DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

GBS Tutoring Tool: A Software Engineered Websystem for the Creation and Organization of GBS Exercises.

Nicolas Mario Arteaga Garcia

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

GBS Tutoring Tool: A Software Engineered Websystem for the Creation and Organization of GBS Exercises.

GBS Tutoring Tool: Ein Software-entwickeltes Websystem für die Erstellung und Organisation von GBS Übungen.

Author:	Nicolas Mario Arteaga Garcia
Supervisor:	Prof. Dr.-Ing. Jörg Ott
Advisor:	Dipl.-Inf. Martin Uhl
Submission Date:	15. March 2021

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15. March 2021 Nicolas Mario Arteaga Garcia

Acknowledgments

This thesis was completed as part of a five-months research project at the Technical University of Munich.

First and foremost, I would like to thank Prof. Dr.-Ing. Jörg Ott for offering me the chance to conduct my Bachelor thesis in his group. His teaching in GBS made me discover my interest in the Unix system and virtualization techniques.

An equal amount of gratitude must go to my supervisor, Dipl.-Inf. Martin Uhl for the introduction to this topic, and his constant knowledge input, and support for my work.

I would also like to thank the organizational team and tutors from the GBS lecture at the Technical University of Munich, who participated in this software project and research. Without their time brainstorming and giving feedback to the different phases of the system, this thesis would not have been possible.

I am grateful for the warm welcome and positive work climate that I experienced in the group of GBS. The helpfulness among colleagues, their support and assistance were pivotal during my bachelor's thesis undertaking.

Finally, I would like to thank the future developers that continue this project.

Abstract

Due to an increasing number of students studying computer science, some courses at the Technical University of Munich (TUM), were confronted with more than thousand students registered. Together with the ongoing Coronavirus pandemic, this introduced the need for more sophisticated and easier to use online learning tools to guarantee a high quality learning experience. One particular system fulfilling these requirements is an exercise management system, which facilitates the creation of new exercises and helps to organize these.

This thesis presents the establishment of a software system for the creation and organization of exercises for the university lecture "Grundlagen Betriebssysteme und Systemsoftware" (GBS). Old systems are substituted or upgraded with extra functionality and organizators receive enhanced tools to improve and facilitate teaching. New techniques to create a more interactive learning are proposed, researched and implemented. The presented synchronization approach is based on requests and analysis of feedback from different actors. Since the project is developed by a single person for many clients, a strategy for collecting and refining this information is proposed.

We conducted a literature review on software engineering methods and formalized requirements and constraints for defining the data of the lecture in exercise objects. Furthermore, we design a system and evaluate the prototypical implementation's applicability in a study with 15 participants.

Contents

1 Introduction

This thesis introduces the software engineered project "GBS Tutoring Tool", starting with the planning and elicitation of the requirements, continuing with the modeling and design of the project, advancing to the implementation and deployment of the system and finishing with a conclusion.

1.1 Problem

At the Technical University of Munich (TUM) there is a lecture called "Grundlagen Betriebssysteme und Systemsoftware" (GBS), that had about 1000 students enrolled in the past winter semester 2020/2021[1]. This number is planned to increase with the continuous growth of the number of enrolled students at the TUM, and most importantly the faculty of Informatics taking into account that GBS is not only a course for the Bachelor of Computer Science and business Informatics, but is greatly chosen in other big TUM degrees as for example TUM-BWL.

The GBS lecturer is Prof. Dr.-Ing. Jörg Ott and the exercise leader and supervisor of this thesis is Dipl.-Inf. Martin Uhl. The lecture organization team consists of only 5 lecture organizers named "GBS Kern", where three of them are part of the 19 tutors. In total there are only 24 persons involved in the maintenance of a big course such as GBS, which shows that organizers need good tools for qualitative teaching.

Good tools are even more important when using an online teaching strategy as they influence the success of online learning as seen in the article "Improving online learning: Student perceptions of useful and challenging characteristics" [Son+04]. In TUM courses like GBS, there are tools like Moodle (a course management system) that enable a communication and the publication of material. But these tools offen do not fit the extra needs of the courses.

For instance, there is a need for the TUM to create other tools like the Automatic "Assessment Management System for Interactive Learning" named Artemis, which "assesses solutions to programming exercises automatically and provides instant feedback so that students can iteratively solve the exercise"[KS18]. These tools bring extra functionality or the possibility of a different visualization of course material (e.g. the listing of course material from GBS at "https://gbs.cm.in.tum.de"), which provide a better overview for both students and organizers.

At the moment, GBS has no internet portal that makes it possible to create of exercises and organize the lecture online in a scalable way, making it easy to be improved in the years to

[1]Data from the GBS Course supervisor

come. This leaves the possibility for a new web system, that can help to improve the teaching (for example The GBS Tutoring Tool), to be created.

1.2 Purpose of the System

The GBS Tutoring Tool makes the creation of LaTEX exercises for the GBS Lecture easier and more interactive. It gives the possibility for tutors to make their own exercises in a reliable way, without having to download any LaTEX libraries or knowing how to write the files in LaTex. It has online access, where every tutor will have an account.

This account gives access to their own profile with their GBS exercises ideas and the pool of all exercises from the lecture. The main users will be the tutor leader and the tutors, but in future plans it is an idea also to have students as users. One main goal is to facilitate the creation of exercises for the lecture.

They will not need to master LaTEX in order to write them. When creating a new exercise, they get a LaTEX template, that makes clear where to write the title, problem question or add images, if needed. They will not need to import any libraries or download the right compiler to create the exercises, since everything will be taken care of from the system. Editing an exercise when an error is found or something is badly explained is just as easy as clicking the "edit" button in the exercise box and changing the typo or adding some additional explanation.

One purpose of this thesis is explaining the system interactions with other systems, such as the web view, where the users interacts, the TUM server from where we retrieve their data, and the old used system. Because one requirement of the system is it being used with the old "GitLab" environment. So that the users still have the option to make changes offline and synchronize them with the Web server via git commands. This research will try to find the best approach for synchronizing the web server with the git system.

There is also the idea of a future integration of Tumonline and Moodle. Here the students registered for the course will get an account in the system and be able to see their own bonus results from Moodle exercises, like the Quizzes, or the programming homeworks.

1.3 Models

It is important to note, that we mainly used the help of concepts from the book "Object-Oriented Software Engineering Using UML, Patterns, and Java" from Prof. Bruegge [BD13] to create a good system with a solid structure and good modelling techniques.

Through this thesis we will present the different unified modeling language (UML) models used, while building the system, in order to describe it better. We use models to abstract and represent the important aspects of the systems. They help developers deal with the complexity of building the system and dealing with its problems.

The first models we will present to describe the system are some visionary scenarios, which will help us bridge the conceptual gap between the developers and the end users (the tutors and students) of the system.

1.3.1 Visionary Scenarios

- The Tutor Leader Martin logs in his account from the GBS Tutoring Tool and enters his LRZ/RBG login credentials. Then he goes to the "My Profile" Tab and creates a new exercise for chapter 5. Then Martin saves the exercise to his personal profile, where different personal exercises can be edited or uploaded to the common pool of exercises.
- Hendrik had an idea for a cool exercise to test their students about the "virtuelle Speicherverwaltung". He thinks it could be beneficial for other students too, so he creates it in GBS Tutoring Tool, and clicks on "load to exercise pool". If the exercise is good, it will be used in the future to create exercise pages or even exams.
- Andrea is a tutor that doesn't think that the students from her class understood how "Petri-Netze" work. She wants them to understand it better, so she logs into the GBS Tutoring Tool, chooses some exercises from Petri Netze and creates an extra exercises sheet for them to practice at home.

1.4 Strategy

Before starting the project, I had BBB[2] meetings with the GBS tutorial instructor (who also supervises this bachelor thesis). In those talks we brainstormed and came up with some possible requirements for the project's system[2.1], but since this project is open to ideas, we decided that the ones that would use the system in the future (tutors) should take part in the planning. That's why we created a new strategy, where the developer would work with many clients and collected all possible ideas, analyzed them and figure the steps for the fulfillment of the first version of the GBS Tutoring Tool.

This new strategy consists in searching for ideas of tutors and organizers, analyzing and evaluating the ideas and rebasing them in new final requirements. This project involves a one-to-many relationship between the developer and the clients, which is very uncommon in nowadays software projects. It is normal that software projects change a lot. The general approach is having a team of developers with different abilities solving a software project of one client. Sometimes, the clients are two to three customers from a company, but even then they communicate beforehand and have a general idea of what they want or what the leader of the group wants. For our project, we had to communicate with the GBS Kern for system related organizational ideas.

In order to have some general ideas for the improvement of the teaching and focusing on the student's point of view, we had to involve as many GBS tutors as possible.

In our specific case, there is no known research about and therefore we needed to reinvent the way of building the project. In practice, the developers of software project use a Software Development Process Life Cycle (SDLC) to produce high-quality software, that meets the client's expectations and reach its completion on time.

[2]BigBlueButton: A free software web conferencing system for GNU/Linux servers.

1.4.1 The Software Development Process Life Cycle

In general, the main stages of a software project are: Planning, Requirements Analysis, Design, Implementation, Testing, Deployment and Future (Maintenance/improvement).

The most classical SDLC is the Waterfall Model (see figure [1.1]), it was proposed in 1970 and it is a lineal sequence of all the main 7 stages that software projects have [BD13]. When a stage is finished, it will not be repeated again. It is simple to understand and it's effective with strict deadlines. But it's not really flexible since problems remain uncovered until testing and it's difficult to make changes since so much from the project is fixed.

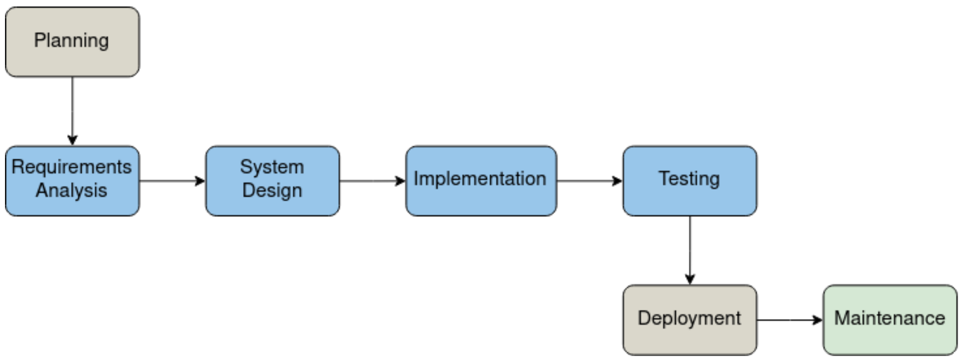


Figure 1.1: Schematic representation of the main seven stages with the Waterfall Model.

Nowadays the Iterative Life Cycle Model (see figure [1.2]) is the most used variant [BD13]. Due to its flexibility, it works well with developer teams and can be easily adjusted to fit their goals. Famous adjusted variants of the general Iterative SDLC Model are Agile or Scrum. It stengths lies is the better work efficiency and contributes to better feedback to the team manager. Although no stage is finished it can be difficult to set the milestones.

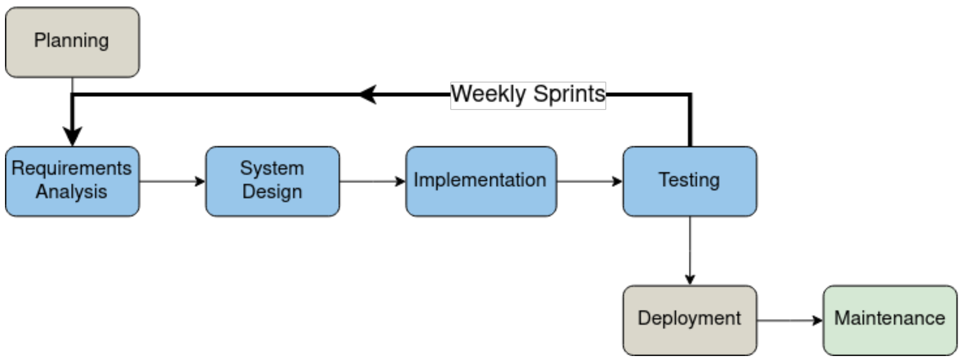


Figure 1.2: Schematic representation of the main seven stages with the Iterative Model.

In this thesis, a combination of both Waterfall and Iterative Models are adjusted to fit the requirements and the deadline. The planning of the project with the requirements elicitation and analysis was made in one block, comparable to the waterfall model. The planning and the requirements analysis was completed in one and a half months (from 15.10.2020 to 31.11.2020).

In this time we analyzed all the ideas and researched how the system should be build. (**Part 1** in figure [1.3]).

In the following months (1.12.2020-15.01.2021) weekly sprints were the System Design, Implementation and Testing were improved and brought to the desired state. (**Part 2** in figure [1.3])

Finally, two weeks were planned for the Deployment (16.01.2021-01.02.2021). Afterwards, the tutors and GBS Kern were asked to give feedback. Some of it was used to improve the system (16.02.2021-15.03.2021). On basis of this, a conclusion was written and future work for this system was documented. Hence, a basis system was build to be maintained and upgraded easily. (**Part 3** in figure [1.3])

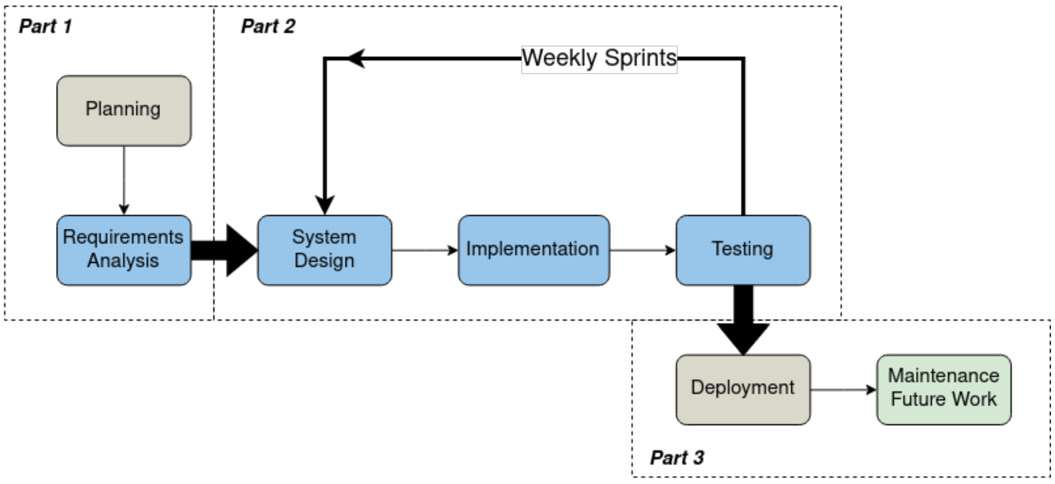


Figure 1.3: The GBS Tutoring Tool SDLC Model

2 Requirements Elicitation

2.1 Requirements Ideas

Before starting the project, some requirements ideas were conceptualized. In section [2.3] the final requirements are described.

Functional Requirements:

Admin:

- Can write new or edit existing LaTEX exercises.
- Accepts and administrates the users.

Tutor Leader and Tutors:

- Can write new and edit existing LaTEX exercises.
- Can actualize or create exercises locally and push them with git.

Non Functional Requirements:

- System should convert LaTEX text fast and uncomplicatedly.
- Every exercise should be combined with its solution.
- A user should be able to create an exercise sheet in less than 10 steps / 5 minutes.
- System should be well documented and easy to expand (It should be easy for the next student or developer to change functionality).

Optional Requirements:

Functional:

- Admin can upload students results in CSV from Moodle to the Webpage.
- Students can access the website to view their results and own homework feedback.
- Students can create own exercises for future semesters.

Non-Functional:

- System is fully in English or German.
- System actualizes the students results data on Moodle automatically.
- System changes exercises randomly for the exam of every student.

2.2 Redefining the Requirements

After the planning was conducted, the second stage was reached. The requirements elicitation and analysis were still open to change and had to be refined in a new way. A more empirical approach was taken in the way of getting the ideas.

The most essential part were the individual BBB conferences planned with the lecture organization team members (GBS Kern). Usually the GBS Kern are organizers for the lecture, which study at the TUM. Important insights were gained at the conferences, because the members have experience using the old system. Therefore, the best way of connecting the old (GitLab) and the new (web) system were analyzed. According to the study "Comparing focus groups and individual interviews: findings from a randomized study" [Gue+17], the meetings were done separatly to ensure the members were not influenced by each other and could speak freely.

The project was briefly explained in a mutual meeting. We told the tutors to collect some ideas through one week. After that, we started individual meetings initiated with the same three questions about their experience with GBS and what they think about new websystem. All these five conferences where recorded with the permission of the GBS Kern and can be watched upon request.

In the other approach I went to the weekly tutors meeting and introduced the system idea, then created a voluntary formular for all tutors.

After some research it came clear that the most fitting option was creating the survey with Google Forms[1]. The tutors had the possibility to use an arbitrary pseudonym, to remain anonymous, since the data passes through the Google servers.

We reviewed the google forms survey tool with the help of the proceedings "Online survey tools: A case study of Google Forms" [VN16]. The tutors could voluntarily answer some general questions to later be able to classify their ideas by different variables such as if they were a bachelor's or master's student [1] or if they were GBS tutors for the first time [2]. Then it went to some single choice questions of the system and in important questions, where the tutors could write and deepen in their ideas.

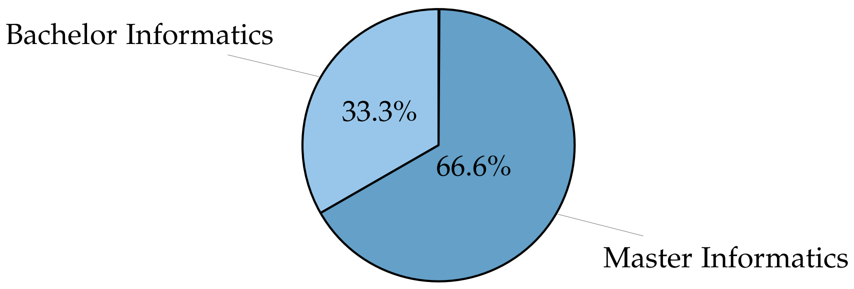
This data is in german and anonymized and can be seen as .CSV files exported from Google Forms upon request.

[1]Google Forms: A web-based app used to create forms for data collection purposes.

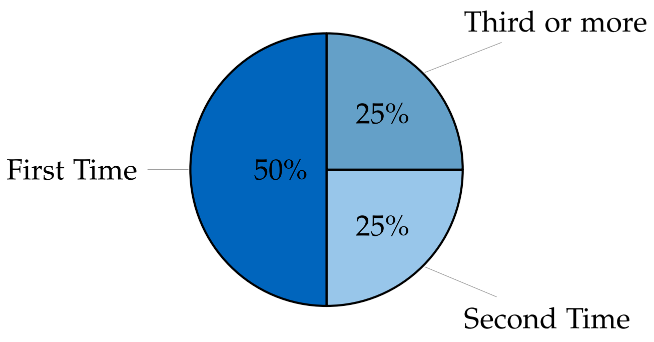
2.2.1 Formulary with Tutors ideas

12 out of the 21 GBS tutors, in the winter semester 2020/2021, conducted the voluntary survey. Four of them, also where part of the GBS Kern. In the following the received answers are shown:

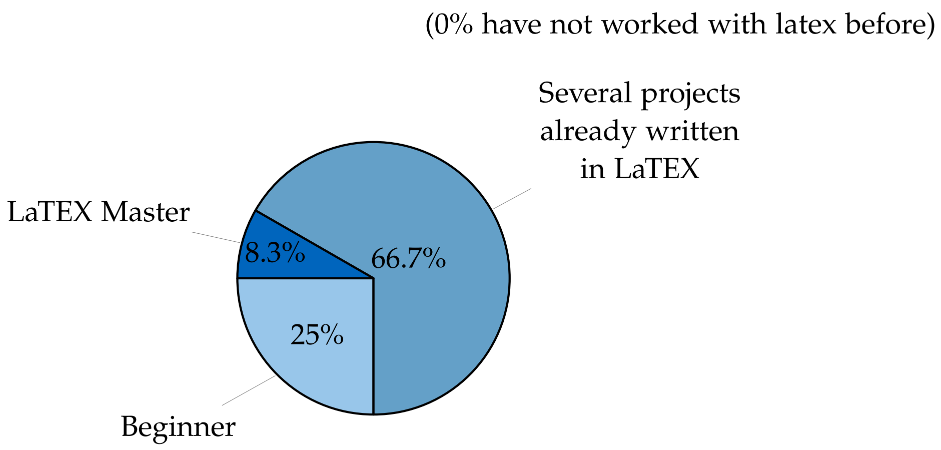
1. **Are you a Bachelor or a Master student?**



2. **How many times have you been a GBS Tutor?**

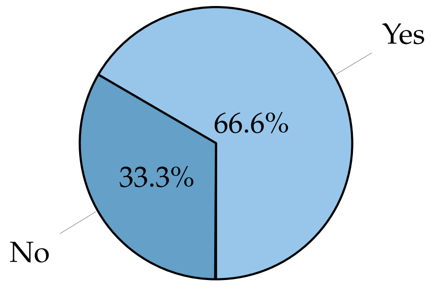


3. **How are your skills in LaTEX?**

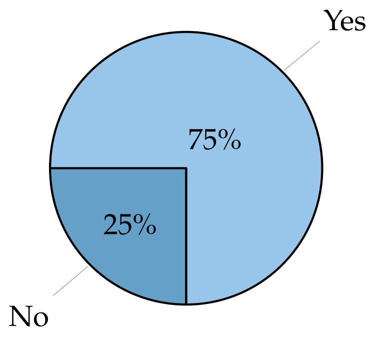


4. **How do you find the idea of making a web app for the GBS tutoring?**
   Positive Feedback:
   "Good", "I think it is great","Interesting", "Generally good, because there are many processes that can be automated", "Good, but still C++ is a prettier language", "Sounds very exciting and practical", "Really good. Having quick access to practice exercises would be helpful for many students and is something that many courses lack.",
   "The option sounds really good, depending on who is supposed to use all of this, another platform prevents a lot of data.", "Excellent".
   Critical Feedback:
   "In principle, it is good if it can replace other tools. Using more and more tools will otherwise become confusing."
   "Not really useful - Artemis and Matrix already exist, so that would be another platform."
   "I'm not sure how necessary something like this is, because it doesn't happen that often, that we create new exercises as a tutor and I can do them relatively quickly with LaTEX."

5. **Can you imagine creating new GBS exercises on the WebApp?**



6. **Do you think that students will send GBS exercises suggestions on the web application?**



7. **Do you have any suggestions for the WebApp for organizing GBS exercises? (Ideas that you would want to have)**
   a) Possibility for a tutor to upload their own slides/PDFs and to make them available to the students. I think this would automatically attract the students.

b) I had actually a similar idea in which students could, for example, randomly generate tasks such as scheduling or buddy alg., etc. so that they would be able to practice a lot of "algorithmic" problems/ tasks quickly.

c) It would be really cool to have the possibility of interactive problems (e.g. for Petri nets), which students can play around with and see for themselves ow their solution plays out.

d) Statistics (which problems are most popular, which tasks are completed (and how well), etc.).

e) For example, graphical compilation of exercise sheets from the (existent) collection of exercises/problems, and, if applicable, (prefabricated) feedback options for students ("this problem was confusing/ difficult", "this problem was doable", . . .).

f) For me, creating the problems using a classic LaTex editor is enough. The greatest benefit for me in the implementation as a WebApp would be the integration of CI/CD, so that material is automatically updated in, e.g., Artemis and Moodle without having to manually upload everything again and again.

g) Fluent design, and drag & drop for the arrangement of the problems.

2.2.2 BBB conferences with GBS Kern:

Although most of the members had a special view with ideas, we repeated the most important topics often to try to find more input. Topic repetitions are ignored in this summary. This are the summarized most important outputs:

- **Christian Menges:** (Meeting on the 29/10/2020, 3x GBS as tutor & organizer)

  He would like to see something being done in the course continuous integration and continuous delivery from the course material to the students. Things can be automated between the three GitLab repositories from GBS. For example, that the lecture script gets compiled automatically and uploaded to the moodle course site every week before the lecture. We found this idea from Mr. Menges really valuable, but after debating with the tutor leader it became clear, that it is not covered by this thesis content. But it could be accomplished with a small script.

- **Fabian Sauter:** (Meeting on the 30/10/2020, 3x GBS as tutor & organizer)

  Mr. Sauter suggested to implement a Drag and Drop for the creation of exercise sheets and told that we should use "Fluent Design"[2] for the Website design. After reviewing this idea with the tutor leader, we rejected it, because the GBS Tutoring Tool should try to be part of the TUM Ecosystem. This is why the web system design is based on the TUM Corporate Design[3].

[2]Fluent Design: Some Microsoft guidelines for the software design of Windows platforms.
[3]TUM Corporate Design: Living Styleguide for the design of digital platforms for the university.

- **Sebastian Kappes:** (Meeting on the 01/11/2020, 2x GBS as tutor & organizer)

  Mr. Kappes most important point was to implement a way to make the exercises more organizable. We brainstormed and came to the ideas of making the exercises organizable looking at the The exercises could have tags in the LaTEX file, that could be visualized as headers for every exercise. For example, for tutoring problems, exam problems, and extra properties like, if it is used in an exam or the time needed for completion. How this advanced can be read in the "Problems Header" section.

- **Leander Seidlitz:** (Meeting on the 02/11/2020, 4x GBS as tutor & organizer)

  Mr. Seidlitz told us that we had to concentrate in what functionality we can give to the students, because the GitLab system is almost only used by the GBS Kern, and they already dominate it. The most value from creating the GBS Tutoring Tool would be to make it easier for the students to interact with GBS. And we agreed that the system does not want to substitute GitLab but work with it and add functionality. Mr. Seidlitz proposed programming the the WebApp system with Python and using the Django or Flask web framework. We go into this in the section "Programming Language".

- **Clemens Horn:** (Meeting on the 06/11/2020, 4x GBS as tutor & organizer)

  Mr. Horn recommended to use well known and tested modules for the web system, by doing a user login system with the TUM Shibboleth and the LRZ account or with the Informatics RBG account.

2.3 Final Requirements

In the data it is easily noted that 50% of the teams are a GBS tutor for the first time, and they did not really have new ideas to work on the system. They found the idea good and were positive about, the system having a possibility in helping at the GBS teaching, but they did not know what to add.

With all this input I could identify some general requirements that were important for the GBS Kern, the tutors or the tutor leader. This is the final version of the functional requirements (short FR) and the non functional requirements (short NFR) for the GBS Tutoring Tool based on the FURPS+ model from [BD13, page 120]. FURPS+ is an acronym using the first letter of the requirements categories: Functionality, Usability, Reliability, Performance and Supportability. The + indicates the additional subcategories.

2.3.1 Functional Requirements

Functional requirements express the functions that the system has to accomplish at the end state. They do not depend on the implementation, and should be accomplished on every plattform [BD13]. The final FRs are:

FR1 **Create Exercise:** Users can create new exercises and view the compiled file in the websystem.

FR2 **Correct Exercise:** Tutors can correct the students problem ideas.

FR3 **Synchronization with GitLab:** The web system actualizes the new problems and changes with the GitLab system. The users should be able to interact with the implemented properties of the web system, in the GitLab subsystem.

FR4 **List GBS Material:** Students can view and download their material from GBS.

FR5 **Search Exercises:** Exercises can be found with a search bar.

FR6 **Create Exercise Sheets:** Users can make exercise sheets and view the compiled file in the websystem.

FR7 **Automatic creation of Exercise Sheets:** System creates exercise sheets based on time constraints and chapters elected.

2.3.2 Non Functional Requirements

Non functional requirements describe the quality requirements from the system, and focuses on its usability, reliability, performance and supportability [BD13]. The final NFRs are:

Usability

NFR1 **Ease of use:** The user data actualizes automatically from the TUM RBG LDAP Server.

NFR2 **Ease of use:** Admin can change in an easy way which users have access to the websystem.

NFR3 **Ease of use:** The system uses the TUM Corporate Design to improve readability.

Reliability

NFR4 **Safety:** The system should check every login with the with TUM LDAP Server and not store the password for extra security.

NFR5 **Robustness:** The system should use well tested modules for authentication & security.

Performance

NFR6 **Performance:** An Exercise should compile and be shown in less than 5 seconds.

NFR7 **Performance:** The system should actualize all the material in less than 10 seconds.

Supportability

NFR8 **Maintainability:** System should be well documented and easy to expand. (It should be easy for the next developer to change functionality)

2.4 The Use Case Model

In Figure [2.1] a use case model is presented that focuses on the part of the future system that handles the creation and evaluation of exercises ideas. Use case models describe a set of scenarios from actors interacting with the system.

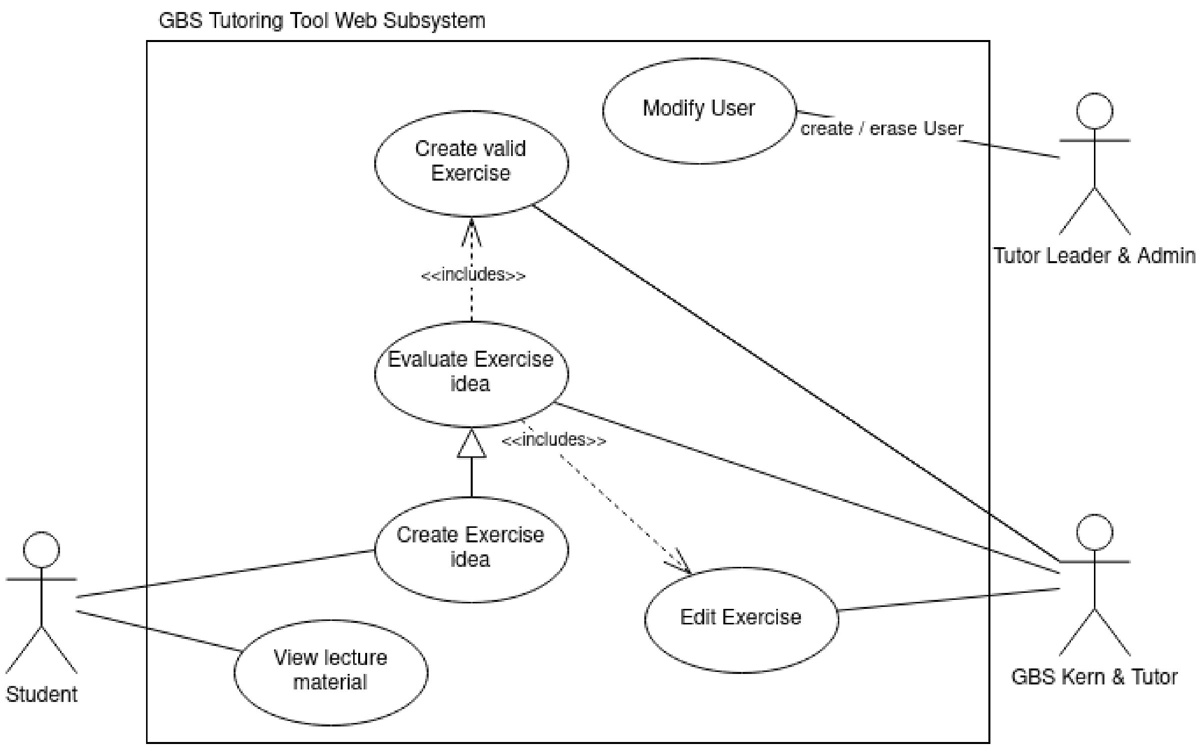


Figure 2.1: Use Case Model. A student creates an exercise idea, that gets edited from a GBS Kern member and changes to a valid usable problem.

3 Analysis

This chapters concentrates on finding the best option for building the GBS Tutoring Tool. We are building a system that has the GitLab system interacting with the new web system. We will now decompose the actual system to be able to understand it better and discuss where and how to establish the connections between both subsystems.

3.1 Decomposition of the Git System

The GBS lecture team uses three Git repositories to organize and manage their lecture documents. Simply explained, a Git repository is a folder with the git version control tool used to navigate in the different changes that this folder had over time and facilitates working in a team and synchronizing the work. We will go more into the architecture of git here at the section 3.6.

The "gbs-tutorials" repositories is where all the problems and files used in the lectures are stored. "Problems" are how the GBS Kern defines the exercises used in the lecture. These files and problems are mostly LaTEX documents that compile with a Makefile. Briefly, LaTEX is a system of text composition, used to the creation of high quality scientific documents. A Makefile is a file used to automate the compilation and create an objective. In this case the Makefile is a complex 400 lines long file, that was inherited from the chair of networks lectures organization. It is used to compile the tutoring PDFs from the LaTEX documents used in the course in different forms. It chooses the different sheet structures from the predetermined LaTEX documents, for example, when executing the make commands in the terminal. The three most important variants will compile and build a PDF for every LaTEX sheet in the directory, but with different outputs.

- The PDF will display its problems in a standarized way.

  ```
  make tutorial01
  ```

- If existing in the LaTEX document. After every problem or subproblem the solution for that part will be showed.

  ```
  make tutorial01-solution.pdf
  ```

- After every problem or subproblem it leaves a blank space for the students to work on when practicing on a sheet.

```
make tutorial01-presentation.pdf
```

While studying the Makefile we came to the realization that a Makefile is prone to change in the future as GBS changes the form of compiling their sheets or the commands the GBS Kern will use. This could be the perfect bridge between the Git and the web system. It is based on the fact that both systems work with the same Makefile, so that the scheme of compiled sheets has only to be changed in the repository for it to take effect on the web system. As seen in [BD13, p.224-228] a good system design should have low coupling, which means that a change in one subsystem should not affect any other subsystem. It should have high cohesion, which refers to having classes that perform similar tasks and have many associations. We can decouple the whole system of GitLab and the GBS Tutoring Tool if we use the same Makefile for both systems.

For example, when the GBS Kern changes the packages for compiling LaTEX docs or chooses to use "biber" or "pdflatex" to compile, there should only have to be one change in the git repository for it to effect on both subsystems. This applies when changing the individual problem sheet used by the web system or an exercises sheet. A change in the git repository takes effect both on GitLab and the new web system.

The second git repository is "Material", that it is used to give place to all the course material for the students. The students either save a local copy of the repository on their laptop, or they can visit the website gbs.cm.in.tum.de which has a directory listing of this repository.

Finally, the last git repository is "Tutorial", which is used by the tutors to coordinate their organization in the course and has the compiled sheets and material that are not released for students. This repository is mostly used for rescheduling tutorial hours. It is planned to be substituted by another subsystem for the GBS Tutoring Tool. This can be a future thesis.

We only need to connect our system with the two first repositories, since with "gbs-tutorials" we can compile all sheets in the different formats. We will use the data inputted in the "Material" repository to make it accessible for students it in the web system. Although we decouple the system by working with the same Makefile that will adapt to the changes, the subsystem still has to know the folder structure from the problems and where to find the Makefile. For the web system to work correctly with GitLab, there has to be a fixed structure and naming of the folders.

Figure [3.1] is the structure of the "gbs-tutorials" Folder.

3.2 Base Problem Structure

While analyzing all the problems used in the last years of GBS, we saw that the problems change a lot based on the different ideas the creators had. The simplest problems only had a question text and solution, but the most complex ones also included different graphs written in LaTEX or solutions only for the subproblems like in figure [3.2].

This lead to difficulties when designing the problem, so our goal is to find how to abstract the problems. After brainstorming, it was clear that representing all different problem

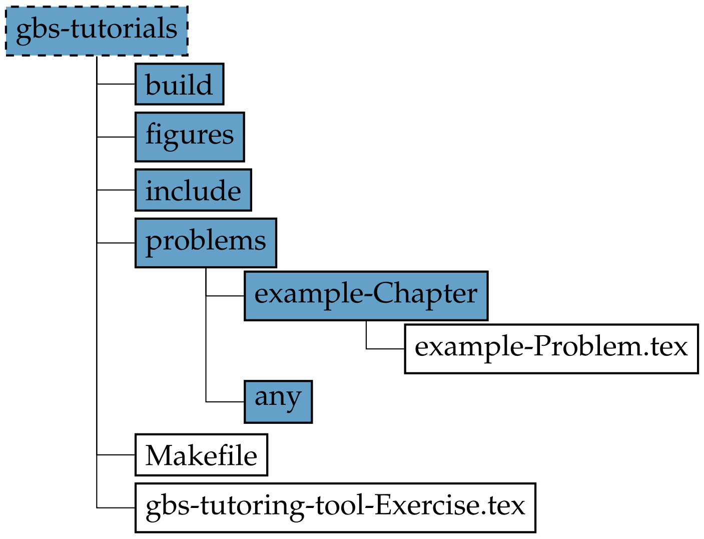


Figure 3.1: The "gbs-tutorials" git folder. White boxes represent files and blue boxes folders.

```
\problem{Scheduling (Klausuraufgabe)}
\vspace{\baselineskip}

\subproblem* %GO \ifsolution \else Die Programmiersprache Go
...
\begin{tabular}{p{10mm}|*{5}{p{\colwidth}|}|*{5}{p{
        \colwidth}|}|*{5}{p{\colwidth}|}|*{5}{p{
        \colwidth}|}|*{5}{p{\colwidth}|}|*{5}{p{\colwidth}|}|*
...
```

Figure 3.2: Problem "scheduling4" from chapter "Prozesse" with approximately 237 lines of tables.

examples would not be possible, owed to the fact that our subsystem would not substitute the Gitlab system. For this reason, we came up with a structure that describes a basic problem in a LaTEX document. This structure is separated in three big sections:

1. The header, which will be explained in the next section [3.3].
2. The problem, which begins with "\problem{}", stops when another section starts and stores everything that wasn't already contemplated in the header or in the subproblem. It stores, for example, the title, the problem question, its solution, all graphs used to explain the exercise better, and all comments or extra arguments that are not inside the other sections.
3. If existing, the subproblem or subproblems. Beginning with "\suproblem" , where the different subquestion are asked for the explanation of the " \problem{}".

We came up with this model of a basis structure file in figure [3.3].

```
-------- HEADER --------

-------- PROBLEM --------
\problem{This is the Title of a Problem} This is the input of
the problem question and etc...

-------- SUBPROBLEM(S) --------
\subproblem
This is the input of the subproblem.
```

Figure 3.3: Structure of a problem file, consisting of header, problem and subproblem.

3.3 Problems Header

Analyzing the system, we came up with a way of decoupling both systems by adding functionality that can be changed from either one of the subsystems. This would consist of adding the properties for the problems commented in their first lines, in a so-called "Header".

The header could enable an input bridge from the problem properties between the web app and the git system, without really changing the workflow of the users of the git system. It enables the git repository users to change the attributes in the commented header, at the start of the file without having to compile anything. The users in the web system just have to change the attributes of the problem and save the instance.

The header will store the different attributes of the problem. These attributes are, for example, the keywords bonded to that problem, so that it is easier to find problems that have key aspects in common. Keywords examples for the GBS problems are: "scheduling, round robin, FCFS, clock, semaphore, mutex, petrinetze, gantt, bitmap, buddy, clock, paging, pagefault, segmente, ext4, inodes, rights, etc...".

Another attribute, if used in an exam, stores in which exam that problem has being used. This comes very handy for differentiating which problems can be used in future exams, or which problems the tutors can give for practicing, that are not going to be used in the exercise sheets.

The last attribute is the approximate time needed for completing the problem. It aims at creating the possibility of making random exercise sheets that are less or equivalent to the time specified from a student wanting to practice.

Other property ideas were revised, but the use they gave was mostly overshadowed by the complexity they introduced. For example, there was an idea of an attribute that told for every term if the problem was already being used in an exercise sheet that semester, and if so, in which one. The possible benefits from the functionality were overshadowed from the work of having to change the different exercises files every week in the git repository.

There were two possibilities of how the header should be written in the .tex files. The first one was to write the header as comments at the start of the file and then use pattern matching to save the attributes in the system. The second one was to rewrite the defined LaTEX "\problem{}" macro and create an input from the attributes like this:

```
\problem[oldExam-input][keywords-input][aproxTime-input]{problemTitle}
This is a problem example...
```

We discussed both possibilities with the GBS Kern and came to the conclusion, that since the web system concentrated a lot in pattern matching to retrieve all data from the LaTEX documents (Problem Title, text, solution, etc...). The implementation with the most sense would be the first one, writing the attributes as comments and getting the properties with pattern matching.

It should be programmed in a way that a header that isn't complete still can be saved and does not disrupt the process of creating new problems. This will require the subsystem to have techniques of dealing with non compilable LaTEX documents. The pattern matching for the header starts when the system detected a start of a line with the special set of characters "%#%". This set works as a comment in LaTEX documents because of the % start, and it draws the attention on the beginning of the page, clearifying where the properties of the problem are saved. After that, it just needed to follow the line to find from which attribute it worked and store the data written in the line.

We can find an example of a header in a problem file in the figure [3.4].

3.4 Programming Language

The programming language chosen for implementing this system was Python. Python is one of the most fast growing programming languages in the world since 2018 [Jou18]. It is very beginner friendly, since comparing it to other programming languages such as Java or C++

```
%#% OldExam: Wi 17-18 Retake / Wi 18-19 Endterm / unused Wi
%#% Keywords: semaphore, mutex, petrinetze
%#% AproxTime: long==30 min, medium==20 min, short==10 min

\problem{Petrinetze: Basics}
Während einer Klausur sind die beiden Hörsäle MW2001 und MW0001
parallel in Benutzung. Die beiden Hörsäle teilen sich jedoch die
Toilette. Somit darf beide Hörsälen insgesamt immer nur eine Person
verlassen. Hierfür wird ein sogenanntes ``Toiletten-Mutex''
benutzt. Die Anfangsbelegung der Hörsäle entspricht den
Prüfungsplätzen, 187 für MW2001 und 127 für MW0001.

Modellieren sie das Szenario und dessen Synchronisation mit einem
Petrinetz. Gehen Sie wie folgt vor:

\subproblem*
Modellieren Sie das grundlegende Szenario mittels Stellen.
Verwenden Sie je eine Stelle für die beiden Hörsäle sowie zwei
Stellen für die Toilette. Dies ist notwendig, damit Studenten nur
in den Hörsaal zurückkehren können, den sie verlassen haben.
```

Figure 3.4: Header problem example for the "petrinetze3.tex" file.

you can write the same programm with less lines of code. This makes the system code easily readable and lowers the entry barriers for new developers contributing for this system. It's a high level language so we do not have to worry about memory allocation like in C++. Being a cross-platform, works on all different operating systems such as MacOs and Linux. This creates a big community of developers and plenty of documentation which be found online [Fou20a].

Comparing pythons performance in a web framework related to other languages, this paper sums up the performance in several benchmarks . As found in [Rog17]: "The conclusion is that the choice of the web framework is of secondary importance. In other words, any given framework is not the ultimate blocker in the development path. The most important thing in the decision process is to take into account programming language preferences, the size of the community and a job market."

Python being used in the web framework, allows to write the front- and backend of the application with the same language (Python), which is another plus for simplicity for the developers. We will go more into the libraries we used in the System Design and Implementation on chapter [4].

3.4.1 Python Web Application Framework

For this project, different web application frameworks came in question for the realization of this web system. Important aspects, that the frameworks had to fulfil, where to be open source, and WSGi compatible. As explained in the glossary, the Web Server Gateway Interface (WSGI), is a specification used to describe how a web server communicates with web applications, and how work together to process one request.

Going into all web frameworks would need a lot of time, and is not really useful, since most important properties of the web frameworks repeated itself. So we will divide them into the two big branches, the full-stack frameworks and the microframeworks, and we will propose the two main competitors, Django and Flask, and why we chose one.

The key characteristics of the full-stack framework Django [Fou20b] are, that it has a really quick learning curve, which enables to develop systems in a fast manner. Additionally, authentication and several other functionalities are already shipped with the framework.

Thus "Django is used as the go-to Web Application framework, when comparing the productivity, features, security, performance and scale flexibility with other programming languages" [Ben20].

On the other hand, the microframework Flask started in 2004 by Armin Ronacher is a really lightweight web framework that uses the "Werkzeug" WSGi and the "Jinja2" Templating Engine [Ron20]. It has no need of downloading extra tools of libraries, but is easily extended to enhance features.

Considering the limited time and the lack of much experience in web development, the best choice for this project was Django, also because of its thorough documentation.

[Bha20] on page [32]: "Based on the study, it is evident that Django can be best fit for large-scale projects with the cost of the learning curve. Flaskisbest fit for the prototyping and small-scale projects but not limited to it. Flask can be learned and set up quickly, but when it comes to managing and maintaining, it requires more work than the former.".

3.4.2 Database

As seen in [KH09], it is recommended to use Django both in the front- and backend. Another suggestion is to use sqlite as a database as there are fairly good python bindings available. Our system design follows these advise and therefore uses Django Templating in the frontend and a sqlite database in the backend. A downside of this design choice is that sqlite does not offer the possibility of ArrayFields. We therefore had to adapt our Analysis Object Model to this restriction.

3.4.3 Virtual Environment

It is recommended to always use a virtual environment for the deployment of the project, since it makes easy to ensure for the program to use the same packages, and its versions as the ones used in production. Hence, while choosing a virtual environment for small software project like this, that does not use many libraries, we learned from the book [Ben20], that for small software projects like ours, is virtualenv more than enough.

Virtualenv is a lightweight python environment, that downloads the needed packages in a folder and makes python take the libraries of the given folder instead of the ones that are available on the whole system. It makes it really easy to develop different projects and be platform independent.

3.5 Analysis Object Model

As described in the "Base Problem Structure" [3.2] modeling the different problem files is a difficult task. Trying to specify the objects much would not work, because the problems structure changes a lot from chapter to chapter. We want a system that makes the creation of problems not only easier, but more adaptable to the different ideas from students. It should have the option of making a simple problem object with "header, question and subproblem" in their respective input panels.

Based on the analysis, the objects should be defined, in a similar way like this:

- An exercise sheet has many problems. Sheet -> [Problem]
- A problem has a one header, one problem question and 0 to many subproblems. Problem -> 1 Header, 1 Question, 0..* [Subproblem]
- All subproblems have one problem. Subproblem 1..1 Problem
- The material for the git repository has many files and folders. Material ->[File], [Folder]

It had to be different because of SQLite3 does not let us construct ArrayFields, like explained in the Database section [3.4.2]. So the final Analysis Object Model for our system is the one in Figure [3.5].

3.6 Tools

The tools used in this project had to be open source and preferably already in the TUM Ecosystem, to speed up the learning curve. In the end, the whole system should only have its data inside the university network and servers, to complain with the privacy policy of the TUM [1].

The Task Manager

A task manager helps to organize software projects, and assists the developers in creating good practices. Furthermore, it gives the possibility of making to-dos in an intuitive way and being able to issue new ideas or changes for the project.

[1] TUM Privacy Policy

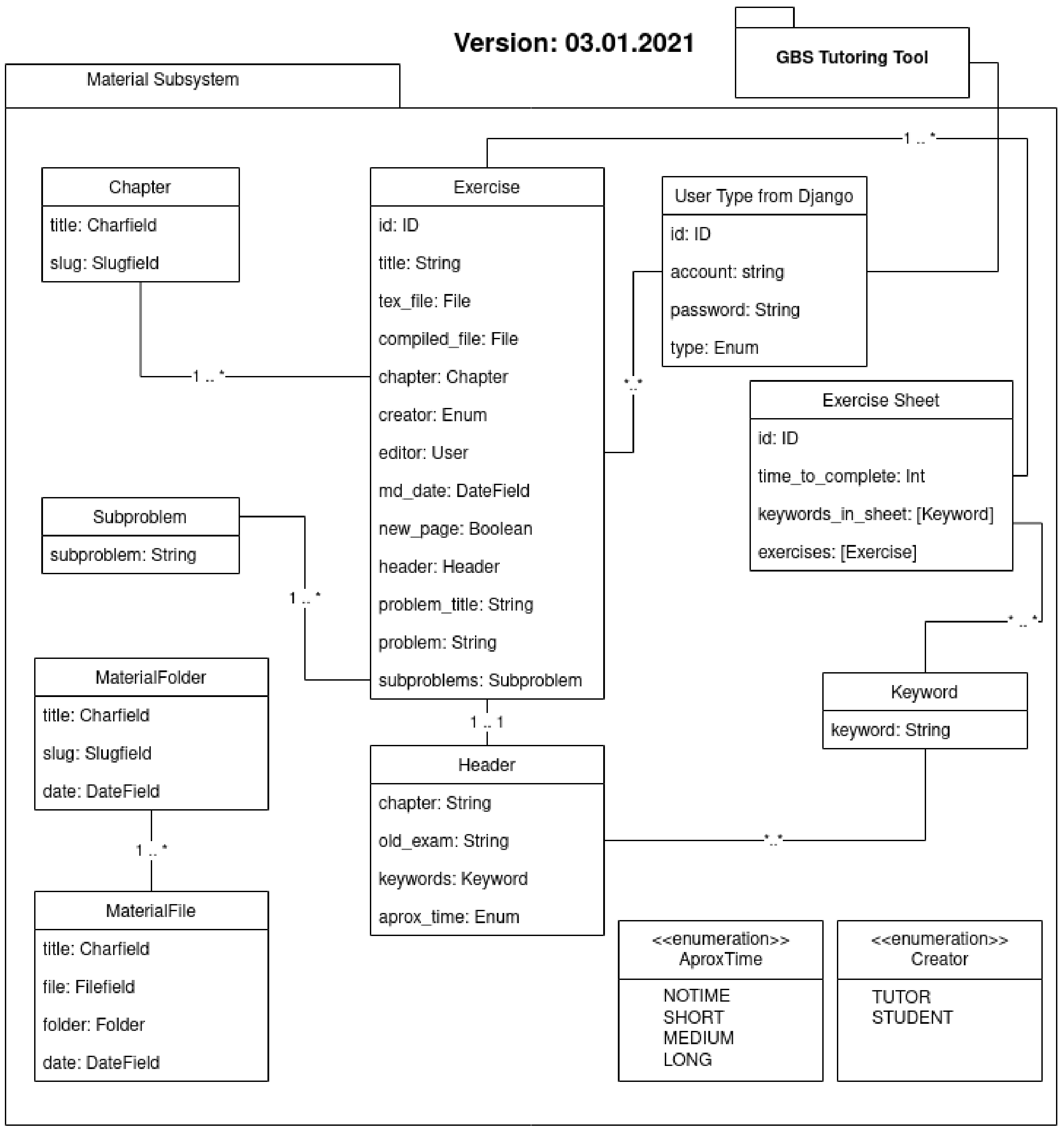


Figure 3.5: Analysis object model describing the classes associations from an application-domain perspective (UML class diagram).

In general there are two considerable options: GitLab, the open source git version control[2] service, and the Jira projects administration from the company Atlassian.

For this project GitLab was chosen, because both have a to-do section, a Milestones plan and a place to document the progress [Con20]. However, the free verson Jira had to be located in their server [Atl20], which creates privacy issues and using unneeded bandwidth. Jira is better designed for big developer teams, which is currently not needed. In the future a change to the more productive task manager, can be easily done if required. Therefore, no new program had to be added to the ecosystem, because GitLab is already used for the GBS projects and runs in the "Leibniz Rechenzentrum" (LRZ) servers at the TUM campus.

[2]Git Version Control is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

4 System Design & Implementation

Researching the software patterns from our reference book "Object-Oriented Software Engineering Using UML" [BD13] chapter 8, we investigated the key characteristics of different software patterns that give more functionality to an older system. Thereon we find the best practices for designing a system with subsystems interacting and how to decompose the subsections.

4.1 Subsystem Decomposition

In order to describe the integration of the proposed system into the GBS Tutoring Tool system, we present a subsystem decomposition in Figure [4.1].

Here you can view how the main components of the new web system work with the environment and the other subsytems. The "**RbgAccountManager**" component of the subsystem interacts with the TUM LDAP Server to retrieve the user accounts data and is implemented in [4.2.1]. It is the only component that interacts out of our system environment, because it uses the TUM LDAP structure. If this structure should change, we also have to adapt our component implementation. But considering that the structure was made to remain unchanged, we probably will not have to deal with actualizing this component in the near future.

The "**ExerciseObjectCreator**" deals with the creation of new problems in the web system and stores them in the "**ExerciseSynchronizer**" if the creator is a logged-in user of the "**RBGAccountManager**". The "**ExerciseSynchronizer**" stores all new changes in the "gbs-tutorials" Git repository and receives the actualization from this once every five minutes to maintain a synchronization between the two main subsystems.

Finally the "**MaterialObjectsCreator**" loads all the material data after every change in the "Material" Git repository. This connection is unidirectional, because only the GBS Kern is in charge of the "Material" Git repository and they prefer to work only with the Git repository. After every change in the repository, the system reflects these changes in its internal state.

4.2 User Registration and Authentication

As stated in our requirements, our system should already use the well known "Rechnerbetriebsgruppe"(RBG), is the system administration group for the TUM Informatics Faculty (RBG) accounts, that every Informatics student at TUM has, to have a more secure system and lowering the system's complexity by not having to deal with user credentials ourselves.

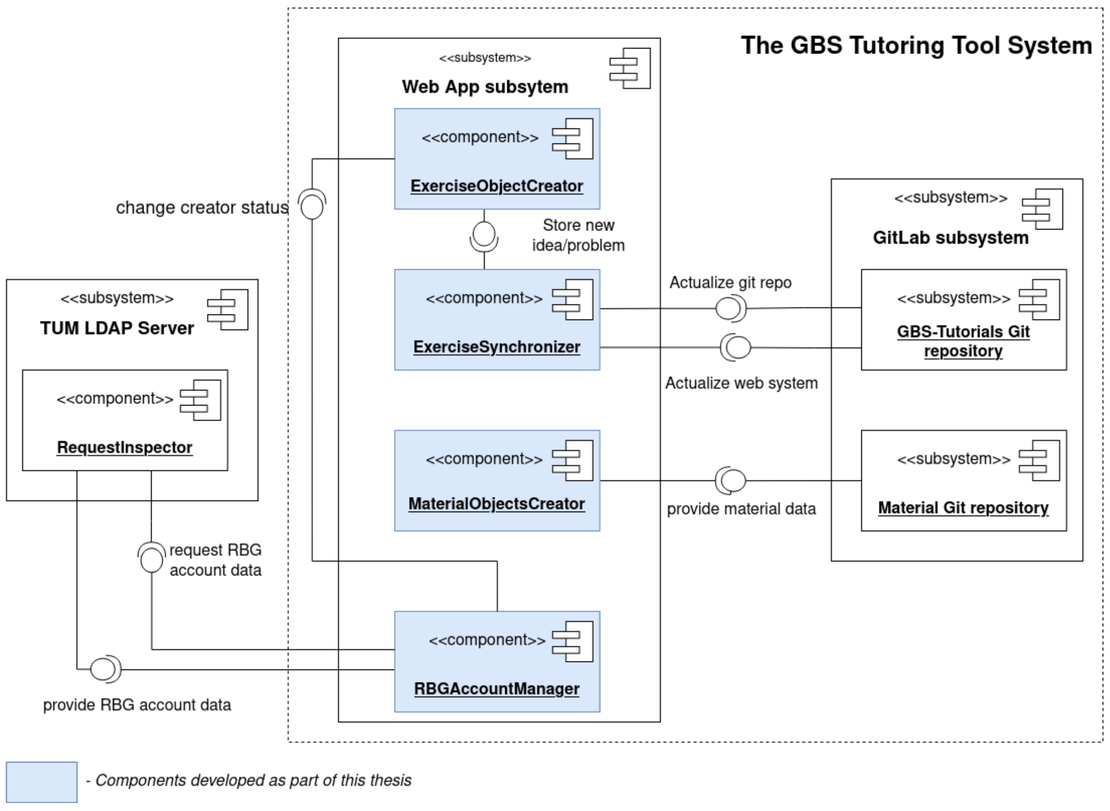


Figure 4.1: Subsystem Decomposition (The UML Component and Subsystem Diagram).

One of the reasons we chose Django is that it comes with really good security and well tested authentication packages [Fou20b], that are important to reduce the risk of creating privacy issues. Another strategy that we wanted implemented, was to not store the passwords of our users, but that the system communicates with the TUM LDAP Server, to not only retrieve the students most basic data but to confirm that the user password is correct.

Lightweight Directory Access Protocol (LDAP), is a base and flexible protocol to be able to access the Directory Servers. When we refer to the TUM LDAP Server, we refer to the server where all the data about the RBG accounts is stored.

There were many possibilities to implement this user logic, but in the end we had to chose the one that the tutor leader and admin found more comfortable to use, because he was going to be the only one modifying the users.

The implementation works as stated in the following: Before the start of every term, the tutor leader actualizes a .CSV file in the project virtual machine (server). In this file, there are written the usernames that can access the web system. The system stores these usernames as user objects, and when you log in for the first time, it actualizes the data from the LDAP server into the websystem. Everytime you log in, it checks first if you are in the list of users from the system and if so, it checks your credentials in the LDAP server. If the LDAP server accepts your request, you are logged in.

4.2.1 LDAP

A LDAP search in the TUM RBG server can be achieved with the following command in a normal terminal:

(For it to work you have to be in the "Munich Scientific Network" (MWN))

```
ldapsearch -H ldaps://ldap.in.tum.de -x -b
'ou=Studenten,ou=Personen,ou=IN,o=TUM,c=DE'
-D 'uidNumber=23118,ou=Studenten,ou=Personen,ou=IN,o=TUM,c=DE'
-W uid=username
```

We based our implementation in this group structure, to find all persons with rbg accounts. Luckily, there is a django package to communicate with LDAP servers. And we used their documentation to program our implementation [Sag19].

```
AUTH_LDAP_SERVER_URI = "ldaps://ldap.in.tum.de:636"
AUTH_LDAP_USER_SEARCH = LDAPSearch("ou=Personen,ou=IN,o=TUM,c=DE",
ldap.SCOPE_SUBTREE,
"(&(uid=%(user)s)(objectClass=inetOrgPerson))")
AUTH_LDAP_USER_ATTR_MAP = {"first_name": "givenName",
"last_name": "sn", "email": "mail"}
```

As can be noted, the search tree is very similar in both implementations, but the different commands documented in our systems source code is what really enables to use only the usernames from the .CSV file

4.3 Synchronization with Git

Our synchronization algorithm was activated every five minutes by the APScheduler and used methods of GitPython to check whether an actualization was available and if so, to perform this actualization.

Our scheduler is a BackgroundScheduler that does not interrupt the system as it acts asychrnonuously. This scheduler was chosen over other solutions like cron because it offers a range of options that allow for an efficient scaling.

The best approach to synchronize the web server with the Git System was using the "GitPython" framework to communicate with the Git repositories [TC15].

4.3.1 GitPython Framework

The GitPython Framework offers a Python-API that permits us to interact with the Git repositories from inside of Python instead of having to run the commands manually on the command line or with e.g. subprocess calls.

4.3.2 Problems Synchronization Logic

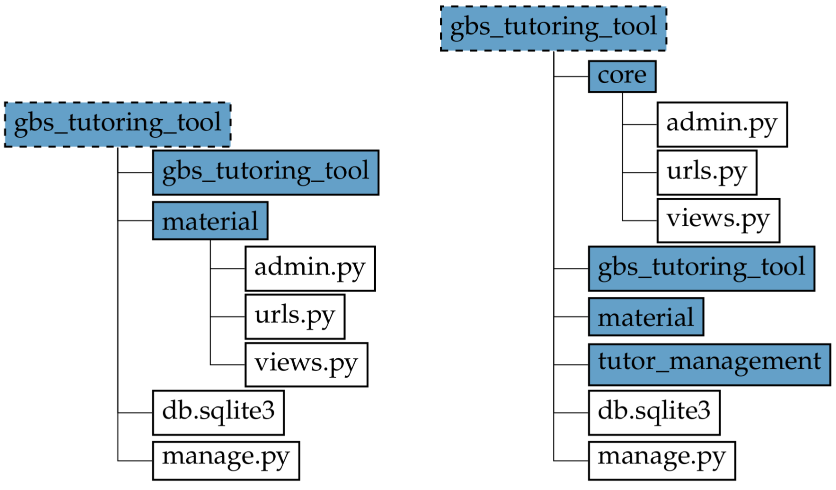
After checking whether a change has occurred in the repository, the synchronization routine checks all files' modification date to find the files that have been affected by the change. A changed file will then by converted into an object by scanning over its contents and pattern matching the parts to the corresponding object variables as explained in the "Base Problem Structure" section [3.2]. This object then is saved in the problems database and can thus be used by the web system.

4.4 Project Structure for new Subsystems

While working on this thesis, a new developer joined the project and started implementating the substitution of the third Git repository "Tutorial", which lies out of this thesis' focus (as seen in the section "Decomposition of the Git System").

The developer wanted to use another framework that worked with Django and to facilitate the communication of his subsystem with the project base, we remodeled the folder and file structure, so that his application only had to communicate with the core methods and files. This improves the adaptability to new subsystems even more, because it allows for a modular subsystem structure. This way, all subsystems can be independent of eachother.

The figure [4.2] represents the folder structure change, based on the django documentation [Fou20b] to work best with our particular case. Before, all the Uniform Resource Locator (URL) logic was contained in the "material" app. Afterwards, changes had only to be made to the "core" when implementing a new subsystem. The "material", "tutor_management" and future apps are completely independent.



(a) Old Project Structure (b) New Project Structure

Figure 4.2: Structure change from the "gbs_tutoring_tool" project folder. White boxes represent files and blue boxes folders.

4.5 Web System Home

As already explained, we wanted to include the overview given by the directory listing from "https://gbs.cm.in.tum.de" in our web system, where it should serve as the "home" page for the GBS Tutoring Tool. All changes made by the synchronization are directly reflected in this view. From this starting point, we also implemented small convenient functionality like downloading files and whole directories from the "home" page.

4.6 LaTEX Editor Implementation

There was the idea in the brainstorming of having an "Advanced Mode" that lets us create a file directly in an editor of the web system, to create for example graphs directly in the web system. But after some time into the implementation and some meetings with the GBS Kern, it became clear that the web system would not be able to perform as good as compiling the latex code locally. Not being the systems original use case, we do not view this as a strong limitation. The GBS Tutoring Tool system should connect both subsystems but not substitute GitLab. Still, we explained the points why it is not possible to mimic GitLab in a good enough way in section [4.6].

"However, it is not always the case when there is a need for a different approach when solving the same problem. For example, if one has to modify the view to work differently than it might require much work. This requires much investigation and reading the source code. "[Bha20]

4.7 The Creation of Exercise Sheets

Implementing the exercise sheets, we ran into some problems. We were able to create exercise sheets with many problems, but we were not able to implement a working solution for merging several problem statements from different files into a single exercise sheet. The tests usually failed, and we could not deliver this functionality completely before the deadline of this thesis.

Since the functional requirement "Automatic creation of Exercise Sheets" (FR7) was dependent of this method, we could not start to implement it, leaving the FR7 not accomplished.

Nevertheless, we will continue to work on these features, and deliver them before the start of the next GBS lecture term.

5 Deployment

5.1 VM Infrastructure

For the deployment of the web system, we requested a virtual machine in the university network. This VM allows us to have the system up and running the whole time. Moreover, the abstraction done by a VM also allows the program to run in a well-defined environment, reducing the amount of possible error sources.

In order to work with the VM from the university, research on the VM infrastructure was made.

Also, to support a possible large number of students accessing the system simultaneously, the VM was equipped with sufficient RAM and computing power, although this can easily be extended as it is a **virtual** machine.

In order to get the machine ready for our system, we had to open the ports 22 (SSH), 80 (HTTP) and 443 (HTTPS).

5.2 The Web Server

We chose to use the Apache web server as it is one of the most established ones available and as it is easy to get a simple configuration working. However, it also allows for complex configurations, making it a suitable choice for this project.

The web server is the base point for all requests and calls our python executable via WSGI, when getting a request.

6 Results

One important aspect of the project was the participation of old and new tutors from the GBS subject, who gave us ideas about which key elements could make the system a success (see Strategy section [1.4] from the Introduction).

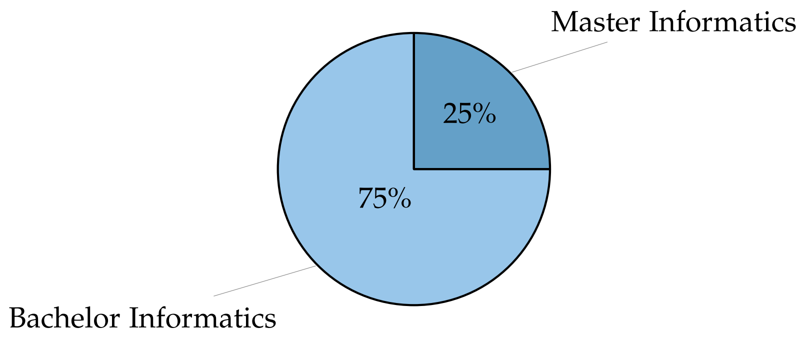
The key aspects for a good accomplishment of the project were the structured communication and organized meetings with all actors. Explaining the possible value and the part the tutors played in the development process of the system, motivated sufficiently many tutors to give us a variable input of ideas and points of view. This way, we also could get more feedback data after the final meeting, where the finished system was explained with a demo:

6.1 Feedback Data

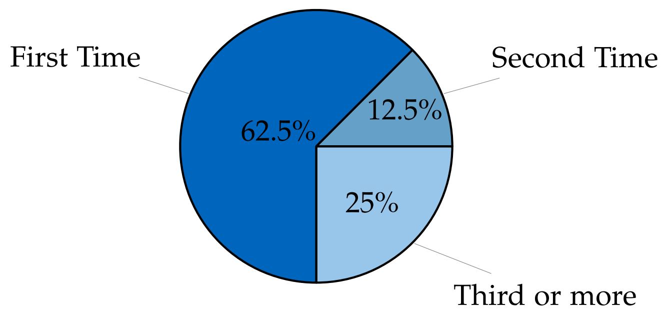
By letting the tutors use the web app for some basic tasks, we are going to be able to get their feedback on how good the suggested elements for the system were accomplished. They should then do the following: enter their profile, create one exercise, change one exercise, create a sheet for the students and search for keywords.

Of the 21 tutors in the GBS semester of 2020/2021, eight took the questions on the feedback formulary, one of them was in the GBS Kern while being tutor. All of these tutors participated in the first survey already. Less members of the GBS Kern took the feedback formulary, as we had a group meeting at the end of the project. Therefore, the change between the formularies about the first two question makes sense. This is the feedback survey data used to improve the final system:

1. **Are you a Bachelor or a Master student?**



2. **How many times have you been a GBS Tutor?**

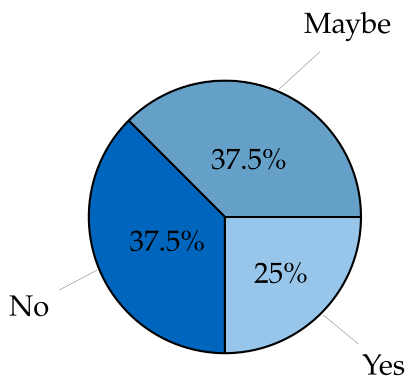


3. **What do you think about the web system substituing https://gbs.cm.in.tum.de ?**

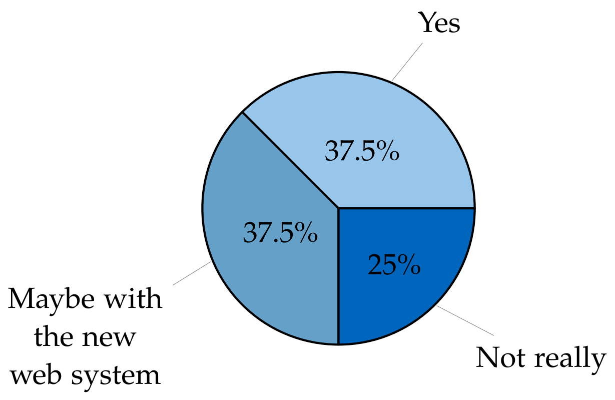
   All respondents found that it was good, that the web system substitutes the directory listing https://gbs.cm.in.tum.de.

   (100% Good, 0% Not helpful, 0% Bad)

4. **Do you think that students will send GBS exercises suggestions on the web application?**



5. **Can you imagine creating new GBS exercises on the WebApp?**



6. **Do you think that you will edit or correct problems on the web application?**

   All respondents answered that they see themselves editing and correcting problems on the web application.

   (100% Yes, 0% Maybe, 0% Not really)

7. **Do you have any ideas for the system's future functionalities?**

   a) Maybe some level of integration with Artemis, more tools for creating the tasks (some tasks like in Introduction to Theoretical Computer Science webtool).

   b) Tools for the visualization of algorithms (e.g. buddy, clock, etc.).

   c) Similarly to many classic LaTEX editors, the finished worksheet over runtime could be displayed on the screen (e.g. on the right).

   d) Central contact point for all GBS matters, i.e. also Moodle replacement. In addition, a separate problem tester can be added in the long term, then there really is only one GBS page and no longer Artemis, Moodle and gbs.cm.in.tum.de

8. **Do you have any suggestions or comments for the WebApp?**

   a) More feedback for wrong username or password.

   b) Write down that Shibboleth is not used but the RBG login is required.

   c) For the "Students GBS Material" a text field which renders the content of the Readme.md, if this is stored in the repository.

   d) Better icons. Maybe the one from here: https://github.com/vscode-icons/vscode-icons since the License (MIT) would fit.

   e) Unicode encoding for files.

   f) Change link Create an Exercise for the GBS Lecture -> Create an Exercise.

   g) No limitation to alphanumeric characters in the title of a problem.

   h) More fancy.

   i) On the start page, it irritates me that the file symbol to the left of the file name does the same thing as when you click on the file name.

   j) It would probably also be good at some point to add a manual on how to create a problem/exercise or how the format should / can be. If it is possible to insert pictures, then also how etc.

   k) I like it very much, only the name "title" in the top line when creating a new problem/exercise confused me for a moment. I thought at first that it was meant to be the title that is displayed, otherwise the webpage is very intuitive.

   l) Interface could be made a bit nicer so that it looks more modern and easier on the eyes. But this is not critical.

6.1.1 Feedback Conclusion

In general the feedback was really encouraging, seing that all tutors had a positive view about the substitution of the old directory listing "https://gbs.cm.in.tum.de" (question [3]) and that everyone saw themself editing and correcting problems on the web application (question [6]).

Most of the tutors could see themselves creating new problems (75%) and were positive that students will use the web application for creating new problems (62.5%), which is very similar to the tutors' thoughts at the start of the semester (66.6% and 75%) (see data from questions [5] and [6] of the first survey).

This is great news, because it shows that the tool achieved the tutors expectations about how the system had to be, in order to be attractive to them and the students.

From the last question [8], we learn that most proposed changes are small adjustments or UI improvements. This indicates that the system was working as it was expected from the tutors. Almost all suggestions were implemented before the delivery of this thesis - only the "No limitation of alphanumeric characters" (see suggestion [8g]) comment had to be refused, since we wanted it to represent a "slug".

6.2 Benchmarks

The NFR6 "An Exercise should compile and be shown in less than 5 seconds." is a bit abstract to prove, since there are many different use cases, because LaTEX is Turing-complete and it really depends on the specific LaTEX file. This is why we tested it for our main use case: The creation of new simple exercises.

We compiled 20 new exercises and measured them with the python shell. We timed from the moment we clicked submit to the moment we were redirected. This contains not only the compilation time, but the network delay, which should be neglectable.

Out of these 20 simple exercises of different lengths, the compilation time for all exercises was very similar and resulted in an average compilation time of around 3.2 seconds.

The NFR7 had to prove that the system can recreate all the internal objects from the material from the "Material" Git repository in less than 10 seconds. The Git repository data is 1061400 bytes big, roughly about 1,1GB when testing the benchmarks. We tested the implemented method directly per ssh with this python code:

```
>>> start = time.time() ;\
... actualizeMaterial(material_git_path) ;\
... end = time.time() ;\
... print(end-start)
```

And created a main value between all 20 tests. We made sure to leave exactly one second between each timing to have more reliable data.

$$\frac{(4.613s + 4.432s + 5.093s + 4.619s + \dots)}{20} \approx 4.804 \text{ seconds}$$

Therefore, this benchmark for the content of the folder of 1,1GB showed us, that our system can handle actualizing all Material from the Git repository in less than 5 seconds, which leaves our non-functional requirement fulfilled.

6.2.1 Benchmarks Conclusion

As can be seen in the conducted benchmarks, the bottleneck of our system is the compilation of the LaTEX files, since even a simple and short LaTEX problem already needs around 3 seconds to compile. For a large amount of files, the combined compilation time can thus be arbitrarily big, scaling proportionally to the number of files (as no multithreading/multiprocessing is implemented).

7 Summary

7.1 Status

The status of the functional requirements (FRs) and non-functional requirements (NFRs), from the section [2.3] will now be presented. The level of fulfillment is divided in these three measures:

● **Requirement was accomplished:** The requirement is implemented and finished.

◐ **Requirement was partially accomplished:** The requirement is not finished, but partially achieved. The implementation has to be completed to be accepted by the client.

○ **Requirement was not accomplished:** The requirement was not achieved.

7.2 Realized Goals

We fulfilled most of our functional requirements as seen in this table at the time of our writing (see full requirement in section [2.3]):

Functional requirements		Status
FR1	Create Exercise	●
FR2	Correct Exercise	●
FR3	Synchronization with GitLab	●
FR4	List GBS Material	●
FR5	Search Exercises	●
FR6	Create Exercise Sheets	◐
FR7	Automatic creation of Exercise Sheets	○

Table 7.1: Implementation status of the functional requirements.
(● completed | ◐ partially completed | ○ incomplete)

Nonfunctional requirements		Status
NFR1	Ease of use	●
NFR2	Ease of use	●
NFR3	Ease of use	●
NFR4	Safety	●
NFR5	Robustness	●
NFR6	Performance	●
NFR7	Performance	●
NFR8	Maintainability	●

Table 7.2: Implementation status of the nonfunctional requirements.
(● completed | ◐ partially completed | ○ incomplete)

As we saw in the given feedback, we accomplished our "ease of use" NFRs. The benchmarks show the fulfillment of the "performance" NFRs, while the "safety", "robustness" and "maintainability" NFRs were fulfilled by the used packages and project structure.

7.3 Open Goals

Although we completed all NFRs, there is one functional requirement that was partially accomplished (FR6) and another that was not accomplished (FR7). FR6: "Create Exercise Sheets: Users can make exercise sheets and view the compiled file in the web system.", could not be accomplished, because as seen in section [4.7], the state of the implementation is not stable. The lack of time played the greatest part in not fulfilling these requirements before the deadline of this thesis.

7.4 Conclusion

In this work, the design and implementation of the software engineered project GBS Tutoring Toolwas presented and its complications elucidated. This new web system makes it possible for the GBS course to have organized exercises that are accessible to tutors and students. The system makes it easy to create and edit exercises, without having to master LaTEX. Nevertheless, this new system can still be used in the old context of “GitLab”, since it interacts with it.

We implemented the system communication with the TUM LDAP server to retrieve the users data and reduce the complexity of user management as the admin only has to change a single file. We created a synchronization algorithm that runs with a scheduler, to maintain both subsystems up to date.

We programmed the logic for the creation and evaluation of new material and developed the strategies for adding functionality to the problems.

In addition, we refined the project's file structure to make it easier to adapt the project to new applications, like the future subsystem "Tutor Management Application".

Finally, we deployed the system in a university server and subjected it to tests with the GBS Kern and tutors.

In order to achieve these goals, the main strategy was to collaborate with tutors to collect as many ideas as possible. Then they were analyzed and the first version of the GBS Tutoring Tool was completed. However, to start a whole new system for only one developer was no easy task. Coordinating meetings with the GBS Kern and the tutors, gathering their feedback and improving the system demanded a lot of time.

Our part now is to motivate new developers to keep on working on the system and make it an even better tool for students and organizers. We already accomplished this with the next thesis, where a "Tutor management Subsystem" will be implemented. However, there are more meetings planned, where we will introduce the system and explain the future work to the next developers for the GBS Tutoring Tool.

7.5 Future Work

The GBS Tutoring Tool is already in a shippable state. Still, we should try to improve it mostly before it begins to be used at the start of the 2021/2022 winter term. The first next steps, will be to find the solution for the exercise sheets creation problem, reimplement it and test it again, until it is in a usable state. Like we explained in the benchmarks conclusion, new techniques for dealing with the system's bottleneck should be implemented, such as multiprocessing or multithreading.

The most valuable new functionality would be that the tutors and students could give feedback to every problem. Then it would be easier and quicker to know which problems should be redefined or are really well explained.

Another idea that was proposed in the Google Forms was to implement a statistics about the problems. This would help a lot to understand which material students are reviewing the most, or which are the problems that are causing the most difficulties to understand. It could enable to measure which parts of the lecture are already explained well and in which chapters there should be more problems for the students to practice.

In our feedback survey, the students proposed more tools inside the web system, such as tools for the visualization of algorithms (e.g. buddy, clock, etc.) or to integrate the system with "Artemis" in the question [7]. It would make the online teaching less complicated if both systems could communicate automatically. But first, it should be investigated what is needed and how the implementation and maintenance efforts relate to the derived benefit.

In addition, if the problems' quantity increases much in the future, there are django packages that help with implementing "Pagination" in the web sites. Pagination lets you decide how many problems to show per webview, to help dealing with possible performance issues that could arise from a large number of problems. As of right now, it doesn't affect the performance of the system, since there are no more that 50 problems.

List of Figures

List of Tables

Bibliography

[Son+04] L. Song, E. S. Singleton, J. R. Hill, and M. H. Koh. “Improving online learning: Student perceptions of useful and challenging characteristics”. In: *The Internet and Higher Education* 7.1 (2004), pp. 59–70. ISSN: 1096-7516. DOI: https://doi.org/10.1016/j.iheduc.2003.11.003. URL: https://www.sciencedirect.com/science/article/pii/S1096751603000885.

[KS18] S. Krusche and A. Seitz. “ArTEMiS: An Automatic Assessment Management System for Interactive Learning”. en. In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education - SIGCSE '18*. Baltimore, Maryland, USA: ACM Press, 2018, pp. 284–289. ISBN: 978-1-4503-5103-4. DOI: 10.1145/3159450.3159602. URL: http://dl.acm.org/citation.cfm?doid=3159450.3159602.

[BD13] B. Brügge and A. H. Dutoit. *Object-Oriented Software Engineering Using UML, Patterns, and Java*. eng. Third Edition. Pearson, 2013. ISBN: 978-1-292-02401-1.

[Gue+17] G. Guest, E. Namey, J. Taylor, N. Eley, and K. McKenna. “Comparing focus groups and individual interviews: findings from a randomized study”. In: *International Journal of Social Research Methodology* 20.6 (2017), pp. 693–708. DOI: 10.1080/13645579.2017.1281601. eprint: https://doi.org/10.1080/13645579.2017.1281601. URL: https://doi.org/10.1080/13645579.2017.1281601.

[VN16] N. Vasantha Raju and H. N.S. “Online survey tools: A case study of Google Forms”. In: Jan. 2016. URL: https://www.researchgate.net/publication/326831738_Online_survey_tools_A_case_study_of_Google_Forms.

[Jou18] T. E. Journal. *Python is becoming the world's most popular coding language*. https://www.economist.com/graphic-detail/2018/07/26/python-is-becoming-the-worlds-most-popular-coding-language. Accesed on: 20/10/2020. 2018.

[Fou20a] P. S. Foundation. *The Python Programming Language Documentation, Version 3.1*. Last accesed on: 10/01/2021. Aug. 2020. URL: https://docs.python.org/3.8/.

[Rog17] J. Rogowski. “The Comparison of the Web Application Development Frameworks”. In: *Institute of Information Technology, Lodz University of Technology* (Mar. 2017). URL: https://itcm.comp-sc.if.ua/2017/Rogowski.pdf.

[Fou20b] D. S. Foundation. *The Django Project Documentation, Version 3.1*. Last accesed on: 15/01/2021. Aug. 2020. URL: https://docs.djangoproject.com/en/.

[Ben20] A. Bendoraitis. *Django 3 Web Development Cookbook - Fourth Edition*. Ed. by J. Kronika. Packt Publishing, 2020. URL: https://learning.oreilly.com/library/view/django-3-web/9781838987428/.

[Ron20] A. Ronacher. *The Pallets Projects: Flask Documentation, Version 1.1*. Last accesed on: 10/11/2020. Aug. 2020. URL: `https://flask.palletsprojects.com/en/1.1.x/`.

[Bha20] V. Bhatnagar. *Comparative study on Python web frameworks: Flask and Django*. May 2020. URL: `https://urn.fi/URN:NBN:fi:amk-2020052513398`.

[KH09] J. Kaplan-Moss and A. Holovaty. *The Definitive Guide to Django: Web Development Done Right, Second Edition*. Ed. by D. Parkes. Apress, 2009. URL: `https://learning.oreilly.com/library/view/the-definitive-guide/9781430219361/`.

[Con20] S. F. Conservancy. *GitLab Documentation*. Last accesed on: 20/10/2020. Oct. 2020. URL: `https://docs.gitlab.com/ee/`.

[Atl20] Atlassian. *Jira Documentation. Version 6.4*. Last accesed on: 20/10/2020. May 2020. URL: `https://confluence.atlassian.com/jira/jira-documentation-1556.html`.

[Sag19] P. Sagerson. *Documentation Django Authentication Using LDAP, latest Version*. Last accesed on: 15/01/2021. May 2019. URL: `https://django-auth-ldap.readthedocs.io/en/latest/index.html`.

[TC15] M. Trier and Contributors. *GitPython Documentation, stable Version*. Last accesed on: 05/02/2021. May 2015. URL: `https://gitpython.readthedocs.io/en/stable/`.