communicated by:
Lehr- und Forschungsgebiet Informatik 9

Prof Dr. Ulrik Schroeder

The present work was submitted to the Learning Technologies Research Group
Diese Arbeit wurde vorgelegt am Lehr- und Forschungsgebiet Lerntechnologien

Evaluation and Test-Driven Integration of SWOFI lecturer interface
Evaluation und testgestützte Integration von SWOFI Dozierendenschnittstelle

Bachelor-Thesis
Bachelorarbeit

presented by / von

Wallmeyer, Peter
434672

Examined by / Begutachtet von

Univ.-Prof. Dr.-Ing. Schroeder, Ulrik (i9)

Univ.-Prof. Dr. rer. nat. Borchers, Jan Oliver (i10)

Supervised by / Betreut von

Görzen, Sergej

Schäfer, René

Aachen, February 24, 2025

Contents

List of Figures

List of Tables

List of Listings

Abstract

The Scientific Workflow Guide (SWOFI) is a tool for the organization of seminars and similar courses and was developed by the chair of the Learning Technologies Research Group at RWTH Aachen University. It guides students through a process consisting of phases and steps, and it is already being used by computer science chairs at RWTH, but it has some room for improvement. The interface for the lecturers uses old web technologies and is hard to use. It does not load the content dynamically and some features are missing. Because of this, the chair developed an interface which is using modern web technologies like React, ViteJS and MUI. But this interface was not complete and was not tested by actual users. It was not connected to the actual system and some features were not implemented or did not work properly.

The goal of this thesis was to find out the user requirements of lecturers that are using the system and evaluate the interface to develop a new version of it which is better customized to the users and combines the requirements of them and the system. The interface was not connected to the SWOFI system, so the necessary API endpoints were created on the SWOFI-Core using a test framework to ensure the quality and security of the API and they were integrated into the website. Many interface and system requirements of the users were found which were not covered by the interface and the usability of the system was improved according to the results of two conducted user studies. The interface is now fully integrated into the SWOFI system, has additional features, and is better tailored to fit the user's needs. The system is production ready, which makes it ready to use for the lecturers after the deployment.

Chapter 1 Introduction

A short description of SWOFI

The Scientific Workflow Guide SWOFI is a tool for lecturers and students of seminars to communicate easily and efficiently with each other and helps the students to work in a scientific and structured way. It guides students through the seminar in phases and steps, allowing for a good overview of the seminar progress as seen on the SWOFI website [1]. It supports students in seminars by providing them with a central interface for the overview and management of all information and tasks related to the course. Instead of writing e-mails from the lecturer to all students and then each student sending their PDF files back per e-mail, there is a central platform on which the files can be uploaded and reviewed. Also, it gives clarity regarding the process of the seminar, including deadlines and meetings that the lecturer offers for the students to sign up for. This improves the teaching for the students and relieves the lecturers by making the communication more efficient, according to the SWOFI website.

1.1 Functionality of SWOFI

High-level structure of SWOFI

The SWOFI system has multiple components. The lecturers can create courses that represent seminars and define their structure. Lecturers invite their students to the course and the students get an e-mail to join it. The process or workflow of a course consists of phases and steps, as seen in Figure 1.1 from the perspective of a student. Each phase of the process can contain multiple steps, and in each step the student gets a task. A submission can be added to each step, which means that the students can upload files which are then accessible for the lecturers. There are several options for setting a deadline for these steps, including the options to allow late submissions or defining only the duration of the step to make it easier to copy the course for coming semesters. The steps include the option to provide helpful resources like files, links, and checklists for the students. And it is possible for the lecturer to directly give feedback to the submissions on the platform or start a peer review in which the submissions are reviewed by other students in the course afterward.

1.2 Problems of SWOFI

Problems of the currently used interface

The SWOFI system already works and is in use, but it still has some problems. There are already two web interfaces in use to access the platform: one for the lecturers and one for the students. The interface for the students is quite user-friendly, but the lecturer interface is not so intuitive to use and misses some features like individual

[1] https://swofi.net

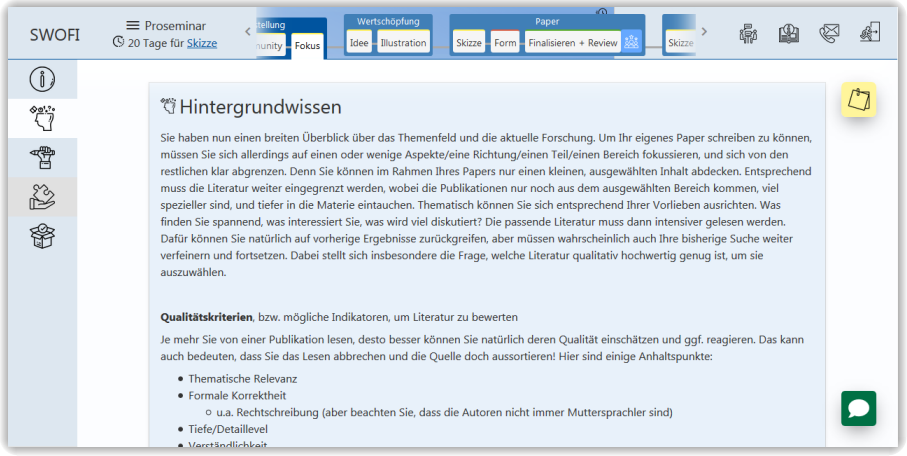

Figure 1.1: Student Interface of the SWOFI system with the workflow at the top

deadlines, as already criticized and stated as future work in a previous paper about SWOFI [UJG+22]. The lecturer interface was not developed with modern web technologies and is less user-friendly. It has the main features that are needed to work, like the course management, submissions, feedback and meetings, but the structure is very nested. Also, newer technologies make use of asynchronous JavaScript loading, which makes it easier to show more information and features on one page. Without asynchronous loading, all content is loaded when the page loads. If there is a large amount of content that possibly requires complex database queries, the loading time increases, as explained in the 2005 publication Garrett, J. J. (2005). Ajax: A new approach to web applications [G+05].

Problems of the new interface

A new interface for the lecturers was already developed by the chair using state-of-the-art technology like React, Vite, and Material UI [MGJS23]. It would be compatible to work with an API to asynchronously load only the content which is needed, but it is not yet connected with the core of the SWOFI system. Also, the structure of the data in the interface is not perfectly suited for the connection with an API. In the core of SWOFI, an API was already implemented for the student client, but it is not complete for the work with the lecturer client because it needs many other API endpoints to enable all the features of the client, like the creation of courses or upload of evaluations.

1.3 Solutions to these problems

Goal and approach to solve the problems

This is the motivation to revise the new lecturer interface in the Requirements Analysis with actual users of the SWOFI system, then rework the user interface, evaluate the changes in another user study, and polish it. In the second part of the thesis, the user interface will be connected with the API, the data structures of the interface will be transformed into a better suitable format for the API, and the API will be implemented completely to support all features of the lecturer interface. So the goal is to make the new modern SWOFI production ready.

Chapter 2 Literature Review

Needed foundations and relevance of SWOFI

Before the approach of the thesis will be described, some essential foundations will be outlined. First, the structure of SWOFI, and the components relevant to this thesis will be explained and then, principles for the implementation and the scientific methods that will be used in the user studies will be introduced. There are many other online learning platforms that help lecturers and students in the learning process, like SWOFI. It will be compared to the platform Moodle to understand why SWOFI is unique and why it is relevant. The similarities, as well as the key differences, between these two platforms will be explained.

2.1 Structure of SWOFI

Low-level structure of SWOFI

The SWOFI system consists of multiple parts. The heart of the system is the SWOFI-Core. It is the server and is connected to the database. It handles all data operations and includes the old lecturer interface, which is currently used by the lecturers. It also provides the API endpoints that are used to transmit data to and receive calls from the student interface. The new interface for the lecturers will use this API to connect to the system. The whole system of SWOFI including the new interface for lecturers and API can be seen in Figure 2.1.

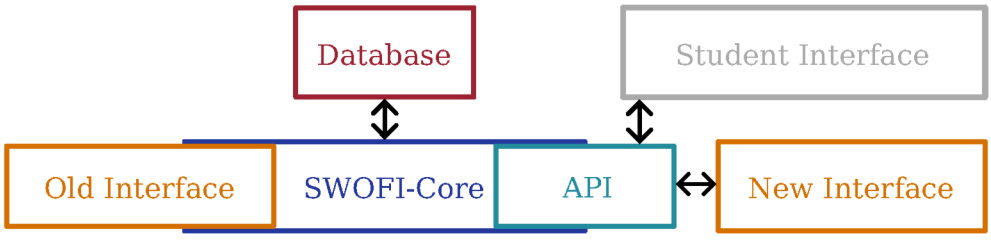

Figure 2.1: System Architecture Integration of coming parts into the current components of the SWOFI system.

2.2 SWOFI database

Advantages of MongoDB for SWOFI

SWOFI uses MongoDB as its database. This is a modern non-relational database system. It is designed for fast development of software and is easily scalable [BGBV16]. Although, this is not so important for SWOFI at the moment, because the user base is very limited, but it could get relevant in the future if the amount of users grows. Also, its data format is very intuitive to use. Data in MongoDB is stored in documents,

which are similar in structure to JSON objects. These documents are stored in collections. This kind of data storage has the advantage that it is very similar to many data schemes that are used in modern programming languages. An object in the program can often be easily stored in a non-relational database. [BGBV16].

Relational data scheme for SWOFI

In comparison, relational databases use rows in tables, which have fields. These contain one value and to create something like an array normally a new table has to be created in which you store these values. To retrieve this data again, joins between tables are used to combine the data again. This approach has the advantage that data can be joined depending on which data is really needed, but the downside of this approach is that the data has to be joined in every data retrieval, which takes a bit of time [BGBV16]. Using SWOFI as example, that would mean that courses and phases would be stored in different tables and every time a course is retrieved, the tables would have to be joined, which slows down the API.

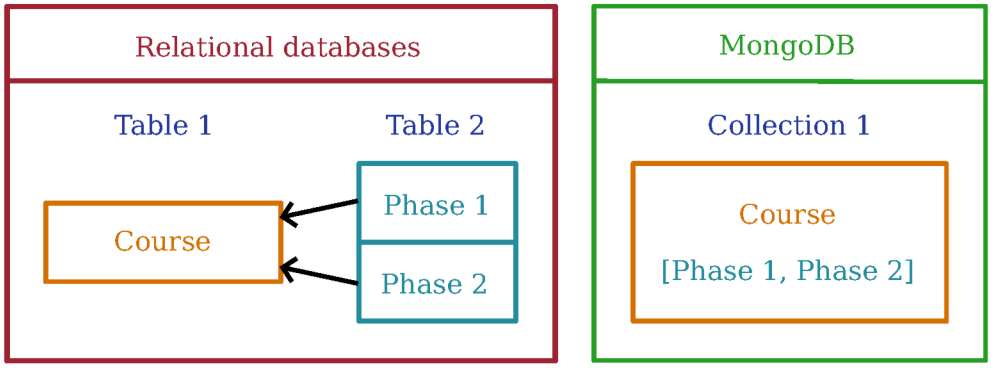

Figure 2.2: MongoDB data storage in comparison to relational databases at the example of SWOFI courses and phases.

MongoDB data scheme for SWOFI

In MongoDB on the other hand, an array can just be created inside the object to store multiple items. When the object is retrieved, all the values in the array are automatically retrieved too, because they are stored all in the same document [BGBV16]. For SWOFI that would mean that the document of the course already contains all the phases and their data, so no extra database calls or joins are necessary.

2.3 REST API principles

Advantages of REST for SWOFI

To transfer the SWOFI data from the database and core to the interface, an API is needed that formats the data in a sensible and uniform way. REST (Representational State Transfer) is an architectural style for designing APIs, which will be used to design the SWOFI API. In this principle, the server and client are clearly separated, and the server has a uniform interface to provide the data as seen in Figure 2.3 for the new interface of SWOFI. One of the key advantages of REST is that RESTful APIs are stateless. This makes it easier to develop the server and client separately from each other without having to worry about side effects in the client as long as the output of the API stays the same. It can also improve the scalability of the service [MAM+17].

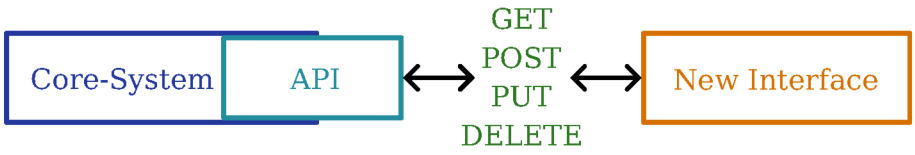

Figure 2.3: REST API principle for SWOFI using HTTP standard methods

Asynchronous REST resource loading

In a RESTful API, everything is a resource. A resource is selected by the URL and often the HTTP standard methods GET, POST, PUT and DELETE are used to signalize, which action should be performed on the resource [MAM+17]. In the case of SWOFI, this helps to make the new interface easier to use compared to the old interface, because through the separation of API and client, the client can load the data without having to reload the whole page again when an action on the server is performed by using asynchronous API calls in the client application.

2.4 API Quality Assurance

Importance of SQA for SWOFI

To always provide the user interface of SWOFI with data, the API also needs to be reliable. To assure the reliability, of the API, the principles of SQA (Software Quality Assurance) will be applied. In the context of APIs that means to test the software regarding its functionality, security, performance, and behavior. The API should return the desired resources and return useful error messages if not. Also, it should be resistant to attacks, answer in a certain duration of time after the request, and behave consistently and as expected [FB15].

2.5 Test-driven development

Concept of TDD

One of the concepts of SQA, which will be used in this thesis is test-driven development. It is a software development process of SQA in which tests are written before writing the software that performs the tasks. This gives clarity about how the API will be structured and forces the developer to think about the functionality as an interface instead of the implementation techniques first. It prevents errors in the implementation of the API by giving the developer a clear sense of what the code should do [Lap10]. Also, it can check whether the principles of SQA apply by trying to access restricted resource and measuring the performance of the API. This leads to better software quality compared to the test-last approach or no testing at all, according to this survey by Desai, Janzen, and Savage (2020) [DJS08].

Steps of TDD

In the first step of developing a new functionality, the test for an endpoint that offers a certain service is written. This test fails, of course, at first because the functionality had not been implemented at that point in time, as seen in Figure 2.4. Then the minimal service is implemented that satisfies the test and the service should then have the desired functionality. In the last step, the code is refactored, which means that the structure of the code is improved and is made clearer as long as the behavior of the code remains the same and all tests still pass. The cycle continues by writing a test for a new feature again as described in the Encyclopedia of Software Engineering by Laplante (2010) [Lap10].

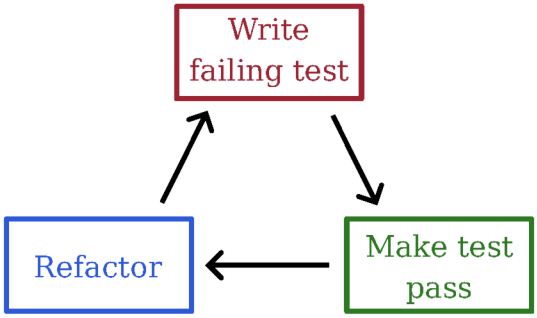

Figure 2.4: Test-driven development cycle for the development of a new feature in a software

2.6 Human-Computer Interaction

Design the user interface of SWOFI

When the data arrives at the interface, the question arises how to display the data. The data in its raw format is not well readable for humans and there the principle of HCI (Human-Computer Interaction) comes into play. It deals with the process of exchanging information between humans and machines. For this, a Human-Computer Interface, also called a user interface, is used, which in this case is the new SWOFI interface. The user interface is a platform between user and machine that should be intuitive to use for the user and lead to a harmonious interaction between human and computer. It has to pass the input of the user to the SWOFI API and the output of it back to the user. To achieve this, interfaces consist of multiple components like display and mouse, but what this thesis will primarily focus on is the software interface. The design of these interfaces is crucial for the effective use of a software. [Cha09].

2.7 User-centered design

Enable efficient interaction with the interface

To improve the SWOFI interface and customize it according to the needs of the users, user-centered design will be used. It is a concept of Human-Computer Interaction in which the users of a system are involved in the design process. The users influence the design of the application through user studies when analyzing the requirements of the application or in the usability testing. The aim is to create a tool that is intuitive to use to shorten the familiarization period needed to interact efficiently with the tool. For this, the inclusion of the end-user into the design process helps to identify requirements for the interface and prioritize certain features that are important to the users [AMKP+04]. This will be realized in two user studies: the Requirements Analysis to learn about the user needs and the Evaluation to see if the changes had a positive effect to the user experience and to refine the interface.

2.8 Comparison to Moodle

Moodle explanation

A system that many universities already use is Moodle. It is a learning management system (LMS) that also supports students and lecturers in the education process by providing a central platform to access information and communicate with the lecturers. It has many features that also exist in SWOFI like providing materials and uploading submissions. But also many more that are not included in SWOFI. Moodle

is due to its plugin system a very flexible tool, which makes it suitable for many use cases [GAB22]. But the plugin system was also stated as a negative point of Moodle in a previous paper about SWOFI because these plugins have to be maintained [UJG+22].

SWOFI guides better through workflows

SWOFI on the other hand is a tool very specialized in seminars or similar courses that follow a workflow-based structure, and there lies its strength. It supports the students by leading them through the workflow in steps and preparing the information. It does not claim to have all the features Moodle has, but tries to present the important features needed for seminars in a way that is easy to understand and use. It guides the students through the seminar instead of just providing them with the materials as explained on the website of SWOFI [1]. This is a big advantage of SWOFI in comparison to other platforms. The design of the interface is specifically designed for workflows like the top bar of the student interface, which is shown in Figure 1.1. It gives an overview of the steps in chronological order and makes it easy to navigate through the process. Additionally, the lecturer platform is also tailored for workflows because it lets users easily access and edit the process, which is possible using features like drag-and-drop in the new interface.

2.9 Similar Tools

Found no other tools with this approach

During the research for tools that also support students in the workflow of a seminar, no other applications were found that were specifically made for seminars or workflow-based courses. In the searches, including “Scientific Workflow,” results about Scientific Workflows in the context of computing and data processing for scientific applications, such as in this paper [JDV+09], were found, with SWOFI being the only exception. In a more general search, many open-source LMSs like Canvas LMS [2] or Chamilo [3] were discovered that support students in learning and relieve the lecturers, but, like Moodle, they focus on traditional courses that provide materials and not on workflows like those in seminars. Additionally, many papers about classroom management in general were found that were not related to online platforms for teaching.

[1] https://swofi.net
[2] https://github.com/instructure/canvas-lms
[3] https://chamilo.org/en

Chapter 3 Approach

UI and data management parts

To understand the structure of this thesis and the topics it covers, the approach will be explained. It is divided into two parts, which focus on the UI and data management. The UI part focuses on evaluating and refining the interface, while the data part primarily addresses the connection between the interface and the SWOFI-Core through the API.

3.1 Revision of the UI

Importance of UI design for SWOFI

The UI is an important part of the system, because it is the part with which the lecturers directly interact with. This makes it all the more important to design it according to the needs of the user. It should be intuitive to use for new users, but at the same time be comprehensive enough to provide advanced users with all the functions they need. To improve the system and adjust the interface more to the user's needs, two user studies will be conducted. First, a Requirements Analysis, in which the new interface which was developed by the chair without any changes will be evaluated, and then a second user study, the Evaluation, will be conducted, in which the changed interface that is then already connected to the SWOFI-Core will be tested.

3.1.1 Requirements Analysis

Need for user studies

The new interface for the lecturers that was developed by the chair, already works well and includes most functionality needed for the system. It has a more intuitive design than the old interface and makes the work with the system easier. It can be seen in Figure 3.1.

User-centered design through user study

Nevertheless, the interface could be possibly improved in some points. Because of that, a user study will be conducted, in which the current state of the user interface (UI) will be evaluated to find its strengths, weaknesses, and the requirements of the users. This is an important step of user-centered design, to better understand the user perspective of the software. The participants will interact with the working version of the interface that is not connected to the Core-System but uses mock data to simulate how it would work if it was connected. The users will have to perform certain tasks on the interface and answer questions about their experience afterward. The goal is to find out where the users had difficulties and where the software could be improved to better meet their requirements.

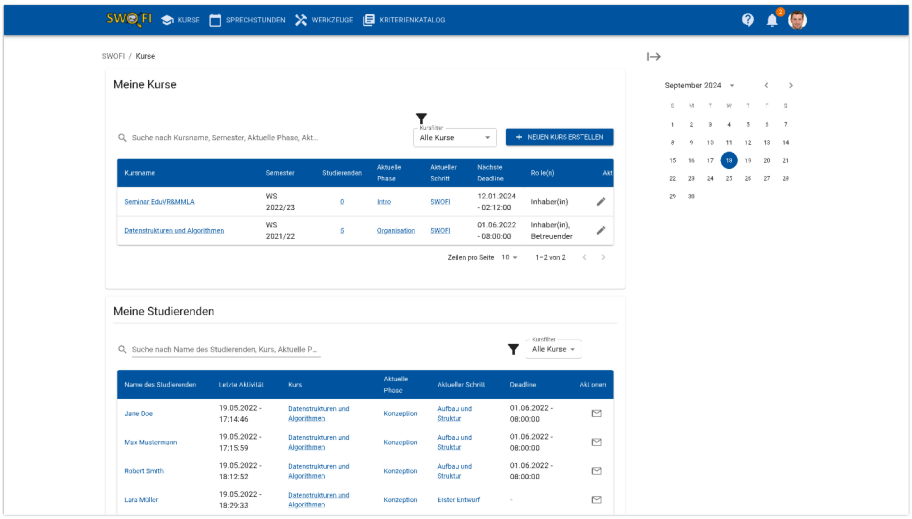

Figure 3.1: New interface of the SWOFI system for the lecturers

3.1.2 Implementing UI improvements

Combining user and system requirements

Then the changes that could help to mitigate the weaknesses that were identified in the Requirements Analysis will be implemented. For this, existing UI components will be changed and new will be created using React and Material UI. The new interface is built with these libraries. A challenge could be to combine the wishes of the participants while maintaining all the functionality needed for the system to work. Additionally, some components of the UI are missing, such as the activation of the peer review feature for submissions, which can be seen in Figure 3.2 or the login screen. These will be implemented and the user feedback will be incorporated to create an interface that is easy and intuitive to use.

3.1.3 Evaluation

Check usability improvement

In a second user study, the improved version of the UI will be evaluated and it is being examined if the changes really improved the usability of the tool according to the users. The participants, who have possibly already tested the tool in the first user study, will then test the improved version of the tool and perform similar tasks again. After that, they get a questionnaire about the usability of the tool to see in which parts the UI got easier to use for the users and where the previous version performed better. This is a crucial part of the user-centered design because it reflects the understanding of the requirements of the users and checks whether the interface was adapted accordingly.

3.1.4 Polishing

Check usability improvement

In the end, the UI will be polished and insights from the evaluation will be used to change the design again if needed. The changes that were made after the requirements analysis will be reflected and kept if they had the desired effect to make the use of the tool easier and more practical. Then, the changes that did not have the

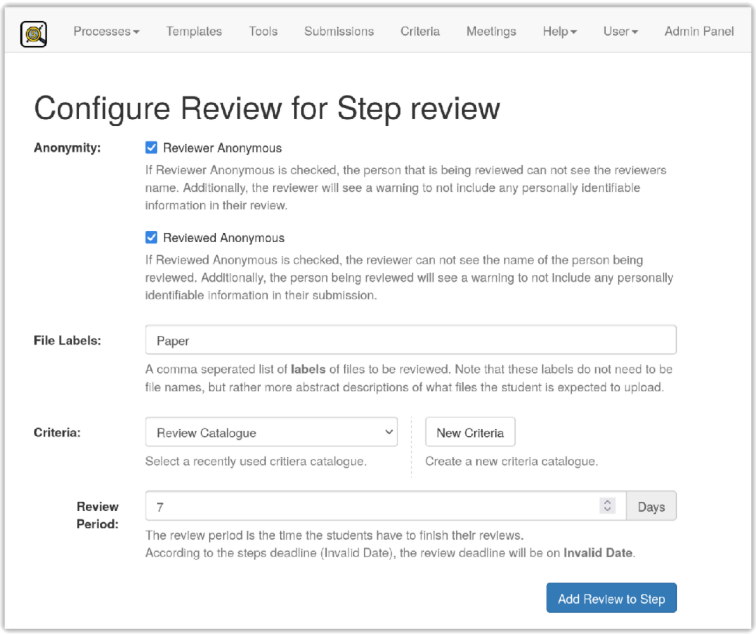

Figure 3.2: Old Interface of the SWOFI system for the lecturers showing the peer review feature

intended effect will be refactored and the interface of the tool will be improved using the combined insights about the user perspective from the Requirements Analysis and Evaluation. This approach of refactoring the interface again, helps to reflect the changes and measures the effects of the adjustments on the user with scientific methods like user studies to follow the concept of user-centered design.

3.2 Approach for the data

Role of SWOFI backend

The other very important part, without which the system would not work, is the Core-System, which acts as the server for the API endpoints. For the correct and reliable functionality of the system, a well-designed and tested backend is crucial. Because of that, the API will be built according to standards like REST and use tests to ensure their quality and reliability.

3.2.1 Connection between UI and API

API service worker and Authentification

The second part of the thesis deals with the data layer of the SWOFI system. To have a good basis from which the development in the direction of the API and of the UI can start, the connection between the UI and API will be established first. For this, an API service worker will be implemented in the new interface from which the service components for the different data objects have a consistent interface to make API calls, like the ProcessService for fetching and performing actions to the processes.

It should handle the connection to the API with an authentication using access tokens. These access tokens can be retrieved by authenticating with OAuth or, alternatively, by using basic username and password authentication, as well as by refreshing existing tokens. The authentication using OAuth and RWTH SSO was implemented for the old interface and the student client, but the new interface client must be changed to be compatible with it.

3.2.2 Updating of UI and data structuring

Update dependencies and reduce API calls

The implementation of the new interface began around two years ago, but the project has not been fully maintained since then. As a result, it relies on outdated dependencies, with the React instance being two major versions behind. Dependencies, that no longer maintained, will be replaced with alternative npm libraries that have similar functionality and support the newest version of React and Material UI to improve the security of the system, which is part of SQA. For Material UI, MUI 6 will be used, which was just released in August 2024 to make the project easier to maintain in the future.

To make the interface better compatible with the API, the goal is to rework the used data structures in the client application. The new data structures help to make the application better-suitable for the connection to an API. At the moment, the new interface accesses the data store very often and partially loads redundant data. The application was developed using only local data storage, so the loading times did not matter. But using an API, each data call takes some time, which is why the reduction of the calls can improve the performance of the application. This is also an important factor for the SQA regarding the SWOFI system in general. So instead of asking the API for a course, and then giving the course ID to a subcomponent that loads the course again, as well as the supervisors of the course in a separate call, the improved application should load the course together with the supervisors and pass the combined data of the course to the subcomponent.

3.2.3 Implementing RESTful API with tests

Current state of the API

The parts of the API that are already implemented in the SWOFI-Core access the MongoDB database using mongoose and typegoose, but the API was written for the student client and is not complete. It can already give back data for certain resources like processes as seen in Listing 3.1, but some data is still missing and the updating of mongoose created some issues. The API interface structure already uses HTTP standard methods and seems to be following common API practices.It will be evaluated if it follows the REST principles and changed accordingly. Maybe the student client, that uses the API as well, will have to be adapted.

```
1  [
2    {
3      "_id": "620a66779519e10013a924f2",
4      "name": "Seminar 1",
5      "description": "A test seminar",
```

```
6       "owners": [
7       {
8         "_id": "62039465f9fcc30012be0bca",
9         "email": "peter.wallmeyer@rwth-aachen.de",
10        "name": "Peter Wallmeyer"
11      }
12      ],
13      "supervisors": [],
14      "isPublic": false,
15      "canBeUsed": false,
16      "groupSize": 1,
17      "phases": [...],
18      "start": "2022-02-12T23:00:00.000Z"
19    }
20  ]
```

Listing 3.1: Example API result to list all processes shortened

Use existing authentification and early error detection using tests

Authentication is an important part of the API and plays a crucial role in the SQA concerning security. The implementation has already been done for the student client, and it works with OAuth tokens that the user gets when logging in to maintain a stateless API. A similar authentication interface will be used for the lecturer client too. The API will be implemented using test-driven development to ensure that the API works fully and fulfills the requirements of the new interface. For that, the tests for the API will be written before the endpoints are implemented to ensure clarity for the communication between server and client and find errors or vulnerabilities of the API already in development to ensure a certain level of security and consistent behavior. In the implementation of the filters and population from the database, errors can occur, which could lead to a wrong or not working functionality in some cases, which would make the API not uniform anymore, like intended by the REST API principles and lead to unexpected behavior in the client application. By using this test method and following SQA principles, these errors should already be detected in the development and therefore can be remedied at an early stage.

Chapter 4 Foundations

API and interface foundation

After planning the approach, a foundation was prepared, with which the API and the interface will be connected. Also, it had to be determined which interface improvements would be general design choices that would have to be made to enable it to work with a delayed data source, the API.

4.1 Core development

Familiarization with the core

It took quite some time to get an overview and understanding of the fundamental structure of the SWOFI-Core. The project is very big, many people worked on it and it has very much source code to familiarize with. But after a while, the system was understood well enough, to begin to think about, on which parts of the system could be worked on and how an API could be implemented to transfer the given HTTP requests and their data into database requests.

4.1.1 Updating of the SWOFI-Core

Removing, updating and replacing dependencies

A problem which occurred was that, also for the core, many of the dependencies were outdated. The system should be transformed into a state with updated dependencies, which was soon realized to be a bigger goal than originally thought. First, it had to be determined which purposes the libraries fulfilled in the system and whether they would be necessary for running the API part of the application or just the UI part, which would only be needed for administrative purposes after the work was completed. The outdated dependencies were then reviewed, and they were either updated, removed if unnecessary, or replaced with modern alternatives if they were no longer supported by newer versions of the framework.

4.1.2 API endpoints

Using express

After the SWOFI-Core was ready for development the creation of the additional endpoints started that are needed for the new interface. For the API, the JavaScript library express[1] is used, which is a popular web framework. The API endpoints were created using the express router. Authentication is implemented through the router using middleware, which then calls the corresponding endpoint depending on the URL and HTTP method. The endpoint then handles the authorization.

[1] Express: https://expressjs.com, visited 22.02.2025

Functionality of API endpoints

To get the information about a course, the GET endpoint for /processes/:id is called. The courses are called processes in the system. It should be called with an actual process ID in place of the :id parameter. The API endpoint can be seen in Listing 4.1. It is already inside a controller for the processes because of which the /processes URL part is left out in the code. First, the validity of the parameters is checked. This happens through express-validator[2], an express middleware for validating requests. Here, only the ID of the process is checked to be of the format of a MongoDB ID. The process with this ID is then searched and the user data, stored in the currentUser variable of the request during the authentication step, is used to check whether the user is authorized to access the process. The function searches in the MongoDB and checks the access rights. If the user is not authorized, the `process_find` function throws an HTTP Error with status code 403 Forbidden which is handled by middleware. If no process was found, the HTTP status 404 Not Found is returned. Else, the process data is returned with HTTP status code 200 OK. The other API endpoints are built similar to this one and are responsible for all requests of the new lecturer client.

```
1  router.get(
2  "/:id",
3  [param("id").isMongoId().withMessage("Invalid ID format")],
4
5  asyncHandler(async (req: Request, res: Response) => {
6    const process = await process_find(
7    (req as any).currentUser,
8    ObjectId.createFromHexString(req.params.id)
9    );
10
11   if (!process) {
12     res.status(404).json({ message: "Process not found" });
13     return;
14   }
15
16   res.status(200).json(process);
17 })
18 );
```

Listing 4.1: API endpoint at the example of retrieving course/process data

4.1.3 Tests

TDD for SWOFI

As already stated in section 2.5, test-driven development of software is a good way to give the developer a clearer sense of what the software should perform and to assure a certain quality and reliability of it. In the context of the development of an API, this strategy means writing a test for an API endpoint and then developing the endpoint itself. The clear separation into different endpoints makes this strategy especially appealing, as the developer can write tests for each endpoint individually.

[2] express-validator: https://express-validator.github.io/docs, visited: 22.02.2025

Jest test framework

For bringing this into practice with SWOFI, Jest[3] was used which is a JavaScript testing framework and is well suited for API tests. The choice fell to this testing framework, as it supports the used technologies and because of its popularity many resources can be found to help in the development.

Helper function for tests

Additionally, to make the development of the single endpoint tests easier, helper functions were developed. All tests use the requestAPI function. In it, the URL of the endpoint, the method, and the input are provided which are sent in the request. The request is further specified by providing which users should be able to perform this action. The helper method automatically creates a test course and instance to check the functionality. There are six different possible users for which the test can be performed: user, student, lecturer, owner, supervisor, and admin. The user represents a student which has no special rights in the system and is not a student in the test process. The student is in the role of a student of the test process. The lecturer has the role of a lecturer in the system, but he does not manage or own the test process. The supervisor and owner are these roles for the course and the admin has administrative permissions.

Helper function example

For example, when editing a course, the student should not be able to edit courses because he is no lecturer, so the error code 401 Unauthorized is expected as seen in line 6 and following of Listing 4.2. The owner should be able to edit the course and should receive a 200 OK status code. In this case, specifying the lecturer and supervisor would be good to test that only the owner of the course can edit it and not any other lecturer. These lecturers get the error code 403 Forbidden, which means in this context that they are allowed to edit courses in general, but not this course because they are not an owner of it.

```
1  requestAPI(
2  "should edit a specific process",
3  "/processes/:processid",
4  "PUT",
5  {
6    httpStatusStudent: 401,
7    httpStatusLecturer: 403,
8    httpStatusSupervisor: 403,
9    httpStatusOwner: 200,
10 },
11 // Before
12 async (user, process, instance) => {
13   return { process };
14 },
15 // Build request (the data that should be changed in the process)
16 async (beforeReturn) => {
17   ...
18   return new NormalRequest(`/processes/${beforeReturn.process._id}`, ...);
19 },
20 // Check response
```

[3] Jest: https://jestjs.io, visited: 22.02.2025

```
21  async (response, request, user, beforeReturn) => {
22    ...
23  },
24  // After
25  async (response, request, user, beforeReturn) => {}
26  );
```

Listing 4.2: requestAPI example to edit a course/process shortened

Details of the helper function

Each test should run independently of others, which may require preparation before performing a test. To do this, a function can be passed to the requestAPI function, which is called before the test is executed which can be seen in line 12 of Listing 4.2. These preparation tasks have to be performed directly on the database to avoid relying on other API endpoints. So the database models are called directly. In this case, no preparation is needed because the test process will be already created by the requestAPI function. Before the actual requests can be sent, the request data has to be put together. For this, a function is called that gets the output of the preparation function, in case it is needed for the request and the return value of this function is then passed to the request. This function can be seen in line 16 of Listing 4.2.

Check the expected output

After the action was performed, the output should be checked as well, in most cases. The status code is already automatically checked depending on the user role, but the returned contents have to be checked manually. For this, another function is passed to the requestAPI call. This function gets the output of the API call and helper functions of the library supertest then check if the values equal the expected values.

Development of requestAPI

This requestAPI helper function concept makes it easy to test up to six different user role scenarios with one test description. It makes it very easy to write many tests and reduce the actual coding work, but it is of course very specific for the use in SWOFI. During the development, the function was changed many times. When the requestAPI function was first written, it was not know which functionality it would need exactly, so it was continuously developed. Over the course of the development, some parameters were added and the function was refined to fit the needs of the tests, but to be also not too complex to use. At the end, a relatively simple to understand solution was created, which contains all options needed for the SWOFI API tests.

4.2 Client side prerequisites

Data and interface changes

The new interface was developed using only locally stored data. To enable the connection to the SWOFI-Core, some preparations in the data management must be made. Additionally, the implementation of certain components and changes to the interface are necessary to allow it to replace the old interface.

4.2.1 Data management and notifications

Loading, data and error states

The notifications are an important part of user feedback, which give the user trust in the reliability of their actions. The user gets validated that the action that he performed was actually successful and that the changes had effect. To signal these different states of user feedback, a data and notification system was introduced,

which represents all possible states of a request. Every data result of the API is of the type `DataType | IError | undefined`, while `undefined` is the default value. Every data that is requested from the API is handled in this scheme. During the time that the request was not answered, and no error occurred, the data state is of value `undefined` and represents the loading state. If an error occurs during sending the request or on the server, an object of type `IError` is returned. It represents an error and contains a short description message of the error. But in the normal case, the returned data should be of type `DataType`, which is just a placeholder here for the TypeScript data type of the returned data. In some cases, no data is returned. Then, the object is of type `INoData`, which contains, like the name already says, no data.

User notifications

This is what happens in the background of the application, but the user also gets feedback about these states all of the time. First, the data is in state undefined and a loading message appears on the top right screen of the user, which can be seen in Figure 4.1. If the action was not successful and an error occurred, the notification that can be seen in Figure 4.2 is shown, including the message that is returned in the error. If it was successful, the notification in Figure 4.3 appears.

Figure 4.1: Loading Notification when creating a course

Figure 4.2: Error Notification when creating a course during having no connection to the internet

Figure 4.3: Success Notification when creating a course

Code for notifications

This behavior is realized in the code in Listing 4.3, which is using helper methods that were written for the notification service. These use the library react-toastify[4] to create the notifications. First, the loading notification is created and only then the actual API call is made using the process service which can be seen in line 1 and 3. After the response came, it is checked whether the response is an IError. In that case, the notification is updated, changed to the error type, and the text is changed which can be seen in line 6. But if the response has no error, the notification is updated to the success notification, which happens in line 14.

[4] react-toastify: https://github.com/fkhadra/react-toastify, visited: 22.02.2025

```
NotificationService.loading(tt("CreateProcess.Loading")).then(
(notificationId) => {
  ProcessService.create(process).then(
  (response: IProcess | IError) => {
    if (isError(response)) {
      NotificationService.update(
      notificationId,
      tt("CreateProcess.Error"),
      "error"
      );
      return;
    }

    NotificationService.update(
    notificationId,
    tt("CreateProcess.Success"),
    "success"
    );
    setNewCourseDialogOpen(false);
    setProcesses([
    response,
    ...(processes && !isError(processes)
    ? processes
    : []),
    ]);
  }
  );
}
);
```

Listing 4.3: Notification and data handling example to create a course/process

4.2.2 Color design

Object manipulation colors

In the existing interface, some inconsistencies were found regarding the colors used for buttons and related actions. Almost all buttons were blue (like for creating a process) and some buttons were green (like for creating a meeting). The idea to connect certain colors with certain actions, that are performed when clicking the button, came up. This button scheme was chosen because it seemed intuitive: blue for creating, orange for editing, red for deleting, as shown in Figure 4.4. It brings the user to associate the color with the corresponding action and hopefully helps him to get a better overview of the possible actions on a site.

Feedback color

For giving feedback to submissions, a similar association was created. The light blue color is used for sending feedback to a submission of a student. The same color is used for the badges, which are shown besides each submission, for example in the table of all submissions. These two components can be seen in Figure 4.5. The light blue color can be associated with giving feedback and helps the user to see faster, to which submission he already gave feedback and to which not.

Figure 4.4: Buttons for creating, editing and deleting resources

Figure 4.5: Feedback design for the status of a submission (left) and sending feedback (right)

4.2.3 Peer review feature

Peer review explained

The old interface has a feature in which the lecturer can create peer reviews for certain steps. In these peer reviews, the students review a submission of another student after the submission deadline is finished. The students get another submission assigned and give feedback based on a criteria catalog which the lecturer chose before. By this method, the students can see submissions of other students and learn from each other. At the same time, the lecturers could be relieved of some of their workload.

Feature and options

A feature like this is included in the old interface, but in the new interface it was missing. So a new section was added in the creation of steps, to add a peer review to a step and therefore also to a submission. It can be seen in Figure 4.6. In the options for adding a peer review, the lecturer chooses a criteria catalog from their own or from templates of other lecturers. Then, he selects whether the reviewer and reviewed stay anonymous or not, and adds descriptions of the files that the students should give feedback on. In the last step, the lecturer sets the duration of the peer review, which determines how much time the students have for giving the review after the submission deadline.

4.2.4 Editable Item List

Design in old interface

In the interface a pattern emerged again and again, that was a bit irritating at first. List of items were always created by providing a text field, and each item had to be typed in to a separate line. This was used, for example, for creating checklists and inviting new students to a course. This might be an easy solution to create the functionality and it was also used in the previous interface of the SWOFI-Core, but it is not that intuitive as the Requirements Analysis will show.

Creation of component

Because of that, a new component was created, which does exactly that. The solution should be simple to use and it should still have all functionality, which is to add,

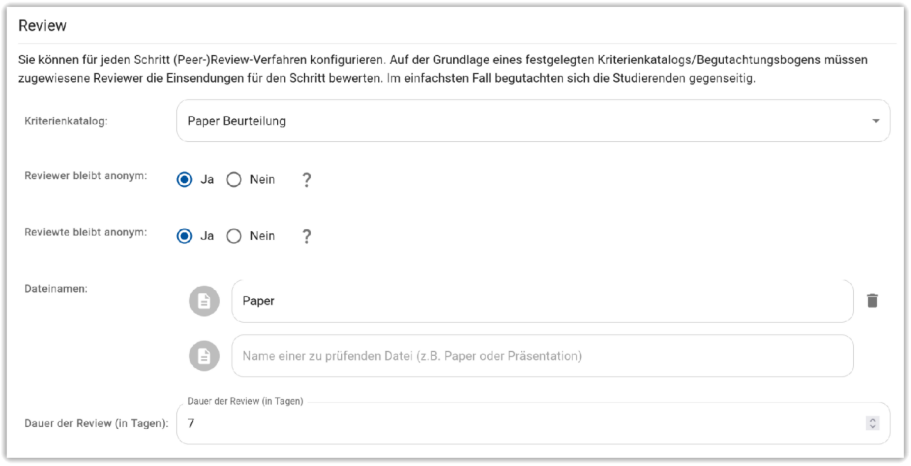

Figure 4.6: Creating a peer review in the new interface

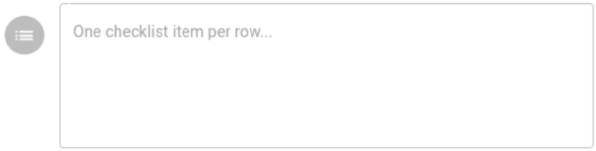

Figure 4.7: Previous component used for lists at the example of checklists

remove and edit items. Also, the number of button clicks should be minimal to create items. After a bit of experimenting, a component was created that is well received by the users, as it will turn out in the user studies later. The first item has just one text field with an icon and a placeholder. As soon as the user types in something, another text field for a new item appears below it. By this, the user does not have to click a button to create a new item. If the user wants to remove an item, he can either click the remove button on the right side of the item or just remove the text inside the text field which leads to the removal of that field and changes the focus to the empty field which is at the end of the list to add a new item. The component can be seen in Figure 4.8.

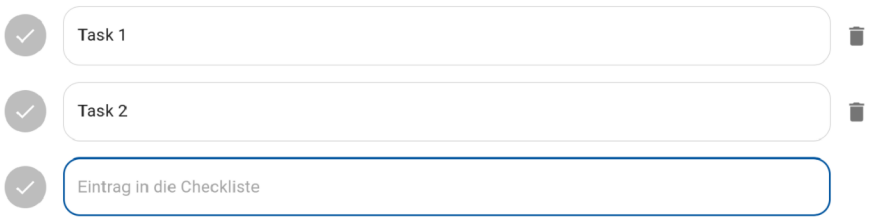

Figure 4.8: Editable Item List at the example of checklists

Additional component options

A validate function can be passed to the component to enable checking of the input fields, like if the item text is an email. And the onChange function returns, additionally to the item values, if all current items are valid and if all items are empty. This makes it easily possible to deactivate the submit button of the form if the item has invalid values or if the list is empty. As seen later in the user studies, it also received positive feedback, as the new component was well received.

Chapter 5 Requirements Analysis

Structure of the user study

The Requirements Analysis has the purpose to get the opinion from users of the interface to find parts where difficulties during the use occur, which could be improved and which already give a good user experience. But also, to get user requests for new features that they missed in the old interface or which are existing in the old interface, but are missing in the new one. It started with a short introduction about what SWOFI is and how it works to bring the participants all on the same page. They could read it, or the content could have been explained to them. The user study took around 60–90 minutes per user to conduct, and it consists of three steps: the general requirements, the guided use and the feedback.

5.1 General requirements for SWOFI

Expectations of the participants

The first step was an interview about SWOFI in general. The goal was to find out what potential users of the system would expect of a tool like SWOFI. Being aware of that it makes a difference in the answers depending on whether a person has already used SWOFI, the first question, that was asked, was whether the participant did. Actually, all three participants already used SWOFI, so in the analysis of the results that has to be kept in mind.

Submissions and Feedback

SWOFI is a workflow based tool and the main purpose is to combine all information and submissions in one place. So, next, the participants were asked about what kind of feedback tools and options they would like to use to give the students feedback. It turned out that many used the existing peer-review feature of SWOFI, in which the students give feedback to submissions of other students. Other than that, they liked to give feedback per text and PDF, like it is already implemented. But a thing that came up several times was the ability to give feedback in a rich text editor to better format the text. To give meaningful feedback, the submissions should also have a good format, so the lecturers were asked about in which form they would wish the students to submit their results. In the old version of SWOFI, there was the ability to upload files and add a comment, which the participants liked and no wish for different formats came up. The same applied to the materials and background information that is given by the lecturer. The existing functionality of links, checklists, PDF files and text was already sufficient.

Accessibility of submissions

But the most important outcome of this part of the user study was the insight in how important the accessibility of the submissions overview should be. The lecturers create or copy the course at the beginning of the semester, but during the semester they

rarely touch the course settings again. The main focus lies then on giving feedback to the students. In numbers, if a lecturer creates a course, he accesses the course once, but if his process has five phases with submissions and there are 10 students in the course, he will already open the submissions view 50 times in the course of the semester. So, this functionality will have to be made especially easily accessible.

Scope of the tool

Regarding the question if the tool should have more new functionality, the lecturers answered that it is suited for this one kind of workflow-based courses, so not many other features are needed. The appeal of the tool also lies in the fact that it contains all the features needed for seminars or similar courses, but does not try to be suitable for all kinds of courses, like Moodle.

5.2 Guided use and resulting changes

Content and results of the usage

In the next part, the participants got a step-by-step guide to use the tool. It covered most of the functionality of the new interface, but some features like the criteria catalogs did not work in the existing version, so they could not be included. The use was centered all around the seminar “3D Shape Analysis”, which is an actual seminar at RWTH to make it a more applied example. Although the study was of course limited in time, so a really realistic course could not be created by the participants. After the use of the tool, the participants gave feedback through structured conversation, in which they were asked about the different parts of the tool. In the following sections the use of each part, the feedback of the users, and the most relevant changes that were made to these parts are illustrated.

5.2.1 Creating a course

Dialog combination and group size

The participants were requested to create a course “3D Shape Analysis” starting in summer semester 2025. The creation of courses in the interface was combined in one dialog with the copying of courses as seen in Figure 5.1. This was not clear for all the participants. Also, the parameter of the group size to create submission teams is part of the data model of the SWOFI system, but the core does not support this feature, so it should not be shown here.

Separation and semester picker

To improve the user experience of the course creation and copying, the dialog is now separated into two dialogs. This should bring more clarity to the fields which have to be filled out. The new creation dialog can be seen in Figure 5.2. The group size field was removed and a semester chooser added to differentiate between a course which exists in multiple semesters. Also, the date and time picker were separated into two separate pickers to improve the usability because it led to confusion for some users when they tried to edit either only the date or only the time.

5.2.2 Inviting students

Add student confusion

The next instruction was to invite a student into the course. To do this, the participants first had to go into the tab *Invitations* on the right side of the course page, which was not so intuitive for the users because there was also a tab *Participants*,

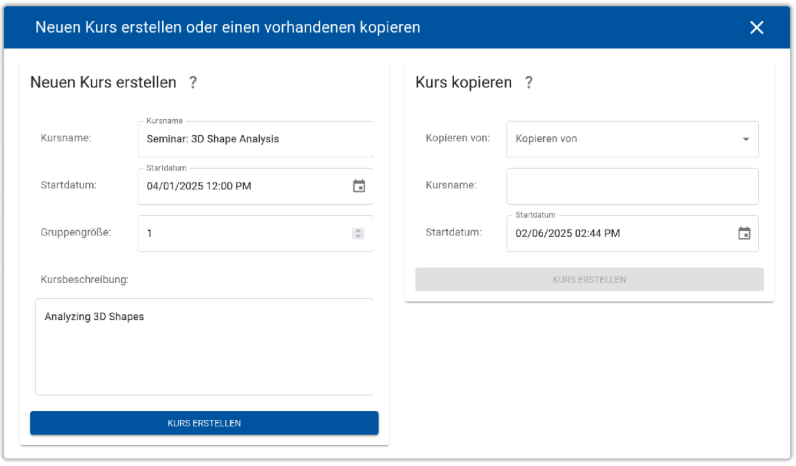

Figure 5.1: Creating a course in the previous interface

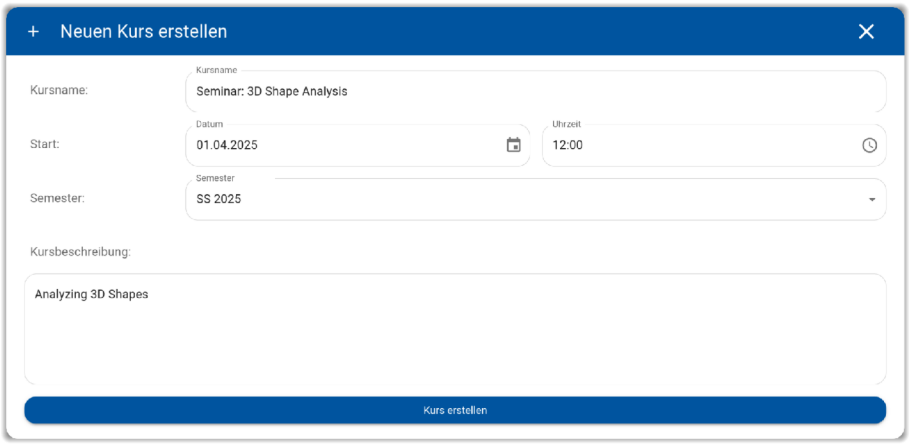

Figure 5.2: Creating a course in the reworked interface

in which the invitation of students is not possible. After clicking the button to invite students, a dialog appeared, which can be seen in Figure 5.3. In it, the e-mail had to be typed in, but to be able to submit the dialog, the button with the plus on the right side of the typed e-mail had to be clicked. This also led to some confusion. The participants expected that after typing the student's e-mail in, it would be possible to submit the dialog and to only click on the plus button to add another student to invite to the course.

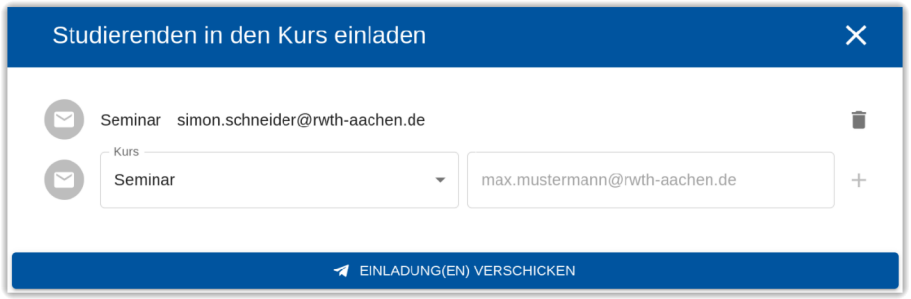

Figure 5.3: Inviting a student to a course in the previous interface

Reorganizing tabs and using Editable Item List

To mitigate those weaknesses of the interface, the tabs were changed to only have one tab *Students* (instead of *Participants* and *Invitations*) and one for the supervisors. This removes confusion during the use of the interface and makes it directly clear where to click. The invitation button and dialog was also moved to the students section. The invitation dialog was simplified by only providing a text field in which an email can be typed in. After typing at least one character, a new text field appears below in which another student can be added to the invitation list in such a way that there is always one empty text field to optionally add another student as seen in Figure 5.4. For this, the Editable Item List component is used, which is further explained in the subsection 4.2.4. This component removes the necessity to manually confirm each student email by clicking the plus button.

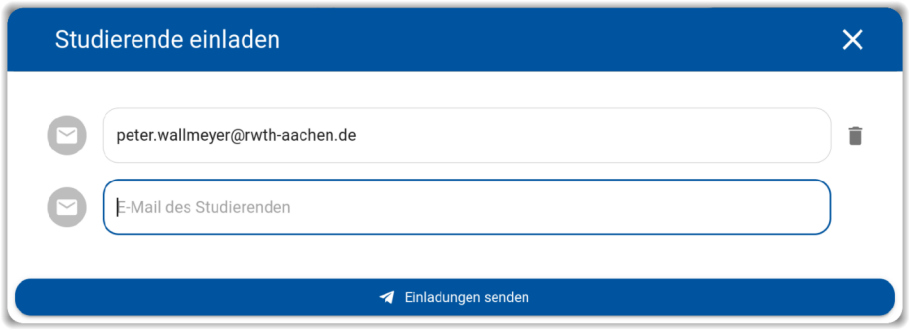

Figure 5.4: Inviting a student to a course in the reworked interface

5.2.3 Creating phases

No problems for first phase

The users were asked to create a phase in the course that they created. The first phase was no problem for all participants, because the interface shows a button *Create Phase*, on which they have to click, and a dialog opens, in which the name of the phase has to be typed in. The interface can be seen in Figure 5.5. After submitting the dialog, the phase is created.

Figure 5.5: View to create the first phase in the interface

Hidden button for second phase

Creating a second phase, on the other hand, was a problem for the participants, and it had to be explained to them how to do it. To create a phase, once at least one phase exists, the mouse must be hovered below an existing phase to show a small overlay with a plus, which when clicked on, opens the creation dialog. It can be seen in Figure 5.6. This is a nice feature for power users that know the system very well and want to create a phase at a specific index in the list, but for the people that do not know this feature, it can become a real problem if they do not find it.

Adding extra buttons

To address this problem, two buttons were added above the table to create a phase and a step, that create them at the end of the process, so after the last existing phase or step. This makes it easy to create a process, because the phases and steps can

be created in chronological order and when a change is necessary, the drag and drop handles can be used to change the order of the steps.

Figure 5.6: View to create a second phase in the interface

5.2.4 Creating steps

Long scrolling dialog

Another task for the participants was to create a step with the given information about it. To create a step, the user had to type everything into a dialog that is very long and requires scrolling. There is much information needed to create a step and bringing all of this on one site makes it a bit confusing according to the participants and the wish to make it more compact and clearer came up. For example, the creation of checklists takes much space, because it needs multiple fields to type the information in. The previous interface for this can be seen in Figure 5.7. The way to type in checklist options was criticized as well, because typing in the options into a text field with a line for each option seemed not to be so intuitive for the participants. Additionally, the same problem that was stated for the creation of phases applies here, when creating multiple steps.

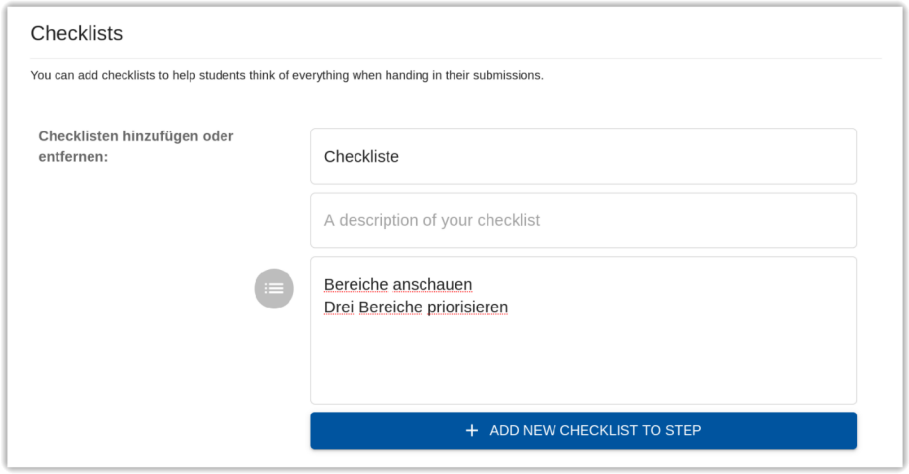

Figure 5.7: Adding a checklist to a step in the previous interface

Step-by-step process

Facing these issues, the dialog was changed to a step-by-step process. The creation of a step is now divided into five steps which can be seen in Figure 5.8. The first step contains the general information about the step, like title and goal. In the second step, the duration of the step has to be given and optional there is an option to set a manual deadline. In the next step, the actual task and background is written. Then the materials, links, and checklists are added and in the last step the tools for the step are chosen and the peer review feature can be activated. This process guides the user through the process of creating a step (Very much in the spirit of SWOFI) and gives the user a better overview of where the options are. As it later turned out in the evaluation, the feedback from the participants was positive, and the responses

indicated that they liked the dialog more than before. At the beginning, it was not clear whether the users would actually use the steps at the top for the navigation or only the previous and next buttons at the bottom, but in the Evaluation the participants chose to use both.

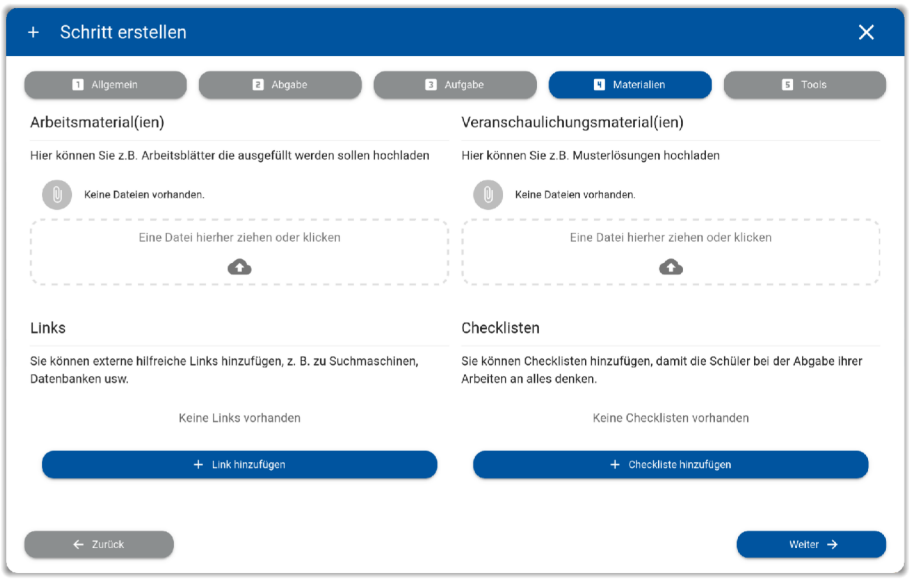

Figure 5.8: Adding materials, checklists, and links to a step in the reworked interface

Relocate forms for better clarity

To make the interface tidier, the creation of links and checklists was relocated into separate dialogs, instead of using the space inside the dialog for step creation. This allowed bringing all four options for adding materials onto one page as seen in Figure 5.8, which is an important step to give a good overview, because these four categories were the most space consuming components in the step creation before. For the checklist options, the text field was replaced with the Editable Item List component which is further explained in subsection 4.2.4.

5.2.5 Creating meeting time slots

End of semester button

To create a meeting time slot, for which the students can register themselves to schedule a meeting with the lecturer, there was a dialog in the meetings section, in which the lecturer types in the start date and time and the end time as seen in Figure 5.9. There was also an option to create a weekly meeting time slot and an end of the recurring meeting. The button on the left of the end date sets the end of the recurring meeting time slot to the end of the semester, in which the start of the meeting lies. Although the task for the study participants was to create a time slot until the end of a semester, no one used this button. In the conversations afterward, it turned out that this was because the button has no caption. So, this was resolved by adding text to the button.

5.2.6 Student submissions

Opening a submission

The submissions of all students in a course could be accessed by clicking a button on the course page. In the newly opened interface, the lecturer could click the pencil icon

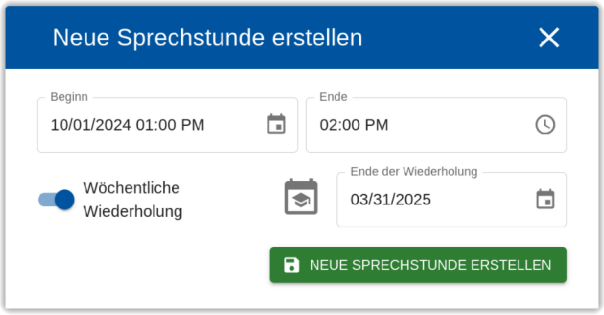

Figure 5.9: Creating a meeting time slot in the previous interface

on the right side to open a dialog for providing feedback, as seen in Figure 5.10. The users were already familiar with this from the currently used interface, where this was also the way to do it, but they mentioned that they did not find it intuitive, especially for new users. The links visible in the table only redirected to the course page and not to the feedback page, which, as the participants mentioned in the conversation, caused confusion due to the large number of links.

Figure 5.10: Submission overview in the previous interface

Add information and simplify opening

To change this, most of the links on the rows of the table were removed, like seen in Figure 5.11. Now there is only one link left that brings the lecturer to the submission where he can also give feedback. Additionally, the user can now click anywhere on the item to open the submission page. This makes it easier to open and removes the confusion about which click leads where, through simplicity. Also, the lecturer can now see directly whether he already gave feedback to a submission by looking at the chip on the right side, which is further explained in subsection 4.2.2. Another important info, that was missing, was the name of the student who submitted the submission, so it was added.

Figure 5.11: Submission overview in the reworked interface

5.2.7 Individual deadlines

Extend deadlines individually

A functionality that was requested in the Requirements Analysis and was also mentioned as future work in a previous paper about SWOFI was to set extended deadlines

individually for certain students [UJG+22]. Lecturers, who used the SWOFI system before, criticized that if students need more time than expected, they cannot upload their submission themselves unless the lecturer has allowed submissions after the deadline for all students.

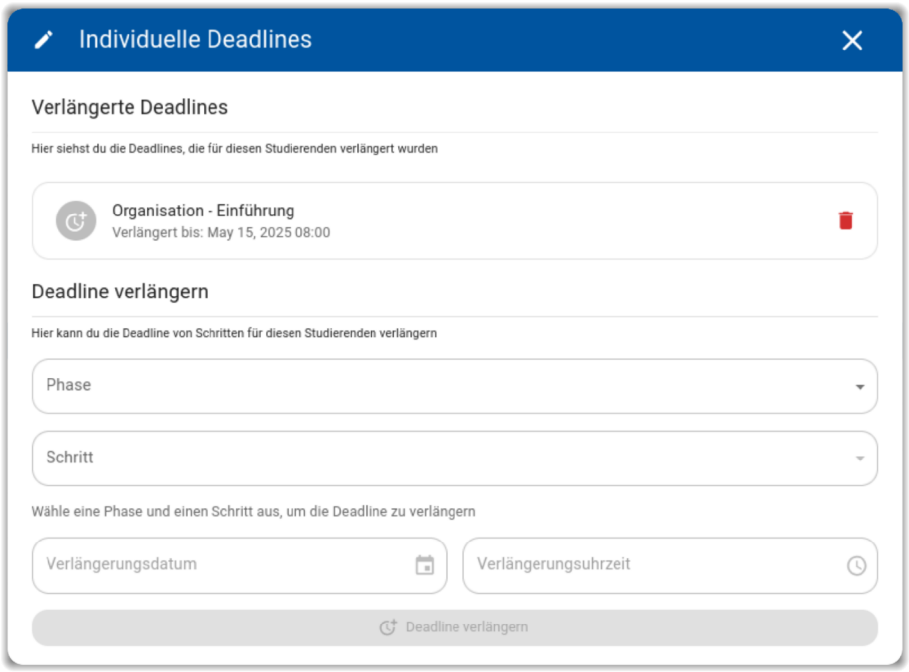

Figure 5.12: Dialog to manage the individual deadlines of a student in the reworked interface

Deadline dialog structure

This functionality was added so that the lecturers would not have to upload these submissions themselves if they get them sent per e-mail after the deadline as seen in Figure 5.12. At the top, a list of all already extended deadlines is shown and these can be deleted using the button at the right of the items. Below that, there is the interface to extend a deadline for a certain step in a phase using dropdown menus and date/time pickers. The date and time pickers were separated from each other instead of using a combined date and time picker, because the combined component led to confusion for some users during the course creation.

5.2.8 Calendar

Calendar unnecessary

On the right side of each page is a calendar, which the user can open and close with the button which can be seen at the top left of Figure 5.13. The participants were asked whether they found it sensible and would use it, and it turned out that with no further options or functionality in the calendar, none of them would use it. So to keep the interface of SWOFI simplistic, the calendar was removed from the sidebar completely in the reworked interface.

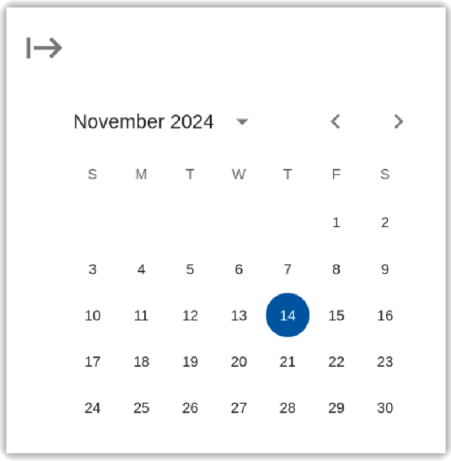

Figure 5.13: Sidebar calender in the previous interface

5.3 System Usability

SUS explanation

To test the usability of the system, the participants also completed a survey in which they were asked to fill out the System Usability Scale (SUS), which is a popular scale for testing the usability of a system on a scale of 0 to 100 and is very robust and reliable [BKM08]. The users answer ten questions about the usability of the tool on a range of 0 to 4, on which 0 is strongly disagree, 2 is neutral, and 4 is strongly agree. By reversing the scores of the negatively worded statements, which are the even-numbered, by subtracting them from 5 and adding up all scores, a number between 0 and 40 is the result. By multiplying this number with 2.5, a score between 0 and 100 is the result, which is an indicator for the usability of the application.

Different format and possible effects

The new interface developed by the chair, without any changes, got a total score of 79.3, which is normally already a really good score. Mistakenly, the participants were offered 6 instead of 5 options, which could influence the results and make it less comparable to ratings of other user interfaces. With 6 options the ratings range from 0 to 5 which after adding up results in a score between 0 and 50 because of which it was multiplied with 2 to get a score between 0 and 100. This change in the user study compared to the normal SUS scale removes the option of a neutral position and adds a slightly disagree and slightly agree instead. This forces the participants to choose a position and that could have an effect on the results, but the general direction of the results should not change through this change. The requirements analysis and the evaluation were conducted with the same format, so the comparison between those two states of the interface is still possible.

Especially positive and negative results

The average results of the user study can be seen in Table 5.1, where the scores are rounded to one decimal place. The statements that were answered very positively were about whether the users would use the tool regularly, whether they would need technical support to use the system, and how cumbersome they found the interface. This showed that the general concept of a new interface was received very well and that it is already good usable. The statement in which the interface scored worst, was the statement about the inconsistencies in the interface. This showed that a task for would be to make the experience of the users more consistent, which was also

mentioned in the structured conversation. The two question which scored the worst after this one were whether the different functions were well integrated into the system and whether most people would learn to use this system very quickly. These results showed that there were some usability aspects in the details that could be improved in the interface, which also became clear during the conversations.

SUS Statement	**Average Score (0-5)**
I think that I would like to use this system frequently	4.3
I found the system unnecessarily complex	4.0
I thought the system was easy to use	4.0
I think that I would need the support of a technical person to be able to use this system	4.3
I found that the various functions in this system were well integrated	3.7
I thought that there was too much inconsistency in this system	3.3
I would imagine that most people would learn to use this system very quickly	3.7
I found the system very cumbersome to use	4.3
I felt very confident using the system	4
I needed to learn a lot of things before I could get going with this system	4
Total Usability Score	79.3

Table 5.1: Requirements Analysis SUS survey results

Chapter 6 Evaluation and Discussion

Structure of Evaluation

After the interface was adjusted to better meet the users needs using the results of the Requirements Analysis in chapter 5 and the interface was modified like described in chapter 4, the evaluation began to determine whether the changes had a positive effect and which components still had room for improvement. It consists of two parts: first the guided use of the system and after it again the questionnaire and the structured conversation. In this user study four lecturers participated, of which three also already participated in the first user study. This has to be kept in mind when looking at the results, the users could be influenced by already having experienced the user interface one time before.

6.1 Guided use and resulting changes

Comparability of the user studies

In the guided use part of the user study, the participants followed instructions similar to those of the Requirements Analysis to keep the two user studies comparable. The implementation of the criteria catalog functionality was fixed, so this part could be included in the user study too. Some smaller bugs were found in the Evaluation that are not included in this section, because they were small changes that do not change the user experience significantly. Also some colors like the rich text editor color and student submission comment were changed to have a more neutral color.

6.1.1 Creating meeting time slots

Start of next semester button

The participants were again asked to create a specific time slot for meetings with students. After the button, to set the meeting end to the end of the semester, was made better visible, resulting from the changes through the Requirements Analysis, the participants used it in this user study. Because the participants did not notice the button in the first user study, the suggestion to add a button for setting the beginning of the time slot to the start of the following semester only came up in the Evaluation. To realize this, the design of the dialog was changed to better fit all the components and a button was added to add the functionality. The reworked dialog can be seen in Figure 6.1.

Manual time input bug

It was also noted that when a participant tried to input the meeting time slot using the keyboard, the application encountered a bug. When writing the time manually in the field, the time value would be invalid as long as the time was not written completely. For example, “13:” is not a complete time and therefore the Date constructor of JavaScript would create an *Invalid Date* object. The application did not catch this

case and had to be reloaded to work again. The MUI time picker was always used when selecting the time in the testing of the application, so it was not noticed before. It was fixed, and now it is working as intended and shows an error message as long as the input fields contain an invalid date or time value.

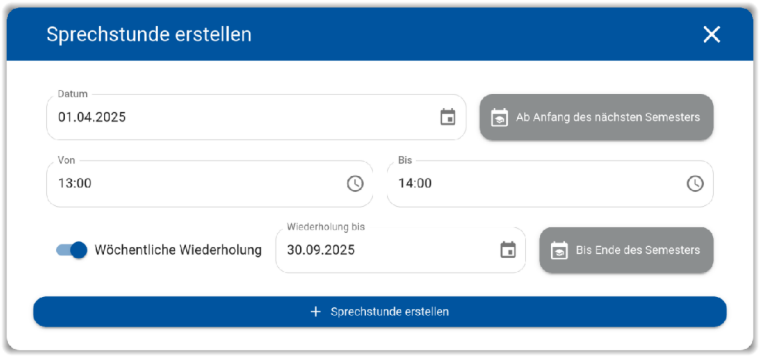

Figure 6.1: Dialog to create a meeting with additional button to set the date to the beginning of the next semester in the reworked interface

6.1.2 Editing criteria catalogs

Complexity of criteria catalog

The criteria catalog functionality could not be tested in the Requirements Analysis, because it did not work in the initial interface, so all the insights about it come from the Evaluation. The criteria catalog was one of the most complex parts of the application for the users, because it is not so intuitive and a more abstract part to imagine in comparison to courses, which most lecturers are used to from other platforms.

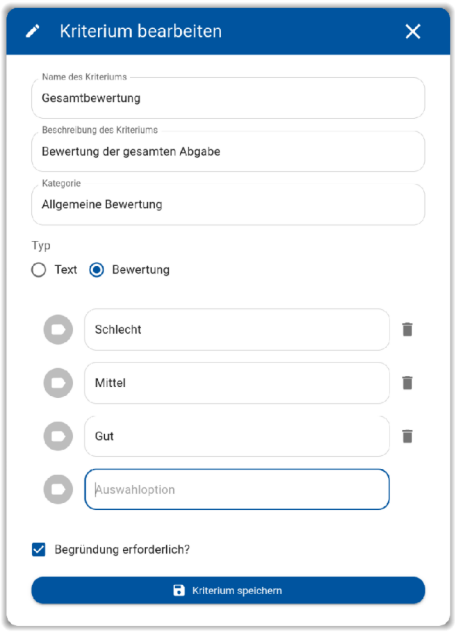

Figure 6.2: Editing a criterion of a criteria catalog in the reworked interface

Fix editing of criteria

In a criteria catalog, a criterion can be created that is assigned to a category. These criteria catalogs can be used for peer reviews, in which the students review submissions of other students, based on these. The functionality was already working and the creation of a criterion worked well, but when editing it, only texts fields were shown to change the title and description of the criterion. This made editing clearer, as no extra dialog needed to be opened. However, it was overlooked that the criterion has a type. It can be "text," where the student types feedback, or "options," where the student selects an answer from predefined choices set by the lecturer. This type was not editable anymore after creating the criterion. Therefore, the interface was changed to open a dialog similar to the one used for creation, allowing the criterion to be edited, as shown in Figure 6.2.

6.2 System Usability

General improvement

The System Usability Score for the interface in the Evaluation was 88.5 which is an improvement to the Requirements Analysis of +9.2. The average results can be seen in Table 6.1 and are again rounded to one decimal place. In the “Change” column, the difference of the results in comparison with the results of the Requirements Analysis is listed. The results are maybe not fully comparable to other interface studies, like already explained in section 5.3 of the Requirements Analysis, but for the comparison between these two user studies it works, because the same scheme was used.

SUS Statement	**Average Score (0-5)**	**Change**
I think that I would like to use this system frequently	4.8	+0.4
I found the system unnecessarily complex	4.8	+0.8
I thought the system was easy to use	4.5	+0.5
I think that I would need the support of a technical person to be able to use this system	4.8	+0.4
I found that the various functions in this system were well integrated	4.3	+0.6
I thought that there was too much inconsistency in this system	4.5	+1.2
I would imagine that most people would learn to use this system very quickly	4.0	+0.3
I found the system very cumbersome to use	4.5	+0.2
I felt very confident using the system	3.5	-0.5
I needed to learn a lot of things before I could get going with this system	4.8	+0.8
Total Usability Score	88.5	+9.2

Table 6.1: Evaluation SUS survey results and changes

Changes compared to Requirements Analysis

The most significant change compared to the Requirements Analysis appeared in the statement about the inconsistency of the system. There the average value changed from 3.3 to 4.5, which is an improvement of +1.2. This indicates that the improvements for the consistency of the system like the color design in subsection 4.2.2 had an effect on the users. Also, the simplicity and ease of use improved by +0.8, which could show that the users could interact with the interface more easily, but certain improvement of this could also be because three of the users participated in the first Requirements Analysis and, therefore, already know how the system works in certain parts. The only statement where the score got worse is the statement about how confident the users felt using the system. There the average score dropped from 4.0 to 3.5 which is a change of -0.5. This could be caused by the addition of the criteria catalogue to the user study, but as the SUS scale does give reasons why the usability score changes, that is not sure.

Discussion of the changes

In these results, a quite noticeable improvement can be seen. The usability scores increased for almost all statements and created a more than 10 percent higher score in total. The changes that were made to build the foundations and components that were improved in the Requirements Analysis had a real effect on the users, and made the overall system better usable. The goal to improve the interface, connect it to the SWOFI system is reached and the plan was successfully realized. The time plan was not followed exactly, because some things took shorter or longer than expected, but overall the thesis stayed approximately in the time schedule, which was defined in chapter 3.

Chapter 7 Conclusion and Outlook

Impact and future extensions

The thesis is complete, and all steps of the approach have been finished. In this final chapter, the impact of the work on the SWOFI system will be explained and it will be highlighted how the improvements and changes contribute to the overall functionality and user experience. Also, potential future extensions of SWOFI will be proposed.

7.1 Conclusion

Impact on SWOFI

The results of this work had a great impact on the SWOFI system. The SWOFI-Core is now primarily used for API access and not for the lecturer interface anymore. The new lecturer interface is fully integrated into the SWOFI system and was further improved. The overall improvement of +9.2 points in the System Usability Scale for the new interface and a better score for nine out of ten statements in the SUS shows that the changes that were made because of the Requirements Analysis and the Foundations had a positive impact on the usability of the system. The new lecturer client is fully usable and contains all necessary components to work as intended. Hopefully, this will make it easier for lecturers to use SWOFI in the future. In April, I will begin working for the chair and assist in completing the final steps required for deploying the SWOFI system with the new interface, which will be available at https://swofi.elearn.rwth-aachen.de.

7.2 Future Work

Improvement ideas

The platform is now already complete and has all important features according to the results of the user studies. Nevertheless, some improvement ideas came up during the process, which could further improve the system, but that go beyond the scope of this bachelor thesis.

7.2.1 Group Submissions

Submissions in teams of students

The system currently only supports submissions of single students that the lecturers give feedback to, but the system already has some structural preliminary work to extend the functionality to submission groups of multiple students. For this, the students in a course are not participants of the course, but are part of an instance which belongs to the course. Currently, there is always only one student in an instance, but SWOFI would theoretically already support having multiple students in an instance.

Open questions

The feature would be possible, but some questions stay open. If a student was not assigned to an instance already, the question arises whether the student can be part of a process, but not in an instance and where to store the participants which have no instance. Another question would be if a student automatically joins an own instance when joining the course, like it is now. But then the lecturer would have to move the student to another instance to create groups, which could be cumbersome.

Not so important for study participants

Also, it turned out that for the participants of the study, the submission in groups is not very important because in their courses all people worked alone, which is also common practice at RWTH. It would probably add complexity to the system, and the questions whether it is worth it to add this feature in exchange with simplicity would have to be answered first. But maybe there is a way to keep it simple and adding the feature at the same time.

7.2.2 Moodle integration

Transfer Moodle data to SWOFI

The main course management system at RWTH and many other universities is Moodle. An integration of Moodle into the platform would make it easier to create courses by copying the course information from Moodle and invite or add all participants of the Moodle course into the SWOFI course. This would relieve the lecturers by not having to input all information and invite students which were already added to Moodle.

Open questions

Although also, this approach lets some questions arise. The RWTH uses the SuPra system[1] for the distribution of seminars. If this system cannot automatically create Moodle courses for the lecturers, they would have to add the students manually. In that case, the integration into Moodle would not make much sense for the use case at the RWTH, but an integration into the SuPra system would be helpful.

7.2.3 Bachelor and Master Theses

Workflow in theses

The SWOFI system could also be used for bachelor or master theses. The workflow based approach also applies to theses and could be used by supervisors to guide students through the process of creating a bachelor thesis. In that case, the process would only contain one student.

Adaptation of SWOFI structure

For this, the structure of SWOFI would have to be used intelligently. The course model could be used, containing one instance with only one student inside of it. But then restrictions would have to be added that in the case of a thesis no other students can be added to the course. And the kind of course (course or thesis) would have to be set at the creation of the course and be not changeable. Also, the design of the course page would have to be changed and the requirements that are needed for students writing a thesis should be reevaluated.

[1] SuPra: https://sc.informatik.rwth-aachen.de/de/supra, visited: 22.02.2025

Appendix A Bibliography

[AMKP+04] Chadia Abras, Diane Maloney-Krichmar, Jenny Preece, et al. User-centered design. *Bainbridge, W. Encyclopedia of Human-Computer Interaction. Thousand Oaks: Sage Publications*, 37(4):445–456, 2004.

[BGBV16] Kyle Banker, Douglas Garrett, Peter Bakkum, and Shaun Verch. *MongoDB in action: covers MongoDB version 3.0*. Simon and Schuster, 2016.

[BKM08] Aaron Bangor, Philip T Kortum, and James T Miller. An empirical evaluation of the system usability scale. *Intl. Journal of Human–Computer Interaction*, 24(6):574–594, 2008.

[Cha09] Gong Chao. Human-computer interaction: process and principles of human-computer interface design. In *2009 International Conference on Computer and Automation Engineering*, pages 230–233. IEEE, 2009.

[DJS08] Chetan Desai, David Janzen, and Kyle Savage. A survey of evidence for test-driven development in academia. *ACM SIGCSE Bulletin*, 40(2):97–101, 2008.

[FB15] Tobias Fertig and Peter Braun. Model-driven testing of restful apis. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1497–1502, 2015.

[G+05] Jesse James Garrett et al. Ajax: A new approach to web applications. 2005.

[GAB22] Sithara HPW Gamage, Jennifer R Ayres, and Monica B Behrend. A systematic review on trends in using moodle for teaching and learning. *International journal of STEM education*, 9(1):9, 2022.

[JDV+09] Gideon Juve, Ewa Deelman, Karan Vahi, Gaurang Mehta, Bruce Berriman, Benjamin P Berman, and Phil Maechling. Scientific workflow applications on amazon ec2. In *2009 5th IEEE international conference on e-science workshops*, pages 59–66. IEEE, 2009.

[Lap10] Phillip A Laplante. *Encyclopedia of Software Engineering Three-Volume Set (Print)*. Auerbach Publications, 2010.

[MAM+17] Lauren Murphy, Tosin Alliyu, Andrew Macvean, Mary Beth Kery, and Brad A Myers. Preliminary analysis of rest api style guidelines. *Ann Arbor*, 1001(48109):4, 2017.

[MGJS23] Joshua Martius, Sergej Görzen, Sven Judel, and Ulrik Schroeder. *Weiterentwicklung eines Dashboards zur digitalen Betreuung wissenschaftlicher Schreibprozesse*. Gesellschaft für Informatik eV, 2023.

[UJG+22] Dieter Uckelmann, Sven Judel, Sergej Görzen, Christoph Greven, and Ulrik Schroeder. Structured digital writing lab-workflow, application and evaluation. *International Journal of Online and Biomedical Engineering (iJOE)*, 18(14):133–146, 2022.

Appendix B Digital Appendix

The digital appendix is submitted on a SD card. To get access to this material they shall contact the author or LuFG i9.