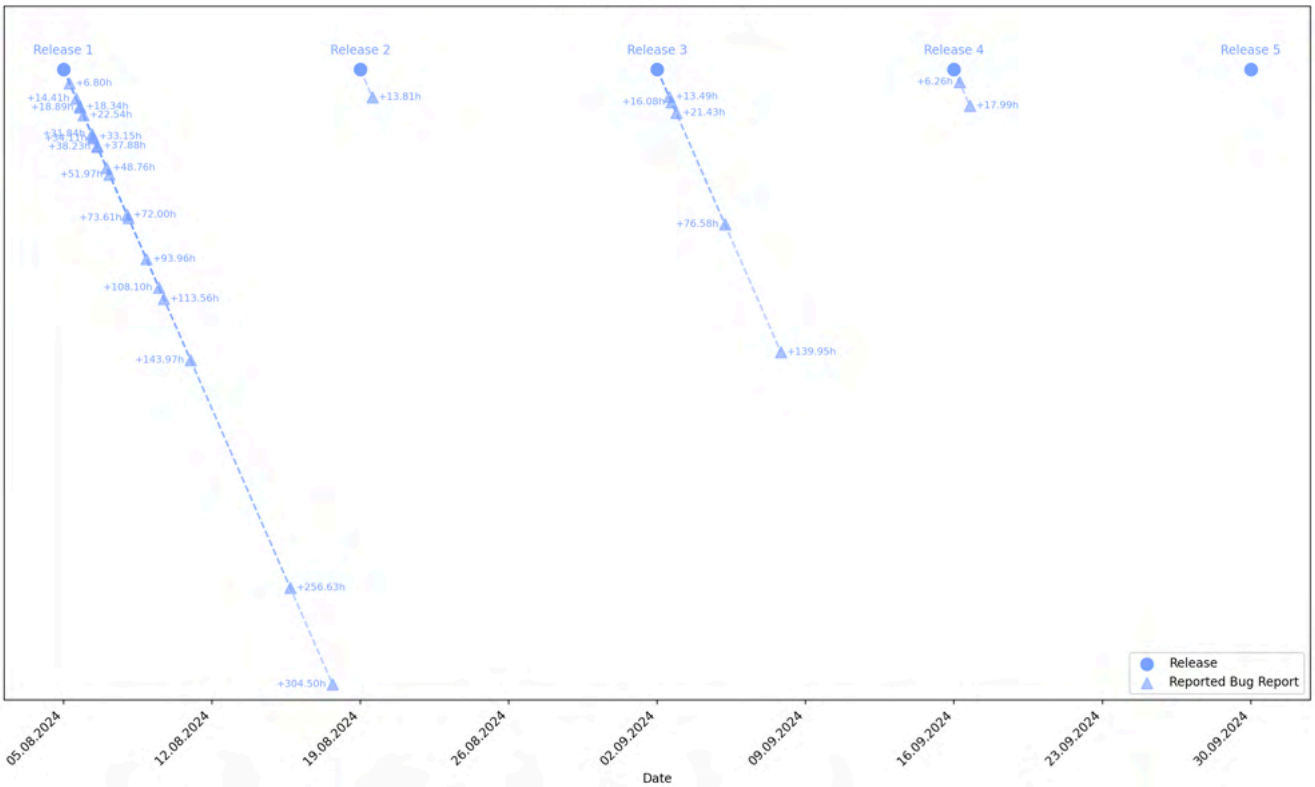


Figure 6.4: Number of bug reports by testing iteration

That could be the reason why significantly fewer bugs were reported in the second iteration. Fixing these bug reports took significantly less time, which meant that a large number of change requests, feature proposals and usability issues could be implemented in the second iteration. In the third iteration, this in turn resulted in the changes and new features causing new bug reports. As the change requests, feature proposals and usability issues from both the first and second iteration were processed during the second iteration, the focus was on a large number of innovative improvements. The increased number of bug reports in the third iteration therefore indicates that this was at the expense of extensive testing of the new release. However, since the bugs were of comparable complexity to those from the first iteration and the number was not as high, fixing them was not as time-consuming as in the first iteration. Due to this fact, there was enough time available during the third iteration to address the reported change requests, feature proposals and usability issues and to test them extensively prior to the next release. As the number of feedbacks to deal with here also only related to those from the third iteration, the changes resulted in fewer bugs for the next release. This is illustrated by the fact that only two bug reports were received during the fourth iteration. This pattern continued, as evidenced by only one bug report in the final usability testing iteration.

Similar to the deployment problems, the times at which the bug reports occurred also provided exciting insights. In contrast to the deployment problems, however, the specific

times at which they occurred are not available for all bug reports. This is due to the fact that some bug reports were also identified via the questionnaires. As these are carried out on site together with the study participants and the feedback collected is later entered into the CheckedInn Feedback App, the process of collecting feedback is asynchronous and it is not possible to derive the specific times at which bugs occurred[12]. Nevertheless, sufficient bugs were reported via the feedback form, which means that the specific times at which they occurred are available for a sufficient number of bug reports, allowing them to be analyzed with relevant insights. Figure 6.5 shows the times at which bugs were reported.



Figur 6.5: Reported Bug Reports timeline

The analysis of the times at which bugs occurred in the software shows interesting patterns that are consistent with the findings from the aggregation by release. Most bugs occurred in the first iteration, hardly any bugs were reported in the second iteration, while in the third iteration the number of bugs increased again slightly before the downward trend continued. It is noteworthy that the first bugs were reported later than the deployment problems. While all deployment problems were identified no later than two and a half hours after the release, the first bug reports were only submitted six hours after the release at the earliest. The majority of bugs were discovered within the first 48 hours after a new release. It is particularly remarkable that in iterations two and four, all bugs were reported within this

[12] As the deployment problems are too critical for the functionality of the system, they were recorded directly without exception.

time frame. However, as the number of reported bugs was significantly higher in the first and third release, the more significant statement that between 50% and 60% of bugs occurred within the first 48 hours after a release can be derived from this. Another noticeable cluster can be seen after six days (144 hours), during which approximately further 30% of the bugs were reported. In the second week of the first to fourth iteration, the remaining bug reports were received via the feedback form.

The patterns of bug detection after the releases suggest that the first 48 hours after a release are a particularly critical phase for bug detection. That around 50% to 60% of bugs are reported within the first two days after a release underlines the importance of this early phase. At the same time, it is noticeable that there is a time delay in the detection of bugs compared to the quickly reported deployment problems. This indicates that many errors are only discovered when the system is used more intensively or through specific user interactions. It can be deduced from this that the initial tests may not be sufficient to detect all problems immediately. Lessons learned from the bug reports could be:

1. **Intensively monitor the first 48h after a release**
   As the majority of bugs were reported during this period, it makes sense to intensify testing and monitoring directly after a release in order to identify and rectify as many bugs as possible at an early stage. This period is crucial for system stability, and it could be advisable to implement special measures such as targeted monitoring of the system during that period of time.

2. **Continuously monitor the system to detect the majority of bugs**
   That around a third of bugs still occur after six days indicates that some bugs are only discovered after prolonged use or under specific circumstances. This underlines the need for long-term monitoring and continuous feedback. System stability can be improved when making sure that a variety of scenarios is tested.

3. **Invest enough time in testing new features and adjustments**
   The renewed increase in bug reports in the third usability testing iteration indicates that neglected testing of new features and adjustments before a release can lead to new bugs. Sufficient testing should therefore be carried out to optimize system stability. This is also evident from the low number of bug reports in the last two iterations, during which sufficient testing was carried out.

6.1.3 Change requests

After analyzing the deployment problems and bug reports in the previous sections, this sub-section examines the change requests received. Figure 6.6 shows the number of reported change requests aggregated by testing iteration.

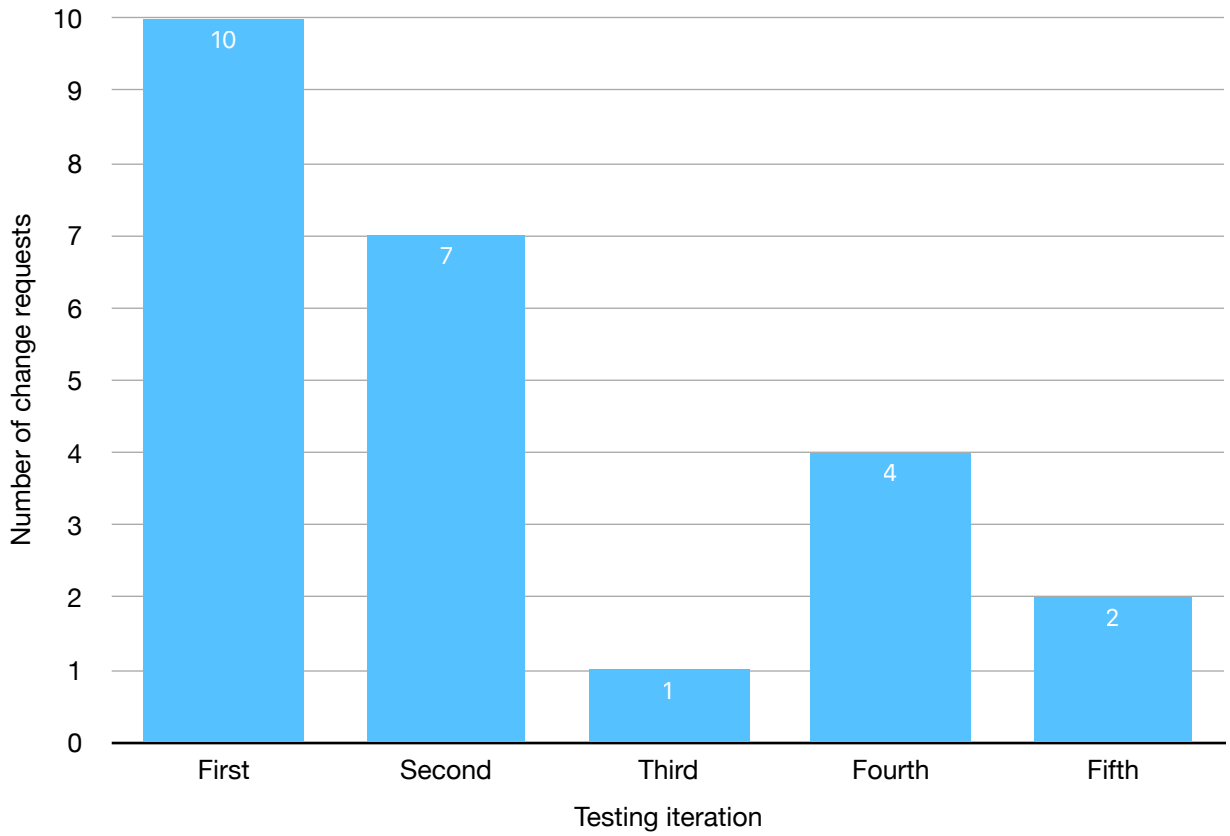


Figure 6.6: Number of change requests by testing iteration

A decrease in the number of change requests can be observed, although the number increased again in the fourth iteration. This development indicates that the existing components of the system were increasingly tailored to the needs of the users over the course of the experiment. The fact that primarily bugs were fixed during the first iteration indicates that the change requests from the second iteration also relate to the initial version of CheckedInn 1.0. That there were a similar number of change requests in the second iteration indicates that change requests in particular also occur after a longer and more intensive use of the system. This is also confirmed by the fact that the change requests in the third iteration are significantly fewer, with only one change request. This suggests that CheckedInn 1.0 was strongly tailored to the users during this iteration. The feature proposals and usability issues addressed in iteration two extended and adapted the existing system with non-basic features, which apparently initially resulted in further change requests after a more intensive use of the system. This is further supported by the fact that

several change requests occurred again during the fourth iteration. The fact that the existing components of CheckedInn 1.0 are eventually well tailored to user needs can be concluded from the few change requests in iteration five. The fact that, despite more intensive use, there were not significantly more change requests, as in iteration four, for example, could imply that the users and their needs were better understood during the experiment. As these were taken into account during the development of CheckedInn 2.0, they apparently required fewer further adaptations after new releases.

An analysis of the exact times at which change requests occurred, analogous to deployment problems and bug reports, would also be interesting in principle. However, most of the change requests were collected through the questionnaires and therefore asynchronously. Thus, there is no sample of change requests with exact times that would allow a significant analysis. However, the fact that most of the change requests were recorded by the questionnaires in conjunction with the fact that the questionnaires were usually completed at the earliest a few days after the start of a new iteration confirms the hypothesis that change requests tend to occur increasingly after a more intensive engagement with the system. Furthermore, unlike deployment problems and bug reports, which objectively describe errors in the system, change requests tend to be subjective in character. For this reason, change requests were not processed as soon as possible. Instead, they were first analyzed to determine whether it made sense to address them. To this end, CheckedInn 1.0, the preferences of the users and the potential effects of the adjustments were taken into account. In particular, change requests that were not submitted by several users and whose effects conflict with CheckedInn 1.0 were discussed with other users. Lessons learned from the reported change requests might be:

1. **Know your users**

   Understanding the users of the system, as well as their needs and preferences, can massively influence the number of change requests that may occur. Taking these into account during development can therefore minimize the need for further adjustments.

2. **Do not immediately address change request**

   Due to the subjective nature of change requests a proposed adjustment might not represent the preference of the majority of users. It can therefore be beneficial to first obtain sufficient feedback in order to gain a differentiated and detailed understanding of the users' needs.

3. **Consider the effects of implementing a change request**

   The impact of changes on the existing system could be to the disadvantage of existing

functionality or be in conflict with the purpose of the existing system. Therefore, a detailed analysis of the potential impact of changes is useful. A cost-benefit trade-off can also be useful.

6.1.4 Feature proposals

Now, in this section, the feature proposals collected during the usability study are analyzed. Figure 6.7 shows the number of feature proposals gathered grouped by testing iteration.

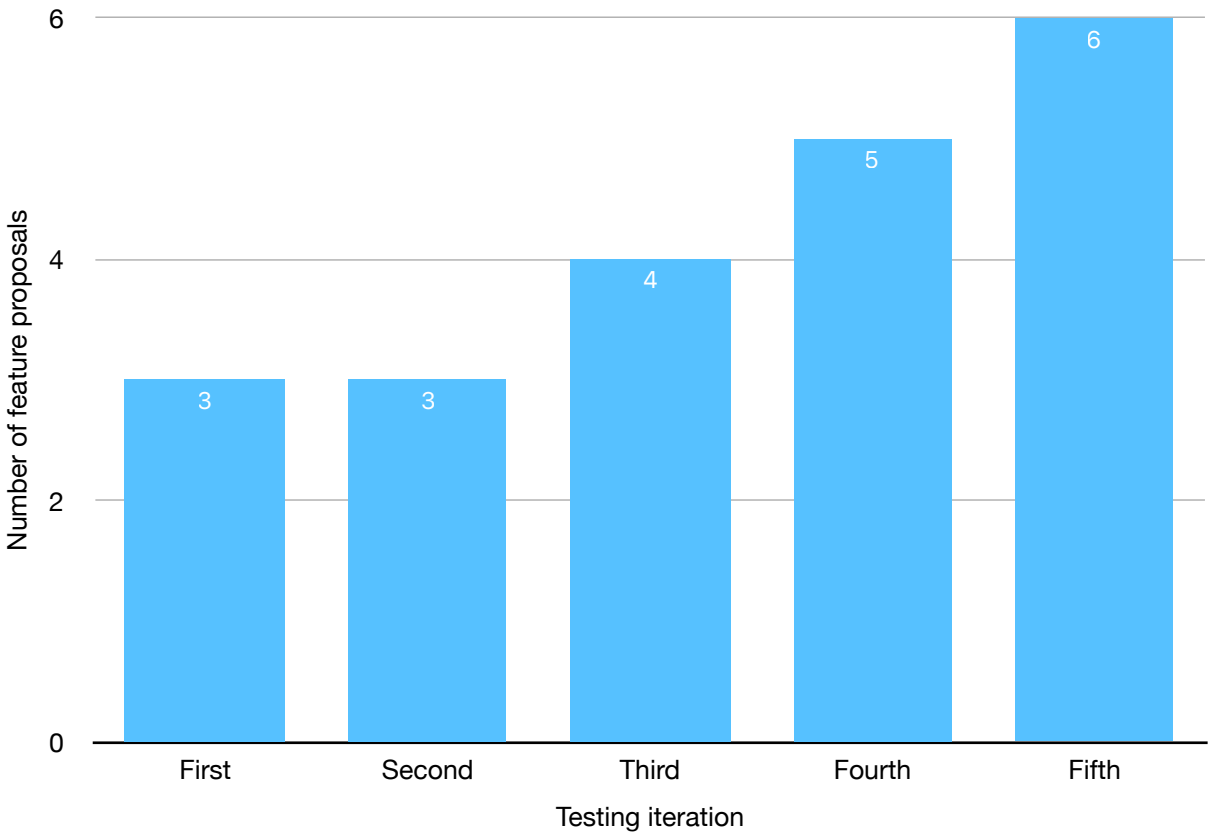


Figure 6.7: Number of feature proposals by testing iteration

When looking at the general development of the feature proposals submitted, a steady increase is noticeable. In the last iteration, six feature proposals were submitted, twice as many as in the first two iterations each. This steady upward trend in feature proposals from users strongly contrasts the development of previous types of feedback - deployment problems, bug reports and change requests - which always showed a downward trend towards the end of the experiment. It should be noted that the increase, or more specifically the non-decrease, in feature proposals can really be seen throughout each iteration. Unlike with previous feedback types, there are no iterations that break out of this linear pattern. This pattern could be explained in different ways. One possible explanation

could be that there is a positive correlation between familiarity with the system and the number of new ideas for extensions to the system. According to this, users would come up with new features as they use the system more and more. This could be because users get to know the system better the more they use it and are therefore able to better assess its strengths and weaknesses. Furthermore, more extensive use could lead to situations in real use cases that make the usefulness of the respective new features clear.

Another explanation could lie in the development of the types of feedback considered so far. Sections 6.1.1 to 6.1.2 showed that deployment problems and bug reports became fewer and fewer over time and were particularly numerous during the first iteration. The change requests also decreased in the overall trend. From this it could be deduced that the system became increasingly robust and better tailored to the users over the course of the experiment. Accordingly, the increase in feature proposals could also be due to the fact that users encountered fewer bugs while using the system and found less need for adjustments. Instead, the system was increasingly error-free and more customized, giving users enough temporal and cognitive resources to think about potential enhancements to the system - consciously or subconsciously. It should be noted that this argument may not necessarily be an alternative, but rather a complementary explanation to the previous theory that more intensive use leads to more ideas for new features, resulting in synergy effects. Furthermore, feature proposals could also reinforce each other. Accordingly, feature proposals could in turn give rise to new feature proposals. The reason for this could be that new features open up previously unconsidered scopes, giving users ideas for additional enhancements that make daily use of the system easier or increase the benefits that the system creates.

It would also be very interesting to look at the times at which feature proposals were submitted after a release. However, the problem here is the same as with the change requests, that a large proportion of the feedback was submitted via the questionnaires and therefore the exact times at which the feedback was submitted are not available. Accordingly, similar conclusions can be drawn as with the change requests. The fact that most of them were only received through the questionnaires indicates that feature proposals tend to appear after a certain amount of engagement with the system.

However, the feature proposals received were processed differently than the change requests. Although the necessity or usefulness of a proposed feature is subjective, its implementation has no direct impact on the existing components. While adjustments change the system, new features merely extend it. Even if some users do not consider a new feature to be necessary, it does not restrict the existing use of the system. Therefore, no considerations were made with regard to the current system as to whether a proposed feature would be implemented, but instead prioritizations and cost-benefit considerations

were made as to whether and when a proposed feature would be implemented. As part of this, it was investigated which feature proposals were received more frequently and which were very similar, as well as being discussed with other users of the system in order to determine which new features provide the greatest added value for most users. Lessons learned from the received feature proposals could be:

1. **Know your users and the existing system**
   In order to create and obtain a system that is as homogeneous and coherent as possible, it is very useful to know the end users as well as the existing system in detail. Especially when expanding the system, this can help to ensure that new features do not reduce the homogeneity and cohesion of the system.

2. **Analyze and prioritize which new features to implement**
   In order to maintain the homogeneity and coherence of the existing system, as well as to obtain the greatest added value from additional development efforts, it makes sense to carefully analyze and prioritize the feature proposals received. It is useful to know the user groups well and to discuss new features with beta testers.

3. **Use beta testers to assess quality of new features**
   Since users who submit feature proposals often have precise ideas about them, it can be very useful to involve users as beta testers during implementation. This can increase the likelihood that the new features actually correspond to the users' ideas and that the deviations between the developers' mental models and those of the users have less significant consequences.

6.1.5 Usability issues

Now that all types of feedback except for usability issues have been analyzed, this section deals with these. As these are the particular focus of this thesis, they were given special attention in the processing. Figure 6.8 shows the number of reported usability issues aggregated by testing iteration as a histogram.
In terms of the overall trend, it is noticeable that the usability issues steadily decreased over the course of the experiment. While a relatively large number of usability issues occurred in the first two iterations, only one usability issue was reported in the last iteration.

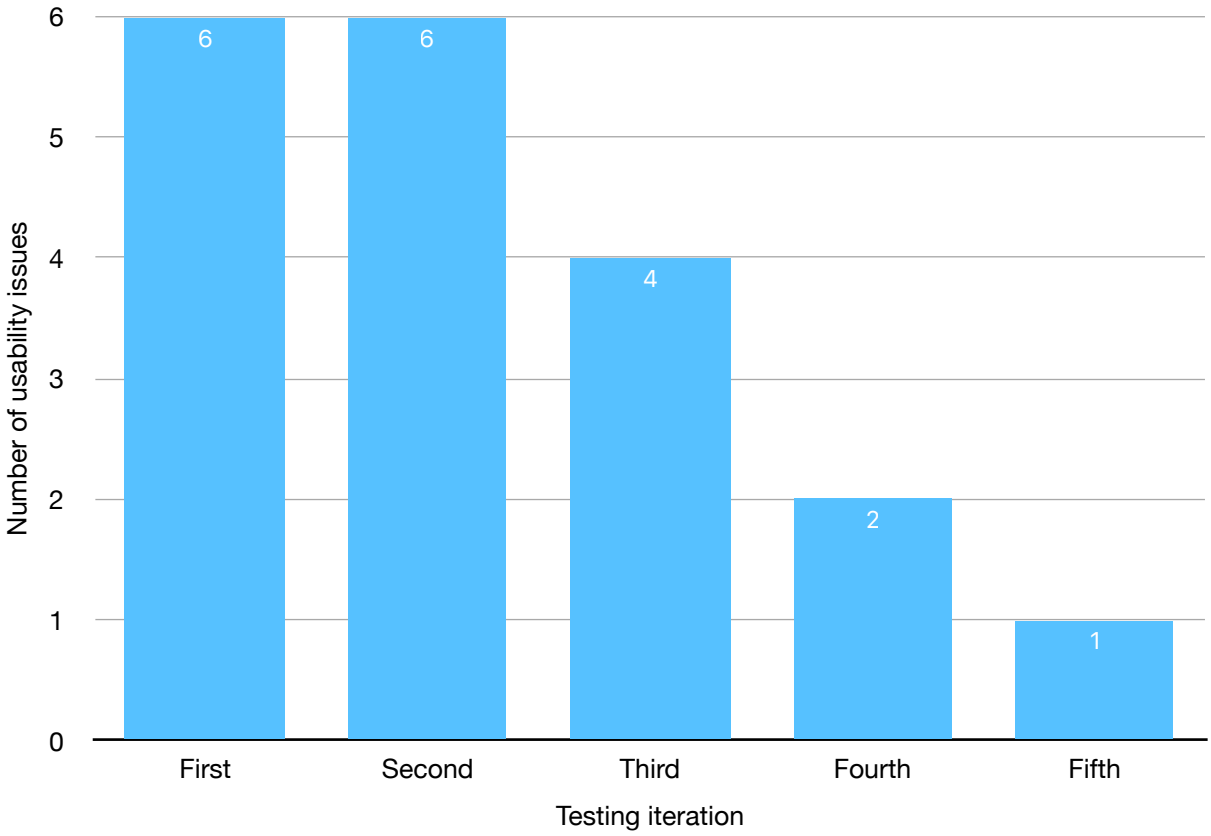


Figure 6.8: Number of usability issues by testing iteration

The high number of usability issues in the first two iterations indicates that the system was not yet precisely tailored to the users during these iterations. The reasons for this could be that the user groups were not yet fully understood due to a lack of experience or that certain patterns that make for good usability had not yet been sufficiently mastered. The comparably high number of usability issues in iteration three suggests that the adaptations and enhancements from the previous iterations were still not precisely tailored to the users. This could also be due to the fact that the focus in the first iteration was primarily on bug fixes, while in the second iteration there was insufficient time for testing and too little time to consider outstanding usability issues. However, it is interesting to note that the number of usability issues that occurred halved after the third iteration. This indicates that a turning point has taken place. As there was sufficient time available in the third iteration to develop new features and adapt existing ones, enough time could be invested to implement good usability. In addition, the decrease in usability issues could be due to the fact that a better understanding of users and their behavior was achieved through more intensive engagement with them. In general, the steady downward trend in the number of usability issues suggests that the usability of the system has improved more and more over the course of the experiment.

It would also have been interesting to investigate the exact times at which usability issues occurred. However, just like change requests and feature proposals, the majority of these

were reported via the questionnaires. Therefore, the exact times at which usability issues occurred are only available for a very small sample. Nevertheless, the fact that the users already had some experience with the system when completing the questionnaires indicates that usability issues occur more frequently with more intensive use of the system. Another argument against looking at the points in time at which usability issues occurred is the special procedure for dealing with them. As usability issues are even more subjective than change requests, the system was not adapted immediately after a usability issue was reported. In order to initially obtain a differentiated and comprehensive picture of the reported usability issues, the usability issues received after a release at the start of a new iteration were initially ignored. Only after the questionnaires of an iteration had been conducted with the users were the usability issues considered. These were then analyzed with regard to the heuristics and solutions were sought. Thus, there were neither sufficient usability issues from the feedback form, nor were these considered before the questionnaires were conducted. Therefore, the exact points in time, at which usability issues were submitted, are not examined. Lessons learned from this might be:

1. **Gather a comprehensive picture of the usability issues**
   In order to avoid unnecessary duplication of work and use the available resources wisely, it may be advisable not to tackle usability problems as soon as they arise. Instead, it may be beneficial to ignore them for the time being until a sufficient sample of usability problems has been collected.

2. **Use Nielsen's heuristics to categorize usability issues**
   In order to better objectify the subjectivity of usability, it can be helpful to orientate oneself towards usability heuristics when developing and adapting the system.

3. **Understand the behavior of users**
   Since usability can be highly subjective and can be perceived differently from user to user, it can be helpful to understand in detail the exact behavior of your user group and their perception of good usability.

In order to capture the underlying usability issues, Nielsen's heuristics were used. These help to make the subjectivity of usability issues more tangible and to assess the feedback more accurately. As the non-functional requirement usability is to be added to CheckedInn 1.0 and is therefore the focus of this thesis, a particular priority was placed on usability issues. Therefore, section 6.2 examines the collected feedback in detail with regard to Nielsen's usability heuristics in detail.

6.1.6 Cumulative analysis

Now that all types of feedback have been analyzed individually in the previous sections, this section provides a cumulative analysis of the development of the different types of feedback. This is intended to show once again how the different types of feedback develop in relation to each other and what this might imply for the development of the system during the experiment. Figure 6.9 shows the amount of feedback received per feedback type, aggregated by testing iteration. The linear functions shown only serve to visualize the development of the amount of feedback in the individual iterations. This is a discrete amount of data, whereby each iteration is to be understood as an independent point and there is no continuous development between the iterations.

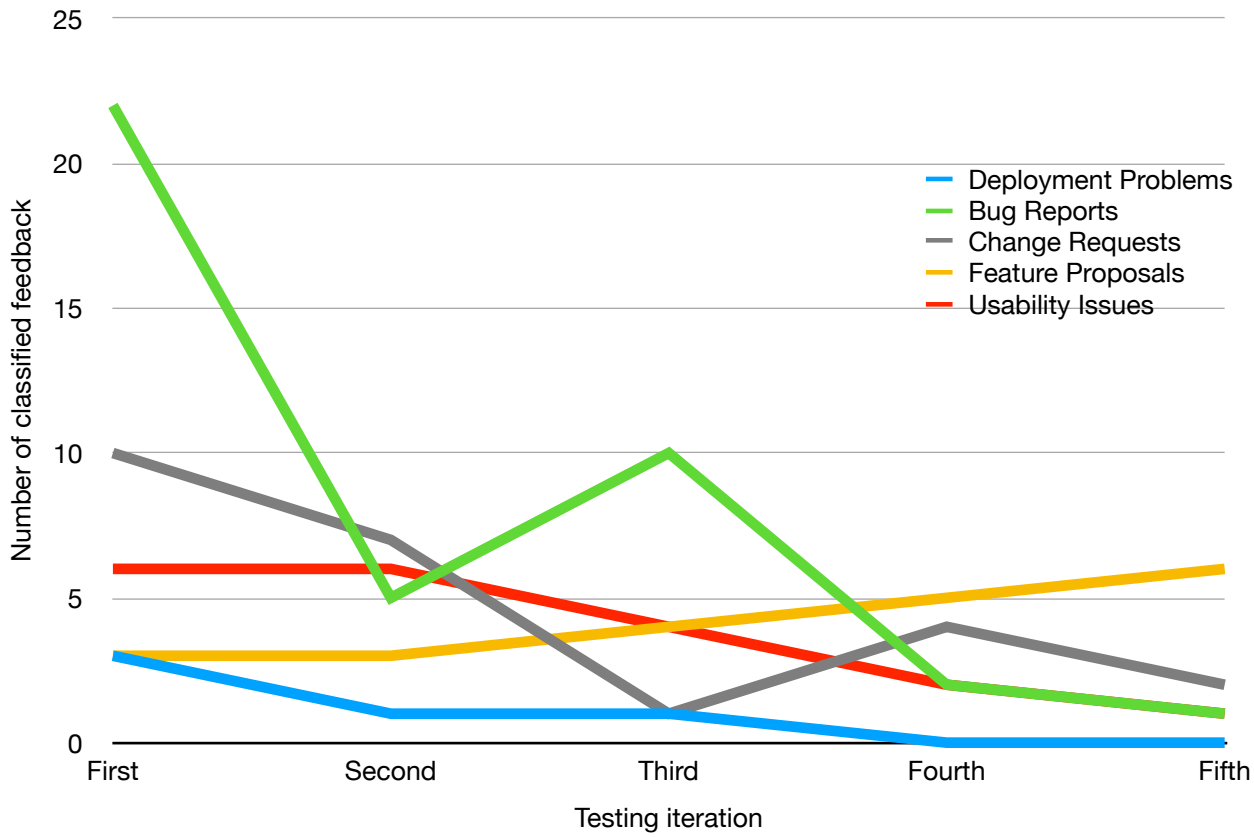


Figure 6.9: Number of feedback types by testing iteration

Overall, a decrease in the amount of reported deployment problems, bug reports, change requests and usability issues can be observed from the first to the last iteration of the usability experiment. Although there was a renewed increase in the amount of feedback reported in some iterations, the feedback received over the entire experiment was nevertheless less. Only the feature proposals showed a steady increase over the course of the experiment. The high number of bugs and change requests at the beginning of the

experiment indicates that the system was not yet optimally adapted for actual use in the first iteration. This can be attributed to the fact that user requirements and actual use cases had not yet been sufficiently incorporated into the development at that time. Over the course of the experiment, however, a significant decrease in deployment problems, bug reports, change requests and usability issues was observed. This development suggests that the system became more stable, more coherent and better tailored to user requirements over the course of the iterations. The steadily increasing number of feature proposals may indicate that users felt more familiar with the system and increasingly suggested ideas for enhancements rather than addressing existing problems. This suggests that new features were also better developed over time and required less adjustment. In addition, the improvement in usability issues indicates that the principles of good usability were better internalized during the experiment, which had a positive impact on the usability of new features and adjustments. This is largely consistent with the findings from the individual analyses of the feedback types. The most significant lessons learned are:

1. **Introduce an additional test phase before the first release**
   Since many bugs and deployment problems occurred, especially in the first iteration, it might be advisable to establish a further testing stage before the first release.

2. **Use beta testing for changes and new features**
   In order to better understand users and avoid the need for repeated revisions, it can be useful to coordinate adjustments and new features with users as beta testers already during the development process.

3. **Know your users**
   In order to create and obtain a system that is as homogeneous and coherent as possible, it is very useful to know the end users as well as the existing system in detail. Especially when expanding the system, beta testing can be a very useful tool to do so.

As all these conclusions only relate to the period of five iterations, it would be particularly interesting to explore how the feedback develops over a longer period of time and whether the trends identified confirm or even new patterns emerge. Especially with the even more extensive expansion of the system, it would be interesting to see whether this continues to function this smoothly.

6.2 Usability Heuristic-based evaluation (SL)

This chapter provides an in-depth quantitative analysis of the survey results carried out as part of the usability tests in accordance with Nielsen's ten usability heuristics. The survey used a Likert scale from 1 to 5 to rate statements about usability and a scale from 1 to 10 for the overall usability assessment. The analysis focuses on categories with significant changes in the survey results in order to identify relevant trends and areas for improvement. The changes in the remaining categories can be found in Appendix D. Finally, the development of the average overall usability rating of the system is examined in order to assess the influence of the system adjustments made on user satisfaction.

6.2.1 Visibility of the system status

Figure 6.10 shows the average progression of the values that could be achieved in the tests for heuristic 1 `Visibility of the system status`. The heuristic plays a decisive role for operating software in particular. As operating procedures in hotels are very process-oriented, it is particularly important for users to be able to track the current status of the system at all times and to be notified of any changes. This should be given particular attention in the context of payments and other finance-related actions. This is because even small errors can have a major impact on the hotel's finances. For this reason, visibility and transparency in the context of the software is particularly important.

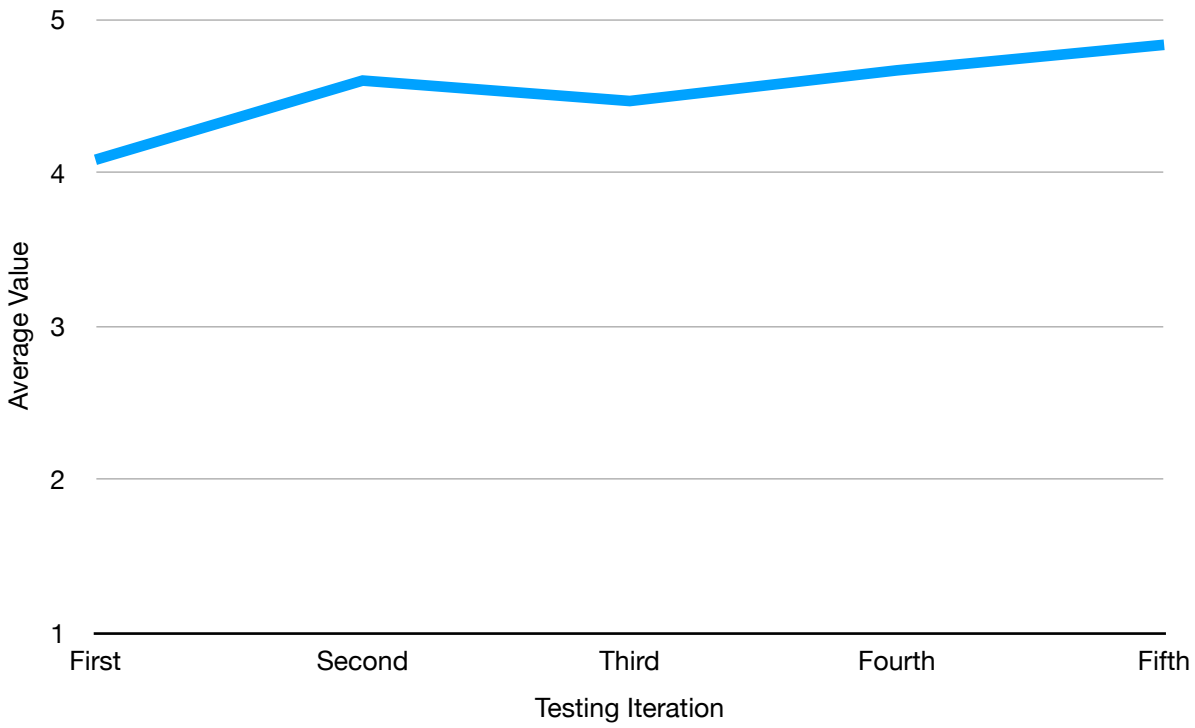


Figure 6.10: Development of the average valuation of H1

Further, it is not only important in terms of efficiency, but also in terms of error prevention. This is because errors can occur particularly frequently due to a lack of information or ambiguity. With this in mind, great importance was attached to visibility when the system was first implemented. This is also reflected in the initially high rating of heuristic 1 in the first usability tests. A score of 4.083 was already achieved in the first iteration, which indicates a solid basic configuration of the system in terms of status visibility. In addition to the positive feedback, participants provided specific suggestions for improvement.

A commonly voiced concern was the need to prolong the visibility duration of notifications. Many participants observed that the notifications vanish too swiftly, raising the likelihood of missing critical information. Additionally, the color of the notifications was highlighted as an area for enhancement. Participants recommended using different colors to indicate the significance of each notification rather than relying on a single color. This change would improve the visual clarity of messages and emphasize their relevance. As a result of this feedback being implemented, the score rose to 4.6 in the second iteration.

The feedback presented after a reservation was completed received particularly good scores from the participants. These status messages allowed employees to easily grasp the progress and confirmation of a booking, which greatly enhanced the overall user experience. Following the second release, some minor modifications were implemented, particularly in relation to adding and adjusting additional services for current reservations. In the third round of usability testing, however, the average score slightly dropped to 4.467, a decline linked to updates in the reservation functionality. The statement `Status updates are visible when changing the booking or service details` was rated less positively by the participants than in the first two iterations. This indicates that the visibility of the system status for certain changes in the booking process was not as well implemented as expected. After adapting the feedback provided and implementing new features, a score of 4.667 was achieved in the fourth testing iteration. The fact that the history provides a good overview of all actions carried out within a reservation was highlighted as particularly positive. The history makes it possible to see exactly which user carried out which action at which time. According to the participants, this has particularly promoted confidence in the traceability of the system. In the fifth iteration, a value of 4.833 was achieved.

The overall improvement over the five iterations can be attributed to various factors. On the one hand, the continuous improvement of the system has led to a better user experience about heuristic 1. On the other hand, however, it is also true that users become more familiar with the processes in the system over time. As a result, they are also increasingly less dependent on system statuses and notifications. However, the fundamentally positive

feedback in iteration one suggests that the system performs well in the underlying heuristics, despite the bias.

6.2.2 Match between System and the Real World

When looking at the other heuristics, the development of heuristic 2 `Match between System and the Real World` was particularly striking. With the focus on user-friendliness, intuitiveness and the ability to quickly familiarize unexperienced employees are particularly important in the context of hotel software. For this reason, terminology and the use of semantically coherent icons are particularly important.
Figure 6.11 shows the development of the average values of heuristic 2 over the five test iterations. At the beginning, a value of 4.375 was achieved in the first iteration. On average, statements 1 and 4 received the better values. These statements relate to the consistency of language in day-to-day operations and language in the software as well as the natural arrangement of information within the software.

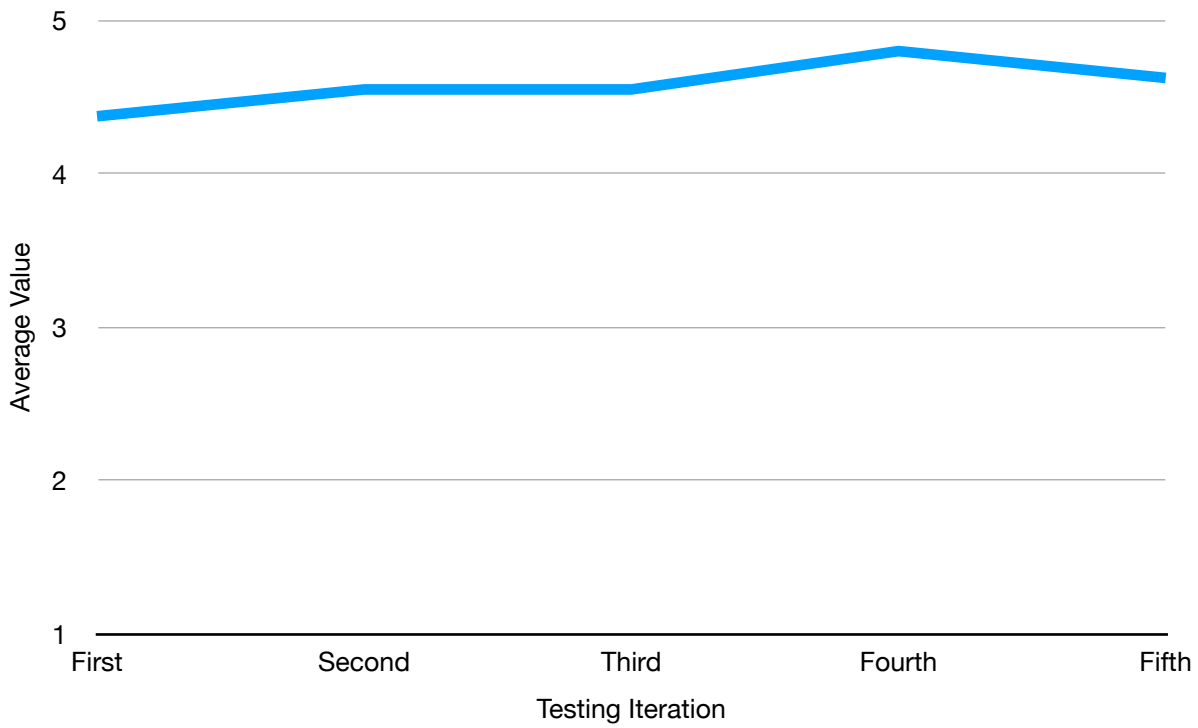


Figure 6.11: Development of the average valuation of H2

In the second iteration, this value was increased slightly to 4.55. It is noticeable here that no significant changes were made to terminology or syntax in the change to this iteration. Only colors and font sizes were changed so that users were made more aware of certain terms. This nudging towards terms that are familiar to users from the industry was also mentioned as positive feedback in the second iteration. The users, who had only been

using the new software for a few days, found it easier to carry out processes based on intuition because no new terminology had to be learned.

Towards the third testing iteration, there were several major changes within the software. With great care taken to retain the terminology commonly used in the industry in the new features, the value of 4.55 also remained unchanged. It is noticeable that the simplicity for unexperienced users in particular was emphasized more positively in the new features (statement 2). The logical arrangement (statement 4), especially in the new features, was also praised by several employees.

In contrast, there was a significant drop in the average rating for statement 3 (Icons and symbols are intuitive and related to real-world hotel operations) with an average value of 4 in the third iteration. This can be attributed to the fact that a new form of button was increasingly used in the third release in the spirit of minimalism. The new text buttons have a much simpler design and only consist of colored text. With a mouseover, the text is underlined and slightly enlarged and the cursor becomes a pointer. While this statement received good ratings in terms of aesthetics and clarity, this was potentially at the expense of heuristic 2. For this reason a text button grouped with an icon has been introduced. This increases visibility and also offers greater readability due to a higher intuitiveness factor through the icon. As a result, the average rating of statement 3 increased by 20% to 4.8 in the fourth iteration. The overall value in the fourth iteration was also 4.8, meaning that an increase was also recorded in the overall view of heuristic 2.

Despite the positive development, there were also suggestions for improvement in the fourth iteration. This concerned the order in which the prices are displayed in the modal for new reservations. Until the fourth iteration, prices were always set and edited via the prices menu item. This is used to link prices to booking platforms via the channel manager. However, these prices also serve as a guide for walk-in customers. If the flat rate paid for a reservation for a booking platform is 20% per reservation, for example, it makes sense for the hotelier to always set prices 20% below the price set for the Channel Manager. If demand is lower, higher prices are naturally more appropriate. In any case, the general price orientation of the hotel, which is defined in the menu item prices, serves as a helpful support, also for reservations made on site. Until the fourth test iteration, the user did not receive any direct insight into the prices for the selected dates from the modal for a new reservation. These were only displayed via a tooltip (a small popup that is opened via a hover on an object). The inputs for the prices for a new reservation were empty by default. This order was perceived as unintuitive, as it would be more logical to display the prices from the pricing by default and not empty fields. After all, the fixed prices are needed as a guide anyway. This would allow the user to make adjustments based on them and not have to find out the prices via the additional hover. In consultation with other users, this feature

received very positive feedback and was implemented in the fifth iteration. This was also noticeable in the increase to 4.75 in statement 4 of heuristic 2. A slight decline was again evident in statement 3. This is due to a poor rating of 2, in which the user was a new employee at the hotel and had migrated from old software that used almost no icons. For this reason, these could not provide any support in terms of intuitiveness. However, with an overall value of 4.625, a clearly recognizable increase compared to the evaluation in iteration 1 could still be identified.

6.2.3 Error prevention

Heuristic 5 `Error prevention` plays a particularly important role in ensuring the correctness and security of the software. As with heuristic 1, the aim here is to ensure stability in use. This applies in particular to actions related to financial transactions or reports, as even small errors can have a major impact here. Heuristic 5 deals with the prevention of such errors and the stable implementation of error avoidance. The course of the average score is shown in Figure 6.12.

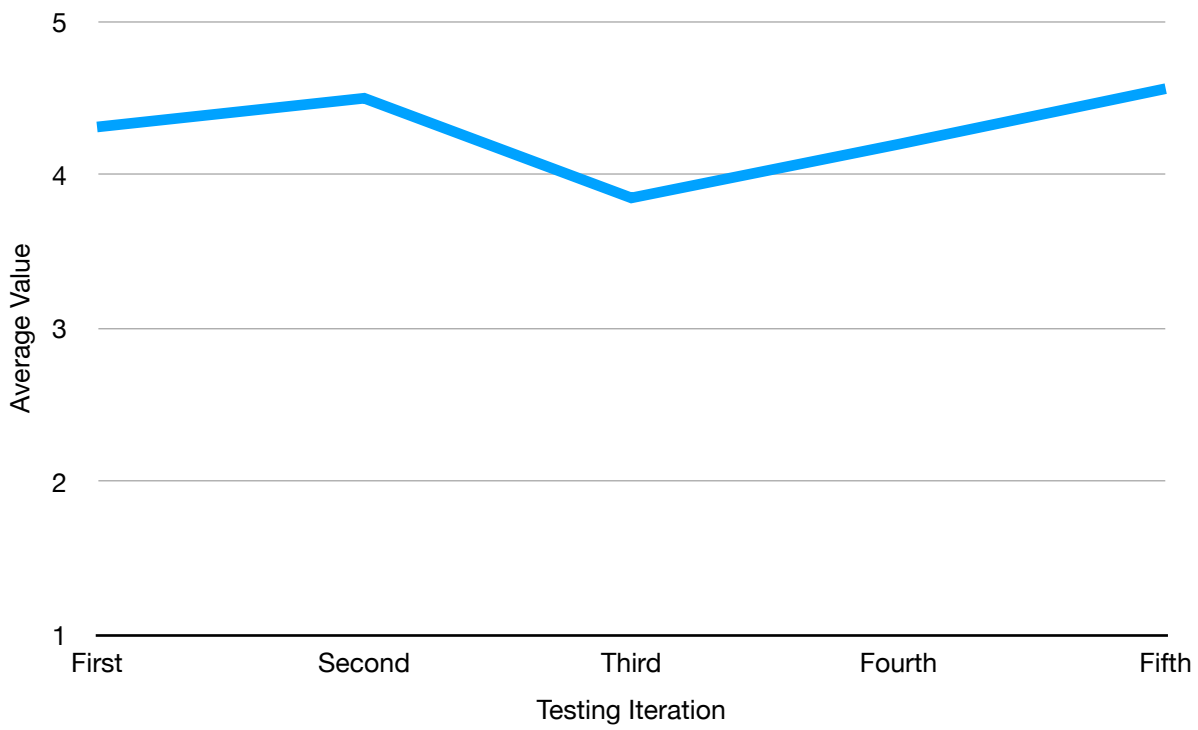


Figure 6.12: Development of the average valuation of H5

In the first iteration, this heuristic received an average score of 4.313. While statement 1, 2 and 4 all scored between 4.5 and 4.8, statement 3 `Confirmation prompts appear before finalizing potentially important actions` was seen as particularly critical. Due to, according to feedback, too few safety mechanisms for critical actions. At the same

time, participants not only wanted to be made aware that the action was critical. It would also be helpful to be shown what the specific consequences of this action would be. As a result of this feedback, all system-critical actions were provided with an alert dialog. This opens as soon as, for example, a button is pressed that is supposed to trigger a critical action. The alert dialog then informs the user that they are about to perform an action that could have far-reaching consequences. The user can then choose whether they want to complete the action or cancel the process. All actions that involve either the deletion of objects, such as employee accounts, were considered system-critical. This also includes all actions that are directly or indirectly linked to payments or prices. If a reservation is shortened on the room plan, an alert dialog opens, as shortening the reservation also results in a new price (the prices of the shortened nights are removed from the total price). Finally, actions that imply other actions are also included. Once a guest has checked out, it should no longer be possible to add services. However, if the user wants to do this anyway, adding the service would imply canceling the check-out. For such actions too, the changes made to the system now prompt the user with an alert dialog asking whether they really want to perform the action.

These changes affect the system across the board and have therefore also become highly visible to users. This was also shown in the second test iteration with a significantly higher value for statement 3. With an increase of 37%, statement 3 received a score of 4.8. The overall score for heuristic 5 was 4.5 in the second iteration, which was an increase compared to the first iteration. A comparatively lower value was achieved in the second iteration with statement 2 `The system validates data entries to prevent common input errors`. With a value of 4.2, this could be due to feedback that emails are not validated for correctness. Accordingly, any input can be selected for email inputs without the system informing the user that this email does not exist. The actual existence of an email is only possible via an email provider. As a result of the feedback, however, a regex[13] check was built in, which validates the email input for its meaningfulness.

In the third test iteration, there was a sharp drop in the score. With an average overall score of 3.85, this is the worst result in the course of the tests for heuristic 5. In consultation with the participants, this is primarily due to a new bug that has a major impact on the system. An error has occurred in the system due to changes to the connection to the channel manager. As the connection to the channel manager is implemented via a Rest API and not sockets, asynchronous behavior can occur. For this reason, overbooking can occur in hotels. In the event of overbooking, more rooms are sold for one night than the hotel actually has. In this case, the hotel is obliged to provide an equivalent replacement in

[13] Regular expression

accordance with Section 651i BGB[14]. Organizing an equivalent replacement is not always easy, especially in times of high demand. For this reason, it is important to inform hotels about overbookings in good time. In the bug in the third test iteration, the overbookings were displayed on the room plan and marked as such, but the notifications that are usually sent to the user were not received. This can have financially unfavorable consequences for the hotel, as they can only book an alternative room in another hotel for the overbooked guest at a later date due to the delayed information. This can result in higher prices and ultimately negative business. With a bug report and rapid implementation, the problem was solved by the fourth test iteration.

In the penultimate iteration, the average overall score was 4.2. The increase in the rating in statements 4 and 3 explains this increase significantly. A small decrease from 4 to 3.8 can be seen for the average overall rating of statement 1. Exact causes for this decline could not be determined. For the final iteration, some changes were again made to the system. Firstly, the stability and consistency of the data were increased. Database checks are used to check whether data is consistent and any errors are reported to the user. For example, it is now no longer possible for users to check out guests who have not yet been charged for all services or to give reservations and the rooms they contain a different status. In addition, the guest application for the registration form has been expanded to include several input checks for telephone number and nationality.

For the final iteration, some changes were again made to the system. Firstly, the stability and consistency of the data were increased. Database checks are used to check whether data is consistent and any errors are reported to the user. For example, it is now no longer possible for users to check out guests who have not yet been charged for all services or to give reservations and the rooms they contain a different status. In addition, the guest application for the registration form has been expanded to include several input checks for telephone number and nationality. Finally, the susceptibility to errors was also reduced and the quality of the error messages on the room plan improved. These innovations could be one reason why Statement 1 and 2 in particular achieved significantly better results than before in the final evaluation round (increase from 3.8 to 4.5 and 4.2 to 5.0). Statements 3 and 4 achieved results comparable to those previously achieved, which meant that the overall average of heuristic 5 was finally evaluated at 4.563. While there was always a clear upward trend in the previously discussed statistics, Heuristic 5 shows a clearer correlation between large, system-relevant errors and the score. This may indicate that the positive increasing ratings of H1 and H2 are causally related to the changes in the software. The influence of increasing user experience cannot be refuted, but the thesis that increasing

[14] Bürgerliches Gesetzbuch

user experience is the only argument for increasing scores seems less likely when looking at the development of the rating of H5.

6.2.4 Overall Usability

Overall usability is asked at the end of the questionnaire. On a scale of 1 to 10, users are asked to use this metric to assess their overall impression of the usability of the software. The test participants were informed that although they can use the metrics assessed in advance as a guide when assessing overall usability, they do not have to rely on them alone. The purpose of this is to break the bias from the frame set by the heuristics and to allow other impressions beyond the heuristics to be included in the evaluation. Any other positive or negative features of the system, as well as the subjective feeling when using it, should also be taken into account in the overall assessment. Figure 6.13 shows the average course of the overall usability over the five test iterations.

An average value of 8 was achieved in the first iteration. This is a good result in absolute terms, but around 7% weaker relative to an average of 4.388 in the heuristics. One possible reason for this is that the software was still very new to the test participants in the first iteration, and many ideas for enhancements and improvements were generated that were outside the scope of the heuristics. Analogous to the good absolute evaluation, the feedback in the optional text field for the overall evaluation was also predominantly

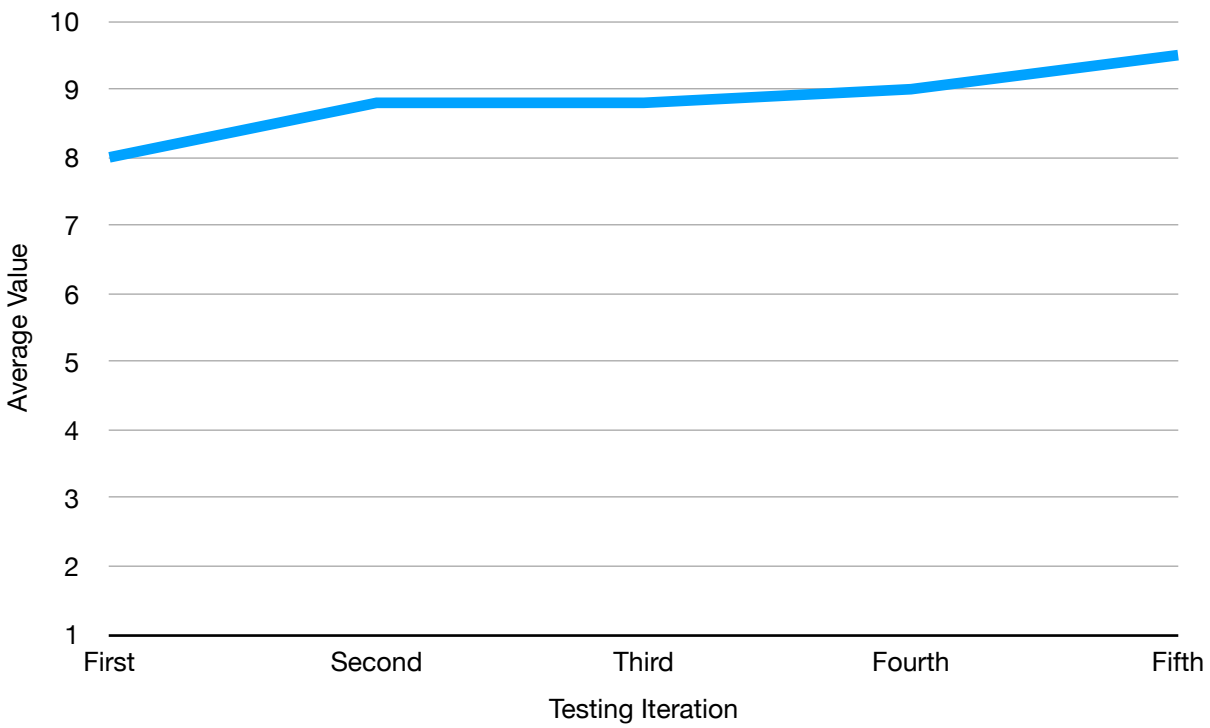


Figure 6.13: Development of the average overall usability valuation

positive. It was noticeable that a large number of ideas were submitted here. This once again sensitized to the fact that the free form text is of great relevance, as it offers users the opportunity to think more freely and contribute ideas.

With a large number of predominantly smaller changes, a value of 8.8 was achieved in iteration 2. Overall, the heuristics were assessed with an average value of 4.522. The increase in both values could potentially be explained by the strong increases in the ratings of heuristic 1 (+12.653%) and heuristic 9 (+8.308%). The latter is possibly due to minor changes in the handling and description of errors. With the strongest decrease in heuristic 4 of 3.856%, there were no major decreases. Relative to the maximum achievable value, the average of all heuristics of 88.041% is significantly closer to the overall usability of 86.667% than in the previous iteration. One possible explanation for this is that users became more familiar with the system and were better able to categorize their feedback in the heuristics. Against this background, new ideas appeared less frequently in the free text for overall usability. For this reason, an approximation of the relative values of the average heuristic rating and the overall usability is conceivable.

In the third iteration, there was a decrease in the average rating of all heuristics with a total value of 4.438. However, the users' assessment of the overall usability remained constant at 8.8. The decline in the heuristics can be explained primarily by a drop of 14.443% in heuristic 5 (see section 6.2.3). There were hardly any increases in the heuristics in the third iteration. As explained, the decline in heuristic 5 was due to an error in the channel manager connection. This does not appear to have had any influence on the assessment of usability. This could be an indication that the general understanding of usability extends over a wider scope than just the correctness of the software. According to feedback, the fact that the software is visually appealing and lightweight is a particularly important component. And while ensuring the correctness of the software is essential, this iteration shows the high demand from employees for modern software in the hotel industry.

Two weeks later, the average overall usability rating increased to 9.0 in the fourth iteration. This increase correlates with the increased average heuristic rating of 4.553. The increase in the average heuristic rating is due to various factors. On the one hand, heuristics 5 and 7 recorded significant growth (9.091% and 9.524%). On the other hand, the high absolute ratings of heuristic 2 and especially heuristic 8 (4.8 and 4.95) were also responsible. This indicates a high level of user satisfaction with the software, which is also reflected in the overall usability. In a relative analysis of the values in relation to their possible maximum values, the two scores are almost identical at 88.833% and 88.888% respectively. This supports the theory that users are better able to categorize issues and recommendations in the heuristics as they become more familiar with the software. This means that developers

are better able to identify the scope of the vulnerabilities, but is potentially at the expense of the users' detached way of thinking in the evaluation.
In the fifth and final iteration, usability was rated with the highest value of 9.5 in the course of the study. The increase in the initial rating from 8 to 9.5 not only underlines how important direct user feedback is, but also that small changes can have a major impact on user acceptance, particularly in terms of usability. The heuristics received an average rating of 4.585 and therefore also received the best values in the course of the study. Once again, the increase compared to the previous iteration can be explained by an increase in the rating of heuristic 5, but users also rated other heuristics very highly, including heuristics 4 and 8.

This section has shown that the assessment of users of the software has increased over the course of the study. Overall usability increased steadily from an initial rating of 8 to a final value of 9.5. Clear upward trends were also evident in the majority of the heuristics. Heuristic 5, for example, is an exception here.
The finding from the heuristics-based evaluation is that proximity to the user has undoubtedly played a decisive role in improving user-friendliness. Through short communication channels and quick adjustments, the software was increasingly tailored to the needs of the users. At the same time, the study and the feedback, particularly for the simple design of the software, also show the high demand for user-friendliness and simplicity in hotel software. A change in the way users evaluate the software was also evident. With increasing experience in using the software, mental structures were much more closely linked to the predefined heuristics and there was less feedback about the free text at the end of the questionnaire. This indicates that it makes sense to test the software in the early phases so that users do not become too accustomed to the existing software, but rather go down the path of further development together with unbiased beta testers.

6.3 Threats to Validity

In the following section, the results of the usability study conducted in sections 6.1 and 6.2 are examined in detail and critically scrutinized. While the analysis and the findings gained from it provide valuable insights into the user experience and the functionality of CheckedInn 2.0, it is now important to identify potential threats to validity. These are threats that could influence the validity and generalizability of the study. These threats can come from various sources, such as the test methodology, the selection of participants or external influencing factors. In addition, the critical examination of the results not only

serves to validate the study results, but also to derive recommendations for future studies and development processes.

Sample size

A clear factor that speaks against the generalization of the results is the fact that a comparatively small sample size of 13 hotels was examined. This is initially due to the approach of discount usability testing, which is aimed at precisely such small samples. This form of testing offers a variety of advantages with its quick insights into basic usability aspects. At the same time, the sample may not reflect the diversity of the actual user population. Different user groups, such as experienced and unexperienced users or people from different demographic backgrounds, may have different needs and behaviors. The risk of overlooking specific usability issues that are relevant to a broader user base is therefore increased.

Other problems can also arise from the small sample size. Hotels have very process-oriented workflows. Although these processes are comparable in the overall view, they can differ in detail from hotel to hotel and contain deviations from one another. Accordingly, it is possible that only one specific interpretation of business processes was optimized in terms of user-friendliness during the course of the study. It is therefore possible that other hotels with other types of processes may not benefit from these changes at all or that the changes may even have made their processes more cumbersome. Although the latter in particular is very unlikely in the scenario of this study, it shows how important heterogeneity is in the choice of sample. Finally, small samples also run the risk of change requests not being representative of the total quantity. This is because it is much more difficult to recognize the significance of an opinion with a small sample size, which is why the meaningfulness of feedback suggested by just one person must be questioned all the more.

Environmental influences

Another factor that plays a role in a large number of all studies and can call validity into question is environmental influences. These include, for example, the Hawthorne effect [26]. This states that people change their behavior when they know that they are being observed. This can lead to unnatural user behavior and thus also change the personal perception of a test participant. As the tests in this study were always carried out in close proximity to the tester and participant, it is possible that the Hawthorne effect could have distorted the results.

The test environment may also have had an influence. In the case of the study, this was mostly the reception, so essentially the natural working environment of the subsequent

users. Nevertheless, external influences such as weak hardware, problems with the internet or noise and distractions may have affected the concentration of the participants and thus distorted the perception of usability. Finally, personal reasons of the test participants may also have brought variance into the respective assessment of usability. Such reasons may relate to the emotional state of the participant, which can be expressed through stress or time pressure. However, social factors can also play an equally important role. For example, it is possible that test participants do not feel comfortable criticizing a product that has only just been introduced by their supervisor. Although users were made aware of the anonymity of the results and the relevance of critical feedback in the run-up to the study, it is still possible that test participants were intimidated to give their honest opinion.

Subjective perception

Subjective perception is a decisive threat, particularly in the context of this work. While there are tendencies towards standards that can define good usability, usability is still very much characterized by individual perception. This can also be influenced by physical conditions. For example, a certain font size may be ideal for one user but clearly too small for another. The lack of objectifiability is therefore a factor that could be used to question the quantified results of this work. This fact is compounded by the fact that this study is based on an absolute perception. As only the CheckedInn PMS was tested, the users had no direct comparison. They therefore had to make an intuition-based judgment as to how good or bad they would rate the respective statement. This factor is reinforced by the fact that an assessment in the Likert Scale is also very subjective. On a scale of 1 to 5 with the scores strongly disagree, disagree, neutral, agree and strongly agree, the evaluation is not exactly tangible. For example, a user who was satisfied with something may tick strongly agree, whereas satisfaction is the standard for another user and they would therefore only tick neutral.

It can be concluded from these factors that although the results of the study are not significantly distorted by subjective perception, the assessments should always be viewed within the framework of a certain variance due to the difficulty of tangibility and quantifiability.

Individual learning curve and adaptability

The last relevant threat to validity concerns the personal skills and mindset of the participants. The user's ability to learn plays a key role here. One goal of usability is quick and easy familiarization with the system. However, this is only possible if the user has the ability to familiarize themselves with a new system. This requires a basic technical understanding on the one hand, but also the ability to abstract. In concrete terms, this

means that a user does not try to recreate processes that they know from other software, but instead follows the flow of CheckedInn and tries to familiarize themselves with it. In addition to the skill of the ability to learn, this also goes hand in hand with the ability to adapt and the willingness to migrate. If a user does not want to change the software from scratch and is not prepared to get involved with a new system, the objectivity of their statements regarding usability must be questioned. In this case, it cannot be assumed that the user will perceive the software neutrally, but will view any changes as negative, even if they are not necessarily bad. This also means that the way in which the software is introduced and the training provided is of great importance. If a user feels compelled to use a new system, this compulsion can lead to resentment. In the same way, a lack of training and too little time spent familiarizing users with the new software can lead to them feeling uncomfortable in the new environment and unable to find their way around.

The usability study provided valuable insights into the user-friendliness of the software and the user experience. Key learnings can also be derived from the identified threats to validity.

When conducting studies, it is important to consider the cost-benefit ratio. The larger the sample size, the easier it is to generalize. However, it should also be noted that a larger sample size results in significantly less flexibility and agility in the implementation of the feedback. And while the latter is a central point in the implementation of discount usability testing, the number of people should be in proportion to the size of the system. It remains to be investigated whether it would make more sense to test with 10 people per iteration for systems of above-average size, as was tested in this study, in order to ensure the agility of discount usability testing. This would also ensure greater generalizability of the data. Regardless of the sample size, attention should always be paid to heterogeneity within the sample. This allows a variety of user perspectives and opinions to be obtained with little effort.

From the experience of this study, reducing environmental influences is a trade-off between ensuring the correctness of the execution of the tasks and minimizing the risk of a Hawthorne effect. If the user already has experience with the system and the tasks are clearly pre-formulated, it can be a great advantage for the test person to carry out the test alone, if necessary with a short initial briefing. Conducting the tests alone offers the user the advantage of fewer environmental influences and therefore potentially less falsified results, while at the same time reducing the tester's workload. However, if the user is inexperienced in using the system, it is advisable for the tester to be present in order to ensure the quality of the test.

Finally, it is recommended that the use of operating software in particular should always be well trained. This gives users a sense of security when using the system so that they can better focus on user-friendliness. In the case of extensive systems, training should take place in advance of the usability tests in order to ensure that participants have a basic understanding of the system's functionalities.

This training can take various forms. It can be a workshop, video tutorials, FAQs and short popovers with help instructions that can be accessed from anywhere. This can ensure that the differences in the users' learning curves are at least partially neutralized, thereby neutralizing the results of the usability tests.

7. Conclusion (SL)

"[...] usability is a necessary condition for survival. If a website is difficult to use, people leave."

Jakob Nielsen, 2012

As Jakob Nielsen points out, the relevance of usability in the context of websites is very high. While in the context of this quote he is primarily referring to users who voluntarily use the website, this way of thinking can also be applied to hotel software. Chapter 2 shows the 5 aspects of the definition of usability that he uses. While a casual user is potentially particularly interested in satisfaction in the context of user-friendliness, efficiency of use, learnability and error handling are all the more relevant in the context of hotel software. The hotel industry is digitally obsolete and therefore particularly reliant on user-friendly solutions to drive efficiency in day-to-day operations.

The aim of this thesis was to examine and systematically optimize the user-friendliness of CheckedInn 1.0. Chapter 4 showed the original requirements as part of the inventory analysis of CheckedInn 1.0. With usability as a missing requirement, this was identified as a weak point of the system. Given this, the system could then be examined with a more specific focus on usability. Chapter 4 also laid the foundation for reverse engineering in order to gain a better understanding of CheckedInn 1.0 as a whole. This step also proved to be particularly helpful in order to maintain consistency in changes to CheckedInn 1.0 in the event of subsequent minor adjustments and further outstanding forward engineering phases, which resulted in the new system CheckedInn 2.0. Finally, the analysis considerably simplified communication by documenting the requirements.
The subsequent reverse engineering in chapter 5 made a significant contribution to understanding CheckedInn 1.0's structure. Thanks to the clear documentation of the internal system structure, it was easier to draw conclusions from mechanisms in the system to the source code itself in subsequent code adaptation processes. Optimizing the system according to specific use cases proved to be particularly helpful. In the course of the study, it became clear that feedback was often related to specific use cases. Having the documentation for the respective use case already available in modular form made adjustments considerably easier. It is reasonable to assume that this will also be the case in further forward engineering projects. In addition, dependencies and connections between components of CheckedInn 1.0 were identified. It has shown how data flows, modules and external systems interact with each other. This is particularly important if the system is

embedded in a larger ecosystem later in the development process, as it makes the effects of changes on other components or third-party systems visible. Ultimately, reverse engineering has also provided a fundamental step towards not only developing CheckedInn 2.0 better in the future, but also ensuring ease of maintenance in particular.
With the requirements and the modeled, reverse engineered CheckedInn 1.0 system as a foundation, the usability of CheckedInn 1.0 was then examined. Discount usability testing according to Jakob Nielsen was chosen as the method for the test process. With a sample of 13 hotels, 4-5 hotel employees were interviewed in each of 5 iterations. The tests were carried out using a selection of tasks that were presented to the test subjects in the form of a usability study scenario. These comprised a full flow that covered the daily work routine in a hotel in the best possible way. From setting prices, creating a reservation and checking in a guest to creating invoices and checking out, an attempt was made to cover the full range of day-to-day business. The evaluation was then based on Nielsen's 10 Usability Heuristics (see section 2.2.1). The feedback was classified into 5 categories using an LLM in order to better categorize the feedback and ensure faster incorporation during further development. Usability issues and bug reports in particular were of high priority for the dynamic adjustments within the test process. The discount usability procedure ensured fast results with limited resources. The cost efficiency results from low financial and time investments and has made it possible to focus on the implementation of feedback. At the same time, no major losses in the quality of the feedback had to be accepted, as over 80 percent of the problems could be identified with just 5 test persons per iteration (see section 2.2.1). The iterative nature of discount usability testing has also contributed greatly to promoting an iterative adaptation process and thus obtaining early feedback on the adaptations of previous iterations. This enabled bugs and new usability issues to be identified quickly and increased the efficiency of system improvements. Finally, the genuine user perspective helped to gain a better understanding of the domain. The discount usability testing approach focuses on how real users use the system. By involving a small, representative group of end users, real usage scenarios were tested, which increased the relevance of the results. This ensured that the usability problems identified actually occur in the real context of use and should be addressed.

The following section shows the results of the usability testing in the form of observations within the quantitative data and at the same time addresses the limitations of the results. Possible future work approaches are presented in 7.2. Here, both scientific aspects that need to be investigated further and further development of the system are highlighted.

7.1 Summary of Findings

As part of the feedback-type based evaluation, there were a number of findings that can be concluded from the quantity and timing of different types of feedback.

Deployment issues always occurred during the first three hours after a new, major release. When problems occurred, hot-fixes[15] or rollbacks were carried out immediately to restore the stability of the system. Overall, the deployment issues were reduced to 0 in iterations 4 and 5. As a functioning deployment is the basis of a functioning system, the focus should therefore be on a functioning and stable deployment immediately before and after a new release. Monitoring tools such as New Relic or a deployment manager within a development team are recommended for this purpose. It is not possible to completely prevent deployment issues. However, the data shows how the risk can be reduced through increasingly extensive testing and raising awareness of critical points in the CI/CD pipeline.

The results of the analysis of bug reports show that the number of bug reports has been significantly reduced. While the first iteration still had 22 bug reports, in iterations 4 and 5 there were only 2 and 1 respectively. The data shows that the number of bug reports can be attributed to two main components: The experience in handling bug reports and the size of the release. A great deal of experience, especially not only of a technical nature but also in dealing with the use of the software by real users, sensitizes the developers to critical points. The realization that not only technical knowledge but also an understanding of the end-use processes is important underlines the value of domain-specific expertise. Developers who are familiar with the day-to-day processes of users can better anticipate potential vulnerabilities in the code and take preventative measures. As the size of the release increases, errors become more likely, which is why it is recommended to keep releases small. In this context, Continuous Integration (CI) practice becomes particularly important as it encourages regular, small updates and reduces the likelihood of larger releases causing unforeseen errors. Finally, it is noticeable that the majority of bug reports were also reported within the first few hours after release, which is why bug reports should be given great attention immediately after release, just like deployment issues.

Change requests have decreased over the course of the study. It is striking that there is only one change request in the third iteration as a result of a major release[16]. In the fourth iteration, however, the number increased again. Due to the large release in the third iteration, this suggests that the size of the release has an influence on the number of change requests. There is no doubt that the quality of the new changes also plays a role.

[15] Hot-fix is an engineer-term for quickly fixing a bug

[16] Meaning a release with a large amount of new features and adjustments

However, it can be concluded that users need time to get used to the system and that the change requests therefore remain low with a large number of changes, as users first want to familiarize themselves with new functionalities before they express change requests. However, this characteristic of the willingness to familiarize oneself with new functionality may also be due to the fact that CheckedInn is an operating software and users therefore inevitably have to deal with it. It might be interesting to investigate whether these values also apply to non-operational software. However, the drop in change requests could also be due to the developers' better understanding of how users use the system. Once again, we can see how important domain knowledge is. The realization that change requests decrease over the course of releases ultimately also shows that users become accustomed to using the software and therefore stop questioning the software. This makes it all the more relevant to constantly test with unbiased people who can view the software from a fresh perspective.

Feature proposals show a clear increase over the course of the study. This can be attributed to various factors. Users are gaining confidence in the system. This is due to familiarization, but in particular also to the increasing stability of the system, as evidenced by decreasing bug reports and deployment issues. It suggests that users are developing a deeper understanding of the system and are actively interested in improving it. Also, the longer the system is used, the more special cases arise that were both unknown to the developers at the beginning and not considered by the experts who initially set the requirements. The greater the depth of use, the more detailed the requirements and therefore the demand for new features. The aforementioned relevance of unbiased testers is thus extended by the relevance of experienced testers. It can be concluded that the heterogeneity of the test group is of the highest relevance for good quality test results.

The usability issues also decreased significantly over the course of the study. It is striking that they could not be reduced to 0 in the last iteration either, despite the strong focus of the study and the adaptation to usability. This underlines the fact that usability can be objectified to a certain extent, as Nielsen did, with regard to some factors. However, the final assessment of good and bad usability remains subjective to a certain extent. In general, however, the higher user acceptance suggests that usability has actually been improved. Another conclusion from the decreasing usability issues is the quality of the study. It shows that Nielsen's 10 usability heuristics are suitable for testing usability. However, the increasing stability of the system also seems to be a driving factor for a lower number of usability issues. These make the user experience appear more trouble-free right from the start. However, the values should be questioned insofar as it could be observed that users tend to shift their focus away from usability and towards possible new features as they become more stable and therefore more accustomed to the system.

The increasing satisfaction of the users with the usability is ultimately also manifested in the heuristic-based evaluation. Over the course of the study, usability was increased with regard to 8 heuristics. Once again, it is confirmed that the increasing values indicate that CheckedInn 2.0 has reached an advanced level of maturity over the course of the study. It exhibits fewer fundamental usability problems in later iterations and offers a more stable and robust user experience that also works well under different conditions of use. This is also potentially due to better adaptation to the needs and requirements of end users. In particular, the sharp increase in H1 Visibility of the System Status by 18.73% from 4.083 to 4.833 shows that the developers have gained a better understanding of actual usage over the course of the study. Each domain requires its own visibility and status indicators. The increase makes it clear that the close collaboration with end users has led to a better understanding of the domain. H7 shows no changes and has the same value in iteration 5 as in iteration 1. H4 Consistency and Standards, on the other hand, has deteriorated by 4.93% from 4.645 to 4.416. The deterioration of H4 is potentially due to the expansion of the system within the test period. The focus on stability fell at the expense of consistency and shows that CheckedInn 2.0 has deficits in this area. Finally, overall satisfaction with usability increased by 18.75% from 8 to 9.5.
The system resulting from the five iterations of usability tests is no longer CheckedInn 1.0. Instead, a new system emerged, which is referred to as CheckedInn 2.0. This system is more stable and of better quality than CheckedInn 1.0, especially in terms of usability.

The validity of the results is restricted by a number of limitations. It must be questioned to what extent the sample size was large enough and heterogeneous enough. As mentioned above, it is important to find a good balance between experienced and inexperienced testers. Experience can refer both to the software itself and to domain knowledge. It also remains to be seen whether five iterations are enough to achieve significant and meaningful results. Furthermore, quantifiability is difficult despite heuristics. As already described, user-friendliness can only be objectified to a certain extent.
Environmental influences must also be taken into account, with the Hawthorne effect playing a key role in this context. It is uncertain to what extent users actually demonstrated user behavior during the study. However, the feedback form in the software proved to be helpful in mitigating this effect. Finally, users exhibit different learning curves, which are therefore also reflected in the intuitiveness of the software and thus in the user-friendliness.

It is important to note that the use of operating software may behave differently to other software. For this reason, there is also a limit to the extent to which the results can be generalized.

7.2 Future Work

The following section deals with aspects that might be interesting to further investigate and implement in future work. The first paragraph refers to recommendations for possible extensions to the usability study and the handling of results. The second section deals with possible approaches for the further development of CheckedInn 3.0.

The most obvious approach to expansion is a more considered choice of sample. Particular attention should be paid to the aforementioned sample heterogeneity, even within iterations. In order to better recognize differences in perception between experienced and less experienced users, it is advisable to label the data obtained and work out differences. Finally, it would be useful for studies to work with a wider variety of hotel types. The hotels surveyed in this study are all a very similar type of hotel, which also results in similar requirements. Significantly larger or smaller hotels or hotels with different accommodation concepts could lead to a greater variance in feedback.
A larger number of iterations could be used to check the extent to which the trends from Section 6.1 continue to be reflected and the assumed long-term convergences actually apply. This would also improve the generalizability of the results.
Furthermore, it would be useful for further work to expand the test scenarios to include sudden changes for the test subject. In this way, it can be achieved that the user has to break out of their usual flow of activities. Inexperienced users in particular who are confronted with such situations can provide valuable feedback on user-friendliness.
The more intensive use of an LLM is recommended for data evaluation. It would be possible to extend the classification made in this study in the form of clusters and prioritization. In the context of clustering, redundant feedback could be eliminated and at the same time the quantity of a feedback category, the time of the feedback or the content could be used to draw conclusions about its importance.

CheckedInn 2.0 offers a wide range of expansion options. Utilizing the usability-focused approach presented in this thesis to further engineer the system could result in an even more powerful tool - CheckedInn 3.0. In addition to a stronger integration of interfaces to third-party providers, the expansion of functionalities for large hotels is particularly useful.

Optimized solutions for hotel-owned restaurants or improved digital key systems are logical concepts.

Increased use of LLMs would also be conceivable for the system. This could automate guest correspondence or accounting processes. A virtual 24/7 reception desk in form of an avatar for guests would be particularly innovative in the hotel industry. An LLM could be used to answer questions about how to get to the hotel, the hotel stay, the payment process or recommendations for restaurants and places of interest. In order to increase user-friendliness for the guest, this technology could take the form of an application rather than a website. With rising prominence of speech-controlled interfaces, a voice-assistent for the CheckedInn Hotel App could improve the system and make it even more accessible for unexperienced employees. The greater integration of guests into the hotel's IT infrastructure offers a future-proof concept for improving and personalizing the guest experience. For this, a possible aspect is a digital centralized document management tool in the CheckedInn Gust App. This would allow guests to directly manage all their bills and other relevant documents for past, current and future trips. Additionally, many hotels offer more assets than simple accommodation, like for example spa areas, parking lots or electric car charging stations. Currently, those are primarily managed by the receptionist. Offering the guest the option to book these via the CheckedInn Guest App could further improve the guest experience. Instead of having to call the reception and inform them that they need a charging station for their electric car, guests could simply enter their estimated arrival time and reserve a charging station when they arrive. Finally, sustainability concepts are conceivable in which hotel purchases are optimized or leftover food is recycled in the hotel restaurant.

Appendix

A Questionnaire

CheckedInn - Usability Study

Usability Study Scenario
Participant

1. Set the price for bed + breakfast in a double room for the next weekend at € 130 per night.
2. Create a reservation with the rate bed + breakfast in a double room in the name of Max Mustermann for Thursday to Sunday next weekend. The guest's email is max@mustermann.de.
3. On the night from Friday to Saturday, the Max would like to stay in a single room.
4. Change the address of Max and add a billing address.
5. Max would like to check in online. Prepare the online check-in. The room code is 3852 and the check-in email should be sent to the guest's email address.
6. Max has the status VIP. Let housekeeping know that they should provide the guest with a fruit basket and champagne in the room.
7. On the day of arrival, the Max does not check in online, but goes to reception. Create a registration form in the guest's name.
8. Check in Max.
9. Max makes a deposit of 100€ by Visa.
10. Max has arrived by car and parks in the hotel parking lot. Book this as an extra service for the duration of the entire stay.
11. On Friday morning, the Max has a visitor in the morning and would like to book an additional breakfast.
12. Max does not want breakfast on Saturday. Change the rate for Saturday to overnight stay without breakfast.
13. On Friday evening, the Max reports that the shower is faulty. Move him to a new single room for the night.
14. Communicate the facility manager to repair the shower.
15. Max would like to leave earlier. Cancel the last night of the stay as well as all extra services for this day.
16. Max has a Tesla and would like to charge it at the hotel's own charging station. Add this as an extra service.
17. On Saturday morning, Max would like to pay the remaining amount on account.
18. Create an invoice for the reservation of Max.
19. Max asks you to send him the invoice by email.
20. Check out Max.
21. Housekeeping will let you know on Saturday lunchtime that the room has been cleaned. Set the status of the room to Clean.
22. It will take a few days to repair the damaged shower. Lock the room until Wednesday.
23. Generate an arrival and departure list for Sunday.
24. Generate a catering list for Sunday.
25. Ask in the hotel messenger if you can arrive at work half an hour later tomorrow.

CheckedInn - Usability Study

Questionnaire 1

Participant

1. Visibility of the System Status

	Strongly disagree				Strongly agree
The system displays immediate confirmation after booking a room	1	2	3	4	5
Status updates are visible when changing the booking or service details	1	2	3	4	5
The system provides a visible log or history of all actions taken for a reservation	1	2	3	4	5

Issues & Recommendations

2. Match between System and the Real World

	Strongly disagree				Strongly agree
The language used in the system mirrors the language used by hotel staff and guests	1	2	3	4	5
The system uses terms familiar to both new and experienced hotel employees	1	2	3	4	5
Icons and symbols are intuitive and related to real-world hotel operations	1	2	3	4	5
Information is presented in a natural and logical order	1	2	3	4	5

Issues & Recommendations

3. User Control

	Strongly disagree				Strongly agree
There is a clear option to undo changes made to a guest's booking	1	2	3	4	5
Users can easily modify or delete entries without having to navigate through multiple screens	1	2	3	4	5
Users have the flexibility to switch between different tasks without having to complete one before starting another	1	2	3	4	5
The system provides shortcuts that enable experienced users to perform tasks more rapidly	1	2	3	4	5

CheckedInn - Usability Study

Questionnaire 2
Participant

Issues & Recommendations

4. Consistency

	Strongly disagree				Strongly agree
UI elements behave uniformly across different sections of the system	1	2	3	4	5
Standard terminology is used throughout all user interactions	1	2	3	4	5
Consistent visual elements are used for error messages, alerts, and notifications	1	2	3	4	5

Issues & Recommendations

5. Error Prevention

	Strongly disagree				Strongly agree
Warnings are issued after conflicting information have been entered	1	2	3	4	5
The system validates data entry to prevent common input errors	1	2	3	4	5
Confirmation prompts appear before finalizing potentially important actions	1	2	3	4	5
The system checks for double bookings and availabilities before confirming a reservation	1	2	3	4	5

Issues & Recommendations

CheckedInn - Usability Study

Questionnaire 3

Participant

6. Recognition Rather Than Recall

	Strongly disagree				Strongly agree
Interface elements are logically organized	1	2	3	4	5
Dropdown menus and auto-fill features reduce the need for memory use	1	2	3	4	5
Frequently used tools are prominently placed on the dashboard	1	2	3	4	5
Different choices are made clear	1	2	3	4	5

Issues & Recommendations

7. Flexibility

	Strongly disagree				Strongly agree
Users can configure the dashboard to display information relevant to their role	1	2	3	4	5
Bulk actions are possible, such as checking out multiple guests at once	1	2	3	4	5
The system supports multi-tasking, allowing users to open multiple tabs or windows simultaneously without performance degradation	1	2	3	4	5

Issues & Recommendations

8. Aesthetic

	Strongly disagree				Strongly agree
The interface is clutter-free, focusing only on necessary elements	1	2	3	4	5
The user interface does not distract from solving tasks	1	2	3	4	5
The look and feal optimizes space and prioritizes task-related information	1	2	3	4	5
Aesthetic elements do not interfere with the functionality of the system	1	2	3	4	5

CheckedInn - Usability Study

Questionnaire 4
Participant

Issues & Recommendations

9. Error Handling

	Strongly disagree				Strongly agree
The system clearly indicates when an error occurs, using both visual cues (like color changes) and textual descriptions	1	2	3	4	5
Users are guided through the steps to recover from common errors	1	2	3	4	5
The system includes proactive error prevention mechanisms, such as validation checks	1	2	3	4	5
The system maintains a consistent approach to error handling across all modules	1	2	3	4	5

Issues & Recommendations

10. User Guidance

	Strongly disagree				Strongly agree
Step-by-step guides are provided for complex procedures	1	2	3	4	5
The system's help documentation is available within the application	1	2	3	4	5
Complex terms are explained in detail in the application	1	2	3	4	5

Issues & Recommendations

CheckedInn - Usability Study

Questionnaire 5
Participant

Overall Feedback

The software provides a good usability

Strongly disagree									Strongly agree
○	○	○	○	○	○	○	○	○	○
1	2	3	4	5	6	7	8	9	10

Further remarks *(positive, negative, features requests)*

CheckedInn - Usability Study

Think Aloud Notes 1

Facilitator

Task 1

Task 2

Task 3

Task 4

Task 5

Task 6

Task 7

Task 8

Task 9

CheckedInn - Usability Study

Think Aloud Notes 2

Facilitator

Task 10

Task 11

Task 12

Task 13

Task 14

Task 15

Task 16

Task 17

Task 18

CheckedInn - Usability Study

Think Aloud Notes 3

Facilitator

Task 19

Task 20

Task 21

Task 22

Task 23

Task 24

Task 25

B Models

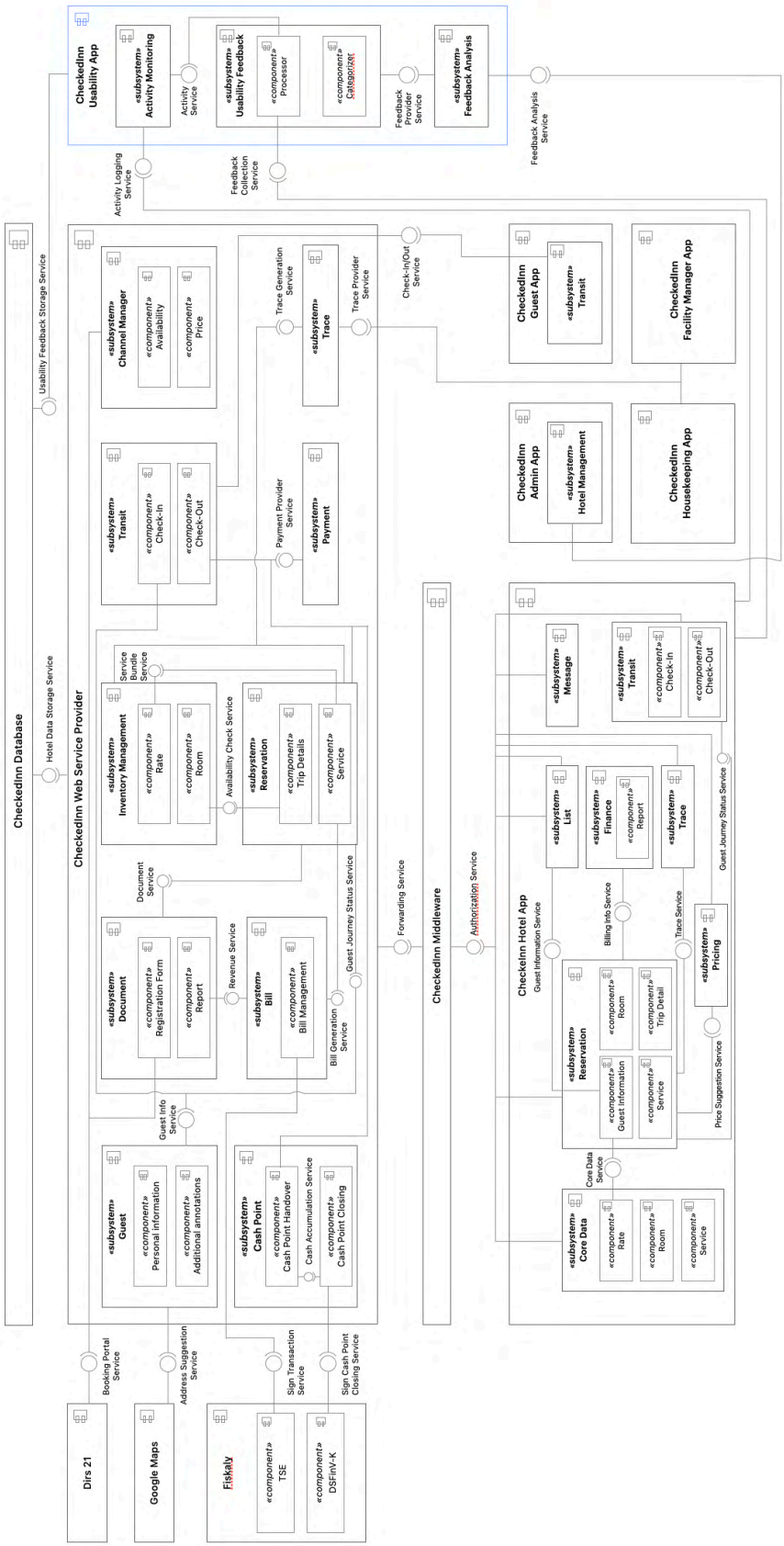


Figure B.1: CheckedInn 1.0 (UML Composition Diagram)

C Exemplary handling of study feedback (KN)

C1 Exemplary handling of a deployment problem

During the deployment of the first release there were problems with the services provided by the CheckedInn Web Service Provider. Any services provided by one of its subsystems could not be received by any of the consuming apps. This resulted in faulty functionality in the CheckedInn Hotel, Guest, Housekeeping and Facility Manager App. In order to investigate the problem, it was examined whether the requests are sent correctly from the respective apps to the CheckedInn Web Service Provider, whether they arrive correctly at the provider, whether they are processed correctly and finally whether the correct response is send to the respective apps. It was verified that the communication from the respective app works without errors. This meant that the apps could be ruled out as a source of error and the focus could be placed on the CheckedInn Web Service Provider. Upon further investigation of the problem, it was noticed that all requests from the respective apps were not forwarded to the CheckedInn Web Service Provider. As the apps were working, it was assumed that the problem must lie within the middleware, as this is the only component between the respective apps and the CheckedInn Web Service Provider. In fact, it was found that all service requests to the CheckedInn Web Service Provider were blocked by the middleware in the deployed version of the system. As the middleware initially provides the Authorization Service to the respective apps and the problem did not occur during the development of the system, it was reasonable to assume that the problem originated in the Authorization Service. In fact, the error laid in the fact that the CORS configuration for the CheckedInn Web Service Provider only permitted the local development environment. The origin of requests is in principle important in order to prevent unauthorized requests to the CheckedInn Web Service Provider. However, it was necessary to configure the deployed version of the system as the legitimate origin and thus enable the consumption of services. By adapting the configuration, the providing of services to the respective apps of the deployed system was permitted. Figure C.1 shows the adapted Authorization Service in

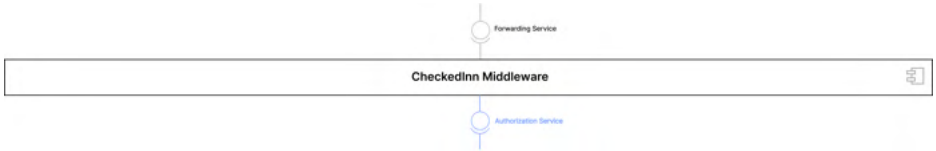


Figure C.1: Patched Authorization Service in the CheckedInn 2.0 Middleware (UML Composition Diagram)

connection with the CheckedInn Middleware. As a result, the requests arrived correctly at the CheckedInn Web Service Provider, but they were still not processed correctly.

Therefore, the provided services to the respective apps were still incorrect. After further investigation it became evident that the services within the CheckedInn Web Service Provider were working correctly, but requests to the CheckedInn database were being rejected. Here too, the cause was that the configuration of the database restricted the origin of requests and only allowed the development environment. By adjusting the configuration, the deployed version of the system was also permitted as the origin. As a result, the services were correctly provided by the CheckedInn database, correctly processed within the CheckedInn Web Service Provider and finally provided to the respective apps.

C2 Exemplary handling of a bug report

Bug description:

The first bug report resulted from the feedback form and was therefore discovered and recorded by a receptionist when using the system outside of the assisted execution of usability studies using the questionnaires. In concrete terms, BR1 is about the fact that the status of a room is not automatically set to dirty after a guest has checked in. For this to happen, the room must be displayed in the CheckedInn Housekeeping App before the guest checks out. This is the only way to ensure that the room is clean again when the next guest checks in. Although the room could also be manually assigned to a housekeeping employee by the receptionist, the room will not be displayed to the receptionist either if its status has not been updated to dirty. BR1 therefore affects fundamental use cases of the reservation and trace scenario and the associated functional and non-functional requirements. Table C.1 shows the use cases of the reservation and trace scenario affected by the bug report, as well as the associated requirements.

Requirement	Reservation Scenario	Trace Scenario
FR 1	`Check-In`	
FR 7		`Assign Trace`
FR 8		`Get compressed information for daily business`
FR 9		`Generate and export lists for daily needs`
NFR 9	`Check-In`	
NFR 10		`Display all traces`

Requirement	Reservation Scenario	Trace Scenario
NFR 11	`Check room availability`	`Get compressed information for daily business`

Table C.1: Use cases and requirements affected by BR1

The use case diagrams of the two scenarios show that the actors receptionist, guest and housekeeping employee are affected by this bug report. Fixing BR1 is therefore a high priority.

Bug analysis:

In order to fix the bug, the participating subsystems of the reservation and trace scenario were analyzed (composition diagram in figure C2). In particular, the CheckedInn Web Service Provider, the CheckedInn Hotel App and the CheckedInn Guest App were investigated. It was noticed that the bug occurs regardless of whether the guest is checked in by the receptionist or checks in themselves online. This led to the assumption that the problem did not originate in the CheckedInn Hotel App or the CheckedInn Guest App. Therefore, the error probably lies in the CheckedInn Web Service Provider. In this subsystem, only the Inventory Management and Transit components influence the status of a room. As Inventory Management is responsible for transmitting the new status to the CheckedInn Hotel and CheckedInn Guest App using the Room Status Update Service, it was then checked whether this was providing the correct data. It was noticed that the service supplied incorrect data, but also did not receive any updates regarding the status of the room after a guest checked in. This made it clear that the transit subsystem in the CheckedInn WebService Provider does not send any updates at all after a successful check-in. This of course means that the status of a room is not updated correctly and explains BR1.

Bug solution:

As the Guest Journey Status Service and Check-In/Out Service were working correctly, it was clear that the transit subsystem itself is working, and only one service is missing, which provides the status updates of the rooms to the room component in the inventory management subsystem. Therefore, the Room Status Update Service is introduced, which is provided by the check-in component in the transit subsystem to the room component in the inventory management subsystem. Subsequent testing has shown that the status of a room is now set to dirty after a successful check-in, as required. The bug from BR1 is therefore fixed. Figure C.2 shows the section of the updated components in the CheckedInn Web Service Provider as a composition diagram.

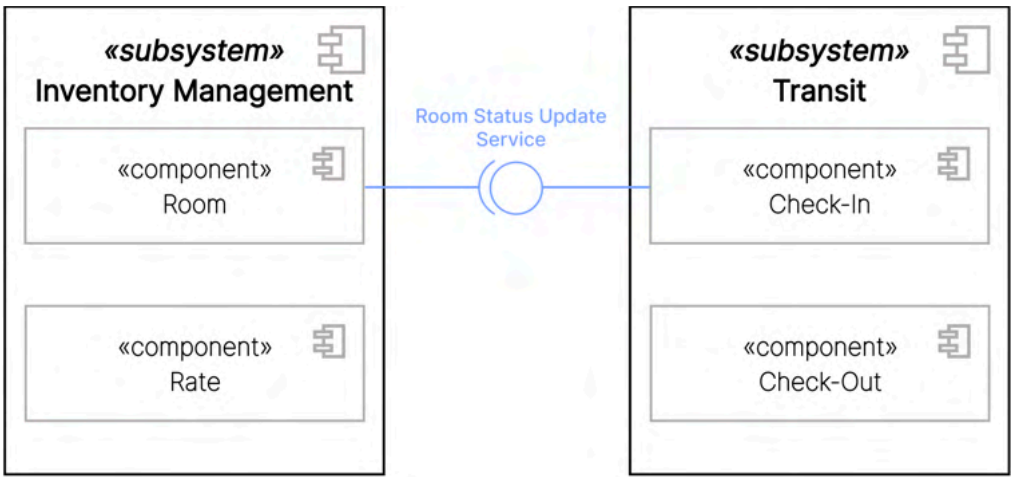


Figure C.2: Subsystems in the reservation scenario for BR 1 of CheckedInn 2.0 (UML Composition Diagram)

C3 Exemplary handling of a change request (SL)

In the second iteration of usability testing, a live feedback from a receptionist that had a significant impact on the further development of the software was received. The receptionist pointed out that although the current occupancy plan allows entire reservations to be moved to other rooms using drag-and-drop, moving partial reservations is not supported. This functional gap leads to considerable additional work and restrictions in the handling of the system.

To move only part of a reservation, users currently have to open the reservation and select the start date for the move via the detailed view under the “Rooms” tab. A further click then displays all available rooms, whereupon the selection of a room enables the partial shift. This process requires nine clicks from the occupancy plan and only allows postponements that start on a specific day and last until the end of the reservation. It is not possible to adjust the time period within the reservation more precisely. So if one only wants to move a reservation for a certain period within the booking, one has to take several cumbersome steps, which can take up to twenty clicks in total. The receptionist therefore suggested a more user-friendly solution that would allow partial reservations to be moved more quickly and directly from the occupancy plan. His idea was to reduce this process to no more than three clicks. A concrete solution concept was not initially available, but various options were discussed, prototypes created and debated in a subsequent detailed meeting. Ultimately, a drag-and-drop solution was agreed upon, based on the existing functionality for moving entire reservations. A solution was implemented for selecting the days to be moved, in which the specific days can be selected by right-clicking on the relevant

reservation in the occupancy plan. The days then appear as individual boxes, which can be selected by simply clicking on them and moved to another room using drag-and-drop. This optimized process now makes it possible to significantly reduce the number of clicks to “2 + number of days”. For a three-day reservation, where only the middle day is moved, this saves 15 clicks compared to the old method. The process of day selection and the status after such an operation can be seen in figures C.3, C.4 and C.5.

To ensure that partial reservations are only moved spatially (by room) and not temporally (horizontally), the user receives an error message if an attempt is made to move the reservation horizontally. This new solution positively influences a number of usability heuristics and not only improves usability, but also fulfills several non-functional requirements, making the system even better suited to the original requirements.

In particular, heuristic H7 `Flexibility and Efficiency of Use` is positively influenced by the reduction in clicks and the fact that the occupancy plan, which often serves as the default page, does not have to be left. This significantly increases both the flexibility and the efficiency of the system when executing the underlying actions. NFR 3 `Performance` and NFR 4 `Scalability` also benefit, as network traffic is saved by eliminating the unnecessary loading of the reservation page.

After the change was implemented, it was discussed again with the receptionist who had provided the original feedback and finally confirmed by him. This iterative feedback loop is critical to the continuous improvement of the system and ensures that the PMS effectively meets the needs of its users while providing a high level of usability and system efficiency.

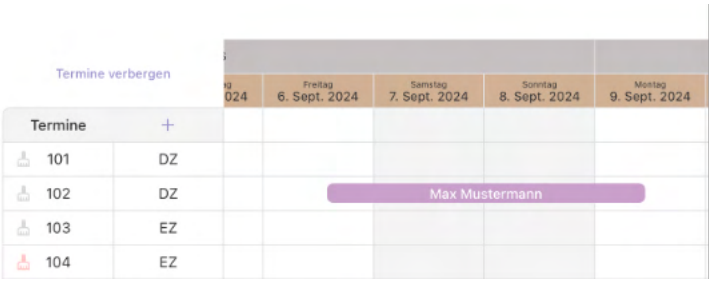


Figure C.3: Reservation on occupancy plan (CheckedInn 2.0)

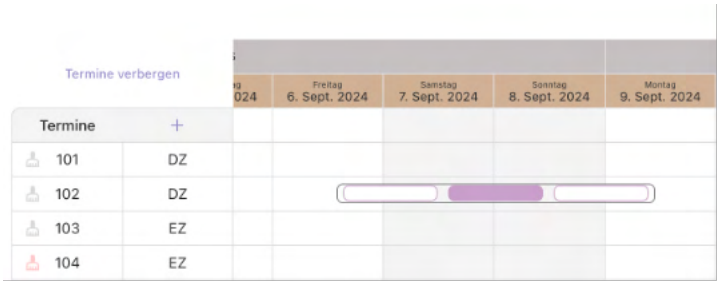


Figure C.4: Reservation after right click with the second day selected (CheckedInn 2.0)

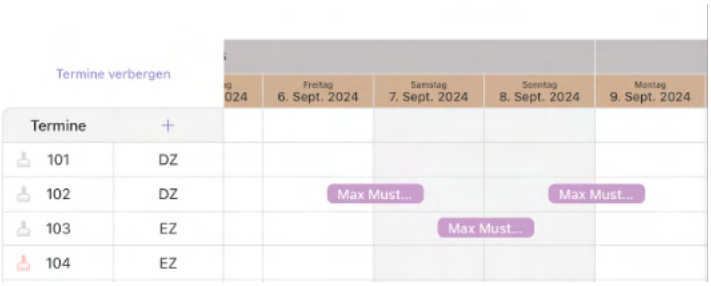


Figure C.5: Reservation after dragging the second day to room 103 (CheckedInn 2.0)

C4 Exemplary handling of a feature proposal (SL)

As part of the first iteration of the usability test, a specific issue was identified in the pricing module for the channel manager by a receptionist. The feedback provided relates to the current limitation of the system to only be able to set prices for specific combinations of day, rate and category. Currently, it is only possible to make price changes either for a specific day or over a defined period of time, with these adjustments always being limited to the specific rate-category combination.

The scenario described by the receptionist illustrates that this restriction leads to difficulties in practice: There are situations where a general price adjustment is required for all categories on a specific day. A typical example of this would be a high occupancy rate for the following day, which indicates increased demand. In such a case, it would make strategic sense to increase the prices for all room categories on that day across the board, either by a fixed amount or as a percentage. Similarly, below-average demand may make it necessary to reduce prices for all categories in order to improve the booking situation. However, the current functionality of the system does not allow such comprehensive price adjustments to be carried out efficiently for a single day; each category must be processed separately, which leads to increased time expenditure and unnecessarily complicates the process flows. This feedback resulted in a feature request to implement a functionality that would allow prices for a specific day to be adjusted either absolutely or relatively for all categories simultaneously. No concrete solution was proposed, but this opens up the possibility of developing creative and efficient approaches to further increase the user-friendliness and efficiency of the system.

The proposed feature request to adjust prices at the level of a daily rate combination for all categories is of crucial importance for the efficiency and effectiveness of the system. This relevance arises both from the operational importance in day-to-day business and from the

requirements that the original system must fulfill. Pricing related functionality is of considerable importance in the context of day-to-day operations. Nowadays, the majority of hotel reservations are processed via online booking portals, either commercial ones or the hotel's own website. Efficient and simple pricing is therefore essential, as it needs to be adjusted regularly to respond to market changes such as increased or decreased demand. The ability to adjust prices for all categories on a given day simultaneously and either in absolute or percentage terms would significantly reduce the workload and increase the user-friendliness of the system. Especially for finance related functionalities, stable and intuitive operation is of utmost importance to avoid errors (H5) and ensure smooth operations.

This feature request also has far-reaching implications for several requirements of the original system. A central functional requirement here is FR12 `Synchronize with external booking platforms`. The implementation of this function ensures that price changes are immediately synchronized with the external booking platforms. The channel manager automatically transfers the changes to the connected portals, enabling seamless synchronization and ensuring that up-to-date prices are always displayed on the booking platforms. This not only improves the accuracy of pricing information, but also minimizes the risk of booking errors that could result from outdated pricing data. The NFR7 `Interoperability` is also directly fulfilled by this feature request. The system's direct integration with the channel manager as a third party system highlights its interoperability. This integration ensures smooth communication between platforms. It also facilitates seamless data transfer across different systems. Another requirement covered by this feature request is NFR4 `Scalability`. The ability to edit prices for daily rate combinations, independent of the categories, contributes significantly to the scalability of the system. Less manual effort makes the system much more scalable, especially for large hotels or chains with a large number of categories and rates. In these environments, prices can change frequently, so the feature also contributes to efficiency and allows the system to keep pace with the growth of the business without compromising on system performance or the user interface. The feature also increases the `Flexibility` (NFR9) of the system. The ability to adjust prices on a percentage or absolute basis for a day rate combination provides users with significantly greater flexibility. This enables a quick response to market changes and specific operational requirements without having to adjust each category individually. The enhanced control over pricing is particularly beneficial in dynamic market environments where quick adjustments are necessary to ensure competitiveness and achieve optimal utilization rates.

As part of the scenario-related analysis, it can be seen that this feature requirement relates in particular to the Booking Manager scenario. Specifically, the requirement relates to the

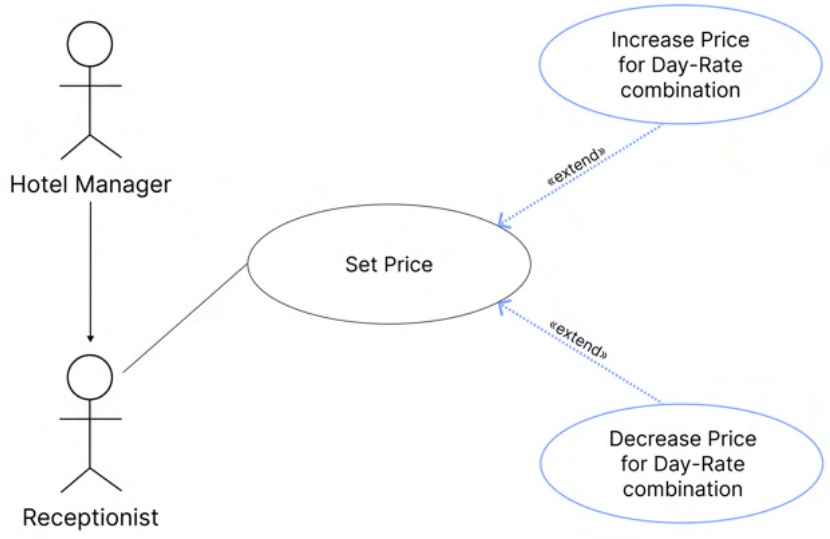


Figure C.6: Section of the new Booking Manager scenario for CheckedInn 2.0 (UML Use Case Diagram)

Set Prices use case, whereby the term set covers both the initial setting of prices and their subsequent editing. Figure C.6 illustrates a section of the use case diagram extended by the feature proposal, which visually depicts the new functionality. In the extended version of the use case, both the hotel manager and the receptionist still have the option of setting prices. However, this functionality is enhanced with the option to not only set prices for individual categories, but also to increase or decrease existing prices for all categories on a given day in relation to a set rate. These new use cases are shown in the diagram as Increase Price for Day-Rate Combination and Decrease Price for Day-Rate Combination. The option to use this functionality is illustrated by the use of the extends mechanism, which emphasises the optional nature of this new functionality.

As no concrete proposal for the design implementation was given in the original feedback, the initial approach was to implement a modal for the price change. This modal should be accessible via a menu that can be opened for each day. However, after detailed consultation with the feedback provider, the idea of using a single *edit prices* menu item was discarded. The reason for this was the high procedural relevance of price changes. Instead, it was suggested to offer two separate menu items for *increase prices* and *reduce prices* instead of choosing between these two options in the modal.

Figure C.7 shows the newly implemented menu. This menu already contained the function to set the availability of a day to zero, which was represented by the menu item *close day*. The menu can be accessed by clicking on the date of the day the user wants to edit. In the extended version of the menu, two additional, separately listed menu items are now integrated: Increase price and decrease price. Selecting the corresponding menu item takes the user to the modal shown in Figure C.8. This modal initially provides an overview of

the rate currently being viewed and the day in question. In addition, as requested in the feedback, there is the option to choose between an absolute and a percentage price change. Finally, the desired amount or percentage can be entered and confirmed by simply pressing a button.

Before the final release of this feature, it was tested in various scenarios to ensure that it met the requirements. After these tests, the feature was confirmed and released in a demonstration by the person providing feedback. This iterative development emphasises the importance of user feedback in the design process and ensures that the final product meets the practical requirements.

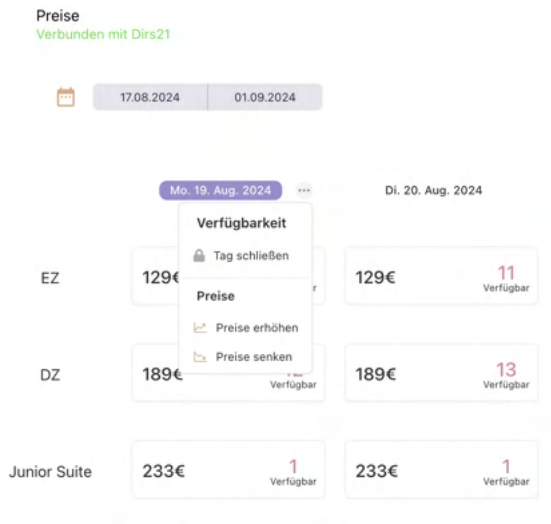


Figure C.7: Menu for opening price adjustment modals (CheckedInn 2.0)

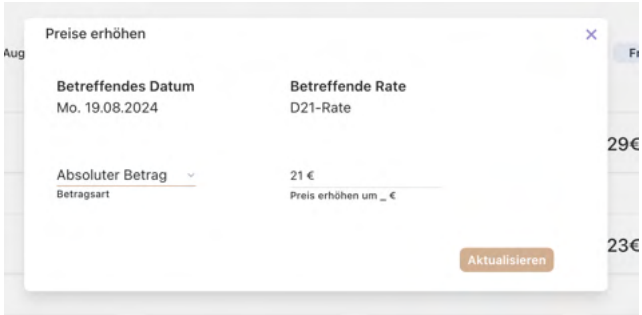


Figure C.8: Modal for price increase (CheckedInn 2.0)

C5 Exemplary handling of a usability issue (KN)

During the first iteration of the usability studies, a particularly interesting usability issue arose. This concerns the attachment of bills to emails sent to the guest. So far, documents related to a reservation can be accessed directly from the reservation, as shown in Figure C.9. However, in order to send the invoice to the guest as an email, for example, the invoice must first be downloaded as a PDF and then uploaded as an attachment from the hard drive, the button circled in green in Figure C.10. This was reported as a usability issue, as it is time-consuming for the receptionist to send the guest an invoice for their reservation by email.

The fact that this is a particularly relevant usability issue becomes clear from the requirements that are affected by it. First of all, it affects FR 4 `Guest Correspondence`, as additional work for the receptionist can result in low efficiency and thus possibly poor

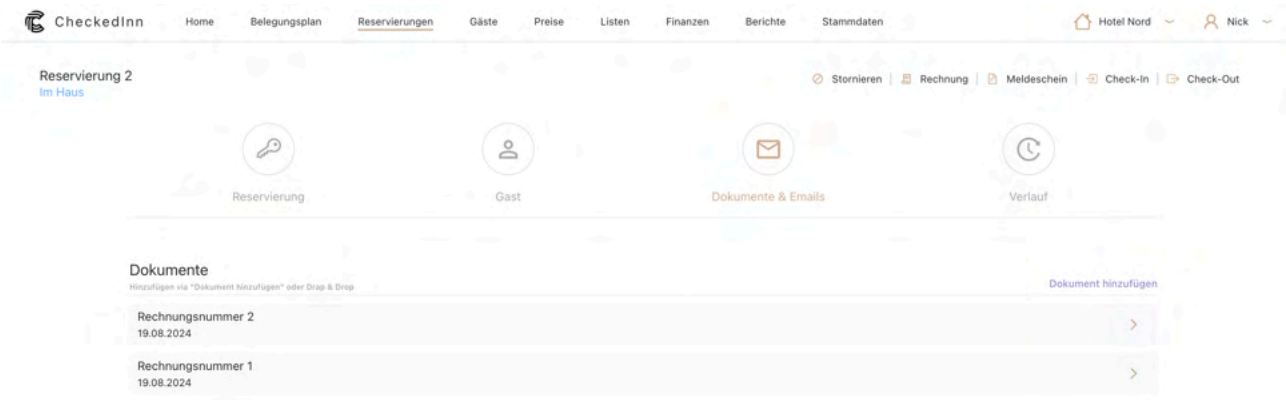


Figure C.9: Register of documents linked to a reservation (CheckedInn 2.0)

quality of guest correspondence. As the sending of bills by email is particularly important for guest correspondence, FR 3 `Create and manage bills` is also affected. With regard to the non-functional requirements, NFR 4 `Scalability` in particular suffers from this usability issue, as it is more difficult to send a large number of guests their bill by email.

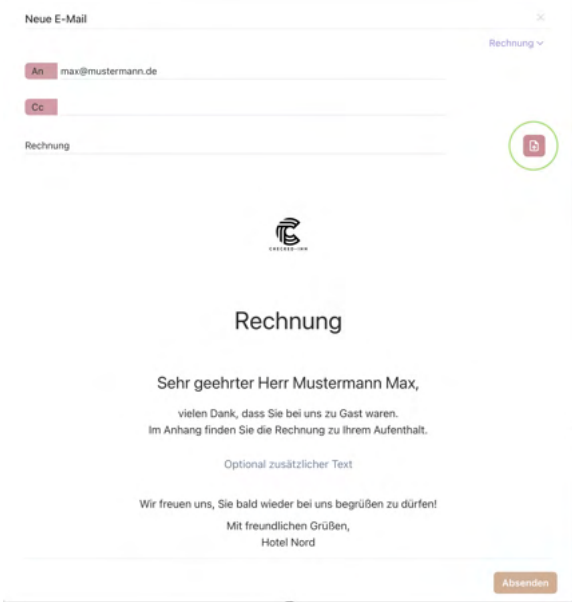


Figure C.10: Modal for sending an email (CheckedInn 2.0)

NFR 9 `Flexibility` is also affected, as the frequent need to send a document on an ad-hoc basis is difficult. In addition to these, however, the newly introduced NFR 12 `Usability` is particularly affected. As this has a particular focus in the Forward Engineering chapter, this issue has a high priority.

Since the sending of bills is particularly relevant, the scenario primarily affected is the Financial Manager scenario. There, the use cases `Export Bill` of the receptionist and the hotel manager, as well as `Access Bill` of the guest benefit from improved usability. In

addition to the purely informative purpose, sending invoices as emails also serves the basic purpose of settling outstanding payments. If guests receive their bills faster and with greater reliability, this makes it easier for them to settle outstanding amounts. This also reduces the need to inform the guest of an outstanding payment and request them to pay. Therefore, the use cases `Settle outstanding payments` of the guest and `Reminder about outstanding Payments` of the receptionist can also be indirectly facilitated in their execution.

To resolve the usability issue, the bill export component in the bill subsystem of the CheckedInn Hotel App was revised. The edited components can be seen in Figure C.11.

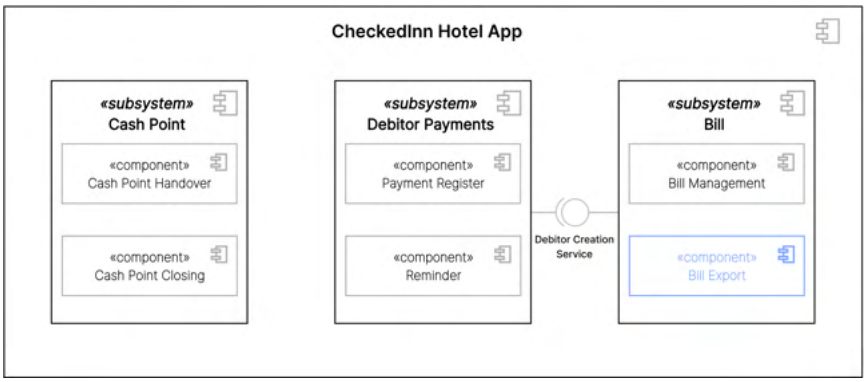


Figure C.11: Edited component visually highlighted in the CheckedInn 2.0 Hotel App (UML Composition Diagram)

In particular, Nielsen's heuristics H6 `Recognition rather than recall` and H7 `Flexibility and Efficiency of Use` were taken into account. As receptionists currently have to download the bill first, they have to memorize both the number of the bill and the path where the bill is saved on their device in order to subsequently upload it as an attachment to an email. Especially since receptionists are often not very tech-proficient, this can lead to difficulties when sending a bill by email. H7's statement that users should be allowed to tailor frequent actions is strongly related to this. The particularly frequent case of sending an invoice should be facilitated with shortcuts. Based on these two heuristics, a solution was found that not only provides a shortcut, but also eliminates the need to remember the invoice number and storage location on the local device. The idea behind the solution is to display all documents associated with a reservation directly in a list when a document is to be attached to an e-mail. The respective document is then added as an attachment with a simple click. This not only saves the receptionist from having to memorize the storage location and invoice number, but also reduces the effort involved in sending invoices by email, as they no longer have to go through the additional steps of first downloading an invoice and then uploading it again as an attachment. H6 was therefore taken into account and H7 was also included in the solution, as it is now possible to tailor

this frequent action to a specific reservation and automate the process. Thanks to the simple design, the receptionist does not even need to be an expert user. This also makes the work easier for novice users. The revised component is shown in Figure C.12. If documents are linked to the current reservation, they are displayed in the list below by clicking on the upload button (the button circled in green in Figure C.10).

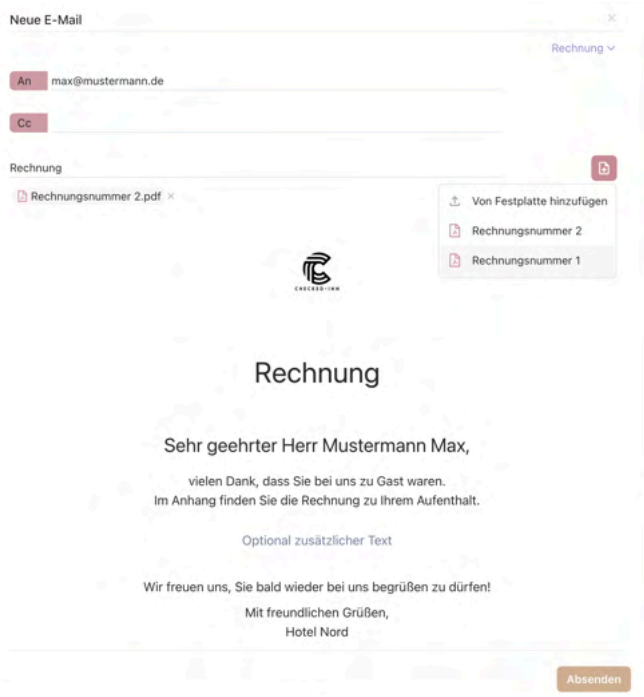


Figure C.12: Adjusted modal for sending an email (CheckedInn 2.0)

D Heuristic Evaluation

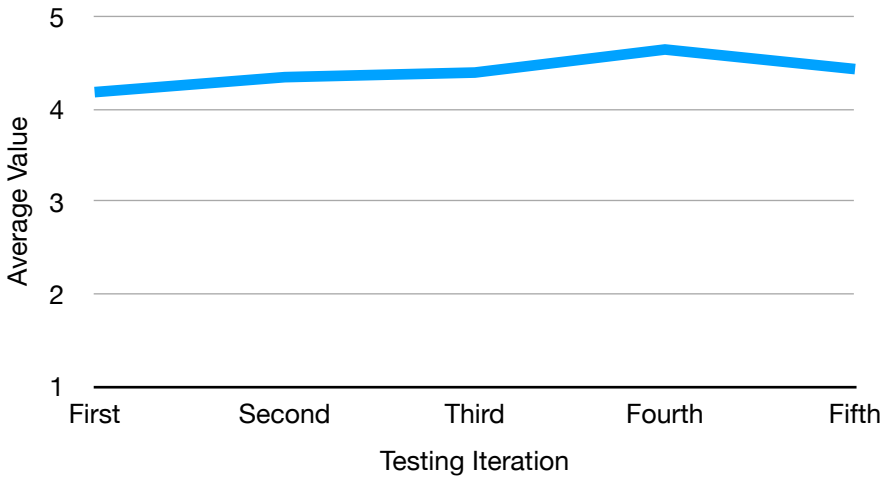


Figure D.1: Development of the average valuation of H3

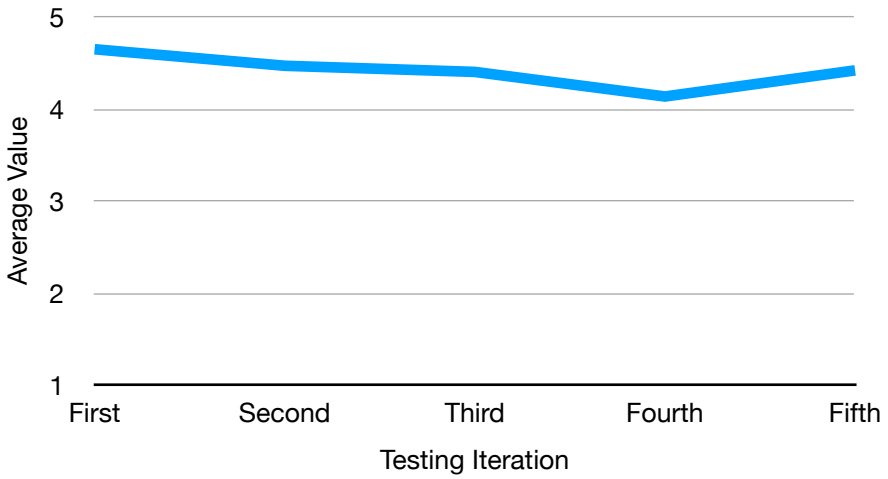


Figure D.2: Development of the average valuation of H4

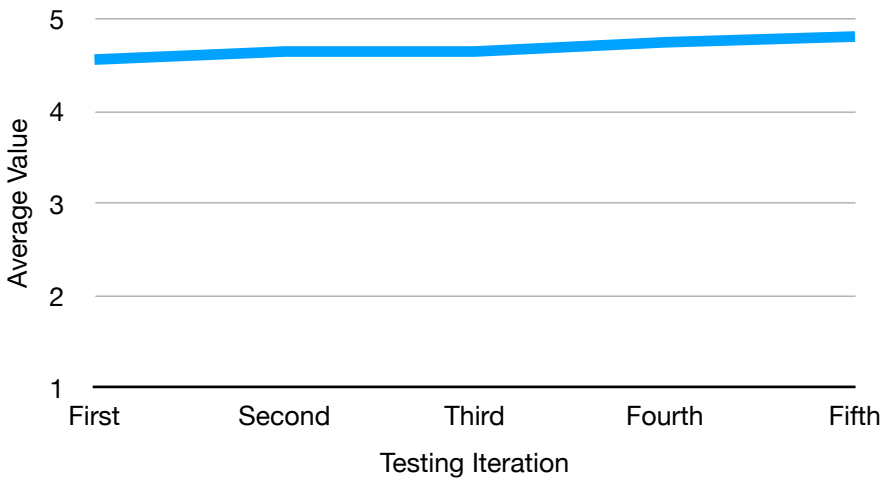


Figure D.3: Development of the average valuation of H6

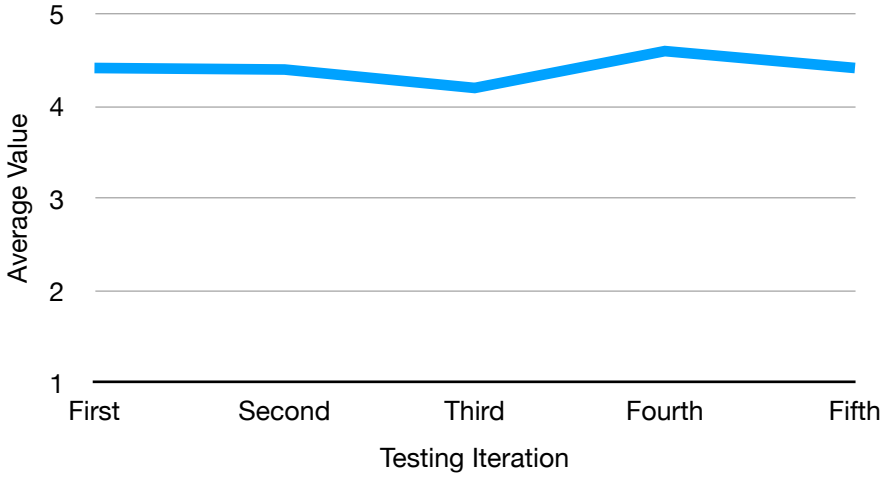


Figure D.4: Development of the average valuation of H7

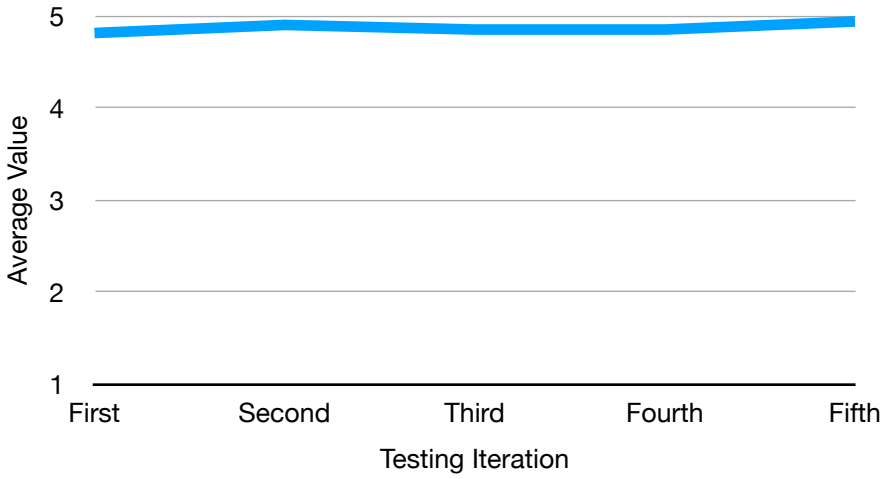


Figure D.5: Development of the average valuation of H8

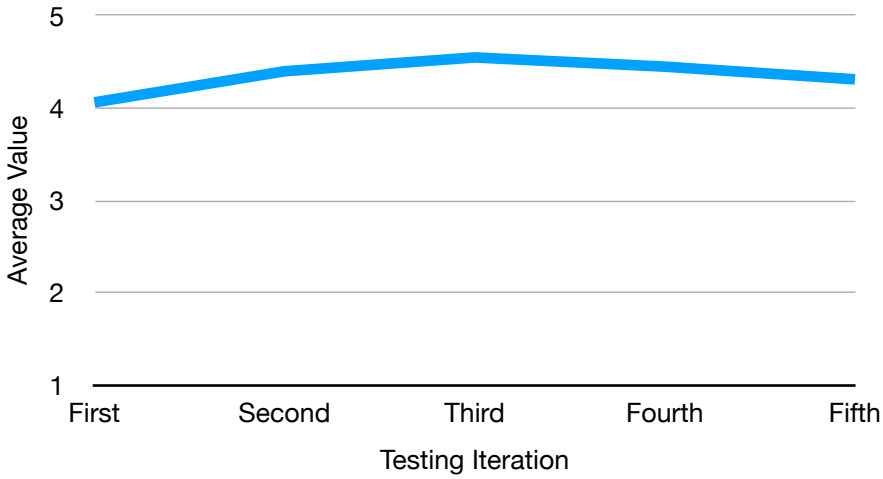


Figure D.6: Development of the average valuation of H9

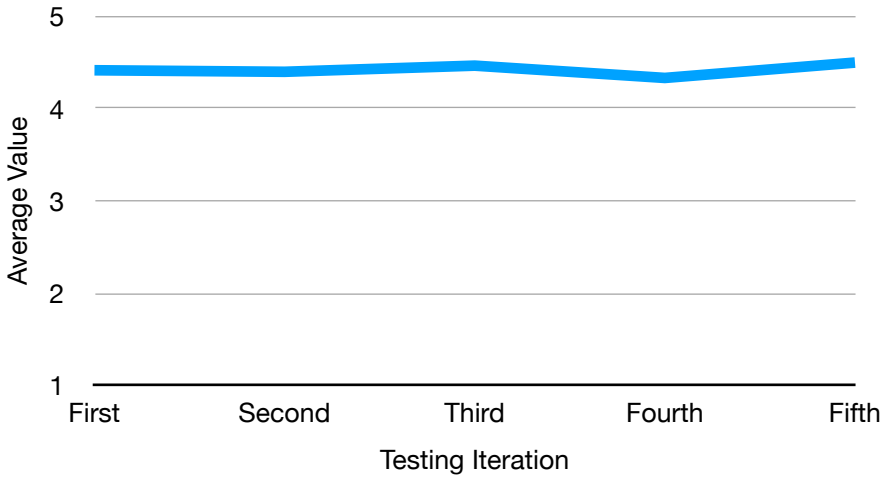


Figure D.7: Development of the average valuation of H10

Bibliography

[1]: Gretzel, U., Sigala, M., Xiang, Z., & Koo, C. (2015). Smart tourism: Foundations and developments. Electronic Markets, 25(3), 179-188.

[2]: Baum, T., Mooney, S. K., Robinson, R. N., & Solnet, D. (2020). COVID-19's impact on the hospitality workforce–new crisis or amplification of the norm?. International Journal of Contemporary Hospitality Management, 32(9), 2813-2829.

[3]: Guttentag, D. (2015). Airbnb: Disruptive innovation and the rise of an informal tourism accommodation sector. Current Issues in Tourism, 18(12), 1192-1217.

[4]*:* Nielsen, J. (1993). Usability Engineering. Boston: Academic Press.

[5]: Preece, J., Sharp, H., & Rogers, Y. (2015). Interaction Design: Beyond Human-Computer Interaction (4th ed.). Wiley.

[6]: Rubin, J., & Chisnell, D. (2008). Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests (2nd ed.). Wiley.

[7]: Nielsen, J. (2020). Ten Usability Heuristics, https://www.nngroup.com/articles/ten-usability-heuristics/, accessed on October 15, 2024.

[8]: Bruegge, B., & Dutoit, A. H. (2001). Object-Oriented Software Engineering: Using UML, Patterns, and Java (3rd ed.). Pearson.

[9]: Kling, R. (1977). The Organizational Context of User-Centered Software Designs. UC Irvine: Donald Bren School of Information and Computer Sciences.

[10]: Card, S. K., Moran, T. P., & Newell, A. (1983). The Psychology of Human-Computer Interaction. Lawrence Erlbaum Associates.

[11]: Monk, A. (2008). Noddy's Guide to Usability. University of York, Department of Psychology.

[12]: Norman, D. A. (1988). The Psychology of Everyday Things. Basic Books.

[13]: Shackel, B. (1991). Human factors for informatics usability. Cambridge University Press.

[14]: International Organization for Standardization (1998). ISO 9241-11:1998 Ergonomic requirements for office work with visual display terminals (VDTs).

[15]: Quesenbery, W. (2002). Dimensions of usability, Whitney Interactive Design, LLC.

[16]: International Organization for Standardization (2018). ISO 9241-11:2018 Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts.

[17]: Molich, R., & Nielsen, J. (1990). Improving a human-computer dialogue. Communications of the ACM, 33(3), 338-348.

[18]: Nielsen, J. (1992). Finding usability problems through heuristic evaluation. Proceedings of the SIGCHI conference on Human factors in computing systems, 92, 373-380.

[19]: Nielsen, J. (2000). Why You Only Need to Test with 5 Users, https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/, accessed on October 15, 2024.

[20]: Nielsen, J. (1997). Discount Usability for the Web, https://www.nngroup.com/articles/web-discount-usability/, accessed on October 15, 2024.

[21]: PRIO Hotels GmbH. https://www.prio-hotels.de/, accessed on October 15, 2024.

[22]: WM Hotels GmbH. https://wm-hotels.com/, accessed on October 15, 2024.

[23]: Hotel Maximo Betriebsgesellschaft mbH. https://www.hotel-maximo.de/, accessed on October 15, 2024.

[24]: Centro Hotel Management GmbH. https://www.99-hotels.com/, accessed on October 15, 2024.

[25]: Maalej W., Biryuk V., Wei J., Panse F. (2024). User Feedback Intelligence: On the Automated Processing of User Feedback. Zenodo.

[26]: Landsberger, H. A. (1958). Hawthorne Revisited: Management and the Worker, Its Critics, and Developments in Human Relations in Industry. Cornell University Press.