BEPSys 2.0

Central Registration Tool for Projects and Groups

J. W. D. Alderliesten
F. P. Doolaard
J. Tilro
N. Warnars

Delft University of Technology

TU Delft

BEPSys 2.0

Central Registration Tool for Projects and Groups

by

J. W. D. Alderliesten
F. P. Doolaard
J. Tilro
N. Warnars

to report on the process of designing and implementing BEPSys 2.0.
The final presentation was held on July 6th, 2017 at 10:00
in lecture hall D@ta (EEMCS)

This report was edited to remove security details about the system active at the time of the report's release to prevent exploits from being utilised.

Project details	
Project duration:	April 17, 2017 – July 06, 2017
Coach:	Alessandro Bozzon
Supervisor:	Huijuan Wang
Product Owner:	Otto Visser

TUDelft Delft University of Technology

Preface

This report was written alongside the development of the BEPSys 2.0 application, a system that provides project registration, management, and approval for companies and research groups that wish to offer projects to students at the TU Delft.

The development of this system was a part of the bachelor project course TI3806. After a research phase of two weeks, the system was developed within seven weeks in Ruby on Rails and delivered to to the client.

The development team would like to thank the following people for their support and help:

- Otto Visser, TU Delft
- Alessandro Bozzon, TU Delft
- Huijuan Wang, TU Delft
- Cynthia Liem, TU Delft
- Eva de Haan, TU Delft
- Lily Stamenova, FeedbackFruits

Without them, this project would not have been possible. Their feedback and desire to help during testing sessions has ensured that many problems and bugs were identified before deployment, and the application is better due to their varied and detailed input.

J. W. D. Alderliesten
F. P. Doolaard
J. Tilro
N. Warnars
Delft, July 7, 2017

Summary

The following report outlines the research, development, and delivery phases of the BEPSys 2.0 application. The application aims to provide central project registration and management for courses at the TU Delft. The request for the development stemmed from Otto Visser, who was the acting coordinator for the Computer Science Final Project Course. A system named ”BEPSys” already existed for this purpose, but it was deemed insecure and unreliable, and warranted a redesign. Additional background information is provided in chapter 1.

Chapter 2 provides an outline regarding the planning of the development process. The first planned phase was a research phase, in which a detailed planning with research goals was created. These topics included group formation, technology to be used, and an audit of the current BEPSys system to determine possible additions upon it. The research phase was concluded with a research report, which can be found as Appendix F. After the research phase, the implementation phase took place, in which a new application was developed from scratch. The third and final phase was a wrap-up phase, in which the final product was polished and enhanced as required. A retrospective is also found in Chapter 2.

Chapter 3 contains the requirements as obtained from a set of stakeholder interviews. These interviews yielded a MoSCoW requirements list, which would be utilised to guide the design of the new application. Stakeholders interviewed include the client, users, coaches, and external companies. Chapter 3 also includes the definition of done employed by the development team.

Chapter 4 outlines the methodologies & tools that were utilised by the development team. The outline covers the development life-cycle model employed, the version control system used, the approach to code review and merge requests, the software employed for issues tracing, and the task automation that was utilised.

Chapter 5 includes technical considerations, including the reasons to start over and scrap the previous version of BEPSys, the decision to employ the Rails framework, and the process and decisions made regarding the migration of data from the old application to BEPSys 2.0. After deliberation with both the client and our technical research, the decision was made to not carry data over between the two applications.

Chapter 6 provides an overview of the design of BEPSys 2.0, including modelling and process diagrams, front-end mock-ups and designs, colour and theme choices made to adhere to the TU Delft’s style guide, and the process outlined for group formation.

Chapter 7 highlights the differences between the implementation and the design, and outlines certain aspects of the implementation that required additional research and validation.

Chapter 8 provides an overview of the testing & quality assurance procedures utilises during the development of the application. These aspects include the software verifications techniques used, such as dynamic testing and automated testing, and the software validation process employed.

Chapter 9 provides an insight into aspects related to the deployment, including the production environment configuration, scalability considerations, and the Ansible configuration automation implementation. A highlight is also given to the Capistrano deployment automation system for the production and staging environments.

Chapter 10 identifies crucial aspects of security. A segment of considerations regarding OWASP’s top ten vulnerabilities is addressed, followed by an overview of security gems em-

ployed by the application, and the process of secure development is outlined.

Chapter 11 outlines ethical considerations that took place for the BEPSys 2.0 application, such as the privacy of student and staff information, and the effects of confirmations within the system due to their binding status regarding student progress at the TU Delft.

Contents

Preface **i**

Summary **ii**

List of Figures **vii**

1 Introduction **1**

2 Planning **2**

2.1 Planning Phases 2

2.1.1 Phase 1: Research 2

2.1.2 Phase 2: Implementation 2

2.1.3 Phase 3: Wrap Up 3

2.2 Retrospective. 3

3 Requirements **4**

3.1 Definition of Done 4

3.2 Resulting Requirements 4

3.3 Stakeholder 1: Client 4

3.3.1 Functional requirements 5

3.3.2 Non-functional requirements 6

3.4 Stakeholder 2: Counsellor 6

3.4.1 Functional requirements 6

3.4.2 Non-functional requirements 7

3.5 Stakeholder 3: TU Coach 7

3.5.1 Functional requirements 7

3.5.2 Non-functional requirements 8

3.6 Stakeholder 4: Company 8

3.6.1 Functional requirements 8

3.6.2 Non-functional requirements 9

3.7 MoSCoW Requirements 9

3.7.1 Must Have 9

3.7.2 Should Have 11

3.7.3 Could Have 12

3.7.4 Would Have 12

4 Methodologies & Tooling **13**

4.1 Development Life-Cycle Model 13

4.2 Version Control using Git & GitLab 13

4.3 Modern Code Review through Merge-Requests 14

4.4 Issue Tracking using JIRA 15

4.5 Task Automation using Rake. 15

5 Technical Considerations **16**
5.1 Starting Over . . . 16
5.1.1 Improving Maintainability . . . 16
5.1.2 Preventing Flaws in BEPSys 2.0 . . . 17
5.2 Rails Framework . . . 18
5.2.1 Model-View-Controller . . . 18
5.2.2 Representational State Transfer . . . 18
5.2.3 Language and Framework Preferences . . . 18
5.2.4 Justifying Maintainability . . . 19
5.3 Migrating data to the new production environment . . . 19
5.3.1 Technical Considerations . . . 19
5.3.2 Client Preferences . . . 19
5.3.3 Migration Procedure . . . 20

6 Design **21**
6.1 Decisions . . . 21
6.2 Modelling . . . 21
6.2.1 Process Flow Modelling . . . 21
6.2.2 Data Modelling . . . 23
6.2.3 Schema Model . . . 24
6.3 Front-end . . . 24
6.3.1 Wireframes . . . 24
6.3.2 Layout . . . 24
6.3.3 Colour Scheme & Design . . . 24
6.4 Group Formation . . . 26
6.4.1 Formation Aspects . . . 26
6.4.2 Group Formation Design . . . 27

7 Implementation **28**
7.1 Back-end . . . 28
7.1.1 Implemented Data Model . . . 28
7.2 Front-end . . . 28
7.2.1 Administration . . . 28
7.2.2 Custom theme for the Bootstrap layout framework . . . 28
7.2.3 Responsiveness . . . 29
7.2.4 Asynchronous Interface . . . 29
7.2.5 Single Sign-On . . . 29

8 Testing & Quality Assurance **30**
8.1 Software Verification . . . 30
8.1.1 Dynamic Testing . . . 30
8.1.2 Browser Automated Testing . . . 30
8.1.3 Test Coverage Analysis . . . 31
8.1.4 Static Analysis using Rubocop . . . 31
8.1.5 Security Testing . . . 32
8.1.6 Continuous Integration using Jenkins . . . 32
8.2 Software Validation . . . 32

9 Deployment **33**
9.1 Production Environment Configuration 33
9.2 Scalability 33
9.3 Configuration Automation using Ansible 34
9.4 Deployment Automation using Capistrano 34

10 Security **35**
10.1 Testing for and preventing OWASP top 10 security flaws 35
10.1.1 Injection 35
10.1.2 Broken authentication and session management 35
10.1.3 XSS: Cross-site scripting 36
10.1.4 Insecure direct object references 36
10.1.5 Security misconfiguration 36
10.1.6 Sensitive data exposure 36
10.1.7 Missing function level access control 36
10.1.8 CSRF: Cross-site request forgery 37
10.1.9 Using components with known vulnerabilities: Ruby Gems 37
10.2 Security Gems 37
10.3 Secure Development 37

11 Ethical Considerations **38**
11.1 Handling of Confidential Information 38
11.2 Status Confirmations 38
11.3 Group Formation. 39

12 Conclusion **40**
12.1 Product Status 40
12.2 Future Development. 41
12.3 Process Reflection. 41
12.4 Process Recommendations 42

Bibliography **43**

Appendices **44**

A Data Models **45**

B SIG Analysis Results **48**

C BEPSys 2.0 Colour Table **49**

D User Testing **50**

E Infosheet **56**

F Research Report **58**

List of Figures

2.1 Gantt chart of the initial project planning. 2

4.1 An snippet of the revision history of the source code under version control, illustrating how branching and merging were applied. 14
4.2 An example of a discussion taking place in a merge-request. 15

6.1 Process flow model for project creation 22
6.2 Process flow model for group joining 22
6.3 Process flow model for course enrolment 22
6.4 Initial data model 23
6.5 Wireframe layout for course selection 25
6.6 Wireframe layout for project selection within a course edition 25
6.7 Wireframe layout for project information 26
6.8 Group status business logic table 27

8.1 An example of a local rubocop violation 32

1 Introduction

The bachelor project (abbreviated as BEP based on its Dutch origin) is the final project to which students commit for the bachelor degree of Computer Science at Delft University of Technology (TU Delft). To assist in the search for a suitable project within the course, students utilize a system called “BEPSys,” also known as the ”bachelor ending project system.” The initial version of this system was developed and released in October 2013 by Sarah Bashirieh and Nima Rahbari (Bashirieh and Rahbari [2013]). In March 2017, Otto Visser (coordinator for the BEP) submitted a request for a revamp of the BEPSys system to accommodate the changing course structure and to alleviate technical and logistical problems with the current system. The final product was delivered on July 6th, 2017. The presentation of the final product took place on the same day in Room D@ta at the faculty of electrical engineering, mathematics, and computer science at the TU Delft.

The planning of the project, including the research and development phases, can be found in chapter 2 on the following page. The requirements elicitation, including the process, definition of done, and a MoSCoW prioritisation requirements list is found in chapter 3. Methodologies & Tooling, including the development life-cycle model, version control, and code review is found in chapter 4. Technical considerations, including the desire to start from scratch, the selection of the Rails framework, and the process of data migration is outlined in chapter 5. The design of the application is found in chapter 6. The implementation, with a focus on difference between design and final implementation, is discussed in chapter 7. The testing & quality assurance process is found in chapter 8. The deployment and production environment is described in chapter 9. Security flaws, including a set of top ten vulnerability mitigations as per OWASP standards, is discussed in chapter 10. Ethical considerations regarding confidential information and status handling is discussed in chapter 11.

2 Planning

This section discusses the initial planning for the project, as well as a retrospective analysis on how this planning was put into practice.

2.1. Planning Phases

In order to get an overview of the project, kick-off meetings were organized with different stakeholders several weeks before the start of the project, including the product owner and the coach. It was decided to partition the available time frame into three phases: two weeks of research, six weeks of implementation of a solution, and two weeks of user acceptance testing and wrapping up the system for deployment.

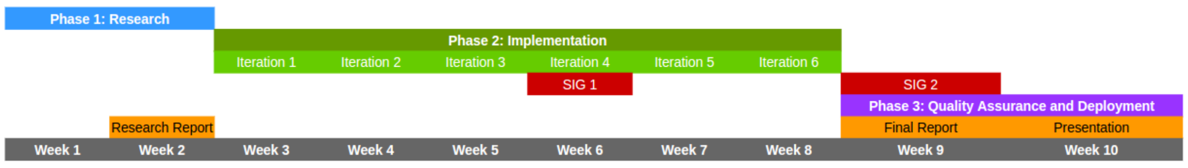


Figure 2.1: Gantt chart of the initial project planning.

2.1.1. Phase 1: Research

During the first days of the project, a detailed planning for the research phase was made. As a directive for planning and picking suitable research topics, the existing system, including its documentation and source code, were analysed. After some struggles with determining suitable research topics, it was decided that time would be spent eliciting requirements amongst stakeholders, study and document technical considerations and their justifications, data modelling, security considerations, and improving the group formation process. During this research phase a focus was also placed on identifying the weaknesses of the current system, including security vulnerabilities, in order to further justify technical decisions and prepare them for implementation. The research phase was concluded with a research report, which was delivered to both the group coach and the client. The entire research report has been included as appendix F.

2.1.2. Phase 2: Implementation

Based on the findings of the research (the designed diagrams and the elicited requirements) a planning for implementation was constructed. This was done by recording and prioritizing issues in the product backlog of the utilised issue tracking system, and distributing these over different iterations. It was planned to build the core for the system (including the application configuration, authentication, and authorization subsystems) in the first week of implementation. Afterwards, the coming five weeks were used to implement all must-have requirements. The remaining one week were to be used for implementing all should-have requirements.

Additionally, the intent was to keep the technical report and its diagrams up-to-date each iteration throughout the implementation phase.

2.1.3. Phase 3: Wrap Up

During the last two weeks, a feature-freeze was planned. This meant no new features would be implemented during this phase; the team would only test, refactor, and fix code in order to improve and assure the quality of the system. The final presentation would be prepared, and the system would be prepared for deployment to the new production environment.

2.2. Retrospective

The actual process that was conducted deviated from the initial planning in several ways.

The implementation of the core of the system upon which the implementations of the required features were to be based, took longer than anticipated. After two weeks, the first stable version of the system was finished.

Not all must-haves were finished in the allotted time frame of five weeks, meaning that the time period in which the focus was placed upon essential features was extended beyond the planned time. This ultimately meant that no feature freeze was introduced. Instead, features were developed until the deadline for the project, due to a large amount of feedback and additional requirements accumulated throughout the iterative and incremental development process.

During implementation the report was not kept up-to-date. Preparing the final report started in the last week before the deadline due to the high implementation demand.

It was decided to start the wrap-up phase one week earlier by planning user acceptance tests in week eight, in order to be able to process the feedback received through these tests.

The reasons for these deviations from the initial planning, their evaluation and the lessons learned are elaborated in section 12.3 on page 41.

3 Requirements

The requirements were elicited through the method of informal interviews during the research phase. The interviewer would pose the questions “What are the current problems with BEPSys?”, “What would you like to see improved with the current BEPSys’ features?”, and “What features should BEPSys have that do not currently exist?” The interview would be guided by the interviewer, but would consist of an informal atmosphere and allow for the interviewed person to go on a tangent to explain their statements or desires if required. All interviews had a formal style of notary recording. These written recordings would be utilised to compile the list of requirements on a stakeholder by stakeholder basis.

3.1. Definition of Done

According to the Agile Alliance, a project team should agree on a list of criteria which must be met before a product is considered “done” (Atern [2008]). A definition of done is required to prevent different stakeholders and developers from having an alternative meaning of ‘done.’ These alternative decisions could lead to conflicts regarding the completion of a project, in turn leading to uncertainty and dissatisfaction with the (final) product and the possibility of a failure to meet required deadlines. This could, in turn, lead to projects exceeding their given timeline or budget.

During the requirement elicitation a requirements list was constructed in MoSCoW (Atern [2008]) style which was used as the basis for the definition of “done”. It was decided that all ”must have” items in the requirement elicitation must be integrated in the application to consider it ”done.” Additionally, the product can only be finished when testing has finished and an automated test suite has been developed.

3.2. Resulting Requirements

Multiple interviews took place with numerous stakeholders over the research period. These interviews included the client, students, a counsellor, TU Delft coaches, and external companies that had utilised the old BEPSys system. This solicited a MoSCoW style requirements list that will be utilized to evaluate the status of the new BEPSys system.

3.3. Stakeholder 1: Client

Otto Visser served as the product owner and client for this project. His requirements were solicited through one formal interview at the start of the project, and numerous informal meetings throughout the initial research phase and duration of the project. At least one informal meeting occurred each week during the implementation phase.

3.3.1. Functional requirements

1. The system must support the utilization of the TU Delft's single sign-on system to increase efficiency of account registration and validate the student and overseeing roles within the system.
2. Projects must be able to share files between student, client, and coach. These files can be both code and literary documents.
3. Related information for a project, such as the location of the software repository and other useful information, must be supported on a project-by-project basis.
4. A single user account must have the ability to obtain multiple roles within a course, such as Course Coordinator and Coach.
5. The selection of a TU Delft coach for a project must support the ability to invite and register them into the system.
6. The creation and maintenance of groups, consisting of multiple students, must be supported. Groups should be able to invite, remove, and manage their members.
7. Groups must be able to claim a project once they match the required participant number.
8. Students without a pre-made group must be able to express their interest in a project to allow the formation of groups for these projects.
9. Companies need to be able to view their previous projects that existed in BEPSys.
10. A company may not see the projects or existence of another company in the BEPSys database.
11. The new BEPSys needs to support multiple moments of entry, such as in the second and fourth quarter of the academic year.
12. Students should be automatically approved to participate in the bachelor end project, and should automatically see a form to request entry through the counsellor if they do not meet the requirements.
13. Example documents and approval forms which are currently presented in a PDF of DOCX format should be altered to exist as a form on the website. New documents that are created should support direct entry and display in the system.
14. Companies should be able to edit projects and then submit these edits for approval to the project coordinator.
15. The new BEPSys needs to be responsive, meaning the system works and displays correctly on both mobile and non-mobile devices.
16. The deletion of user accounts should not lead to the removal or "orphaning" of their company or projects.
17. Notification messages will be sent to all users on a 24-hour basis, when applicable.
18. Features which exist in the current implementation of BEPSys need to either be altered or kept in the new version of BEPSys. No features should be lost during the development of this new version.

19. The sender of email notifications must be configurable per course.
20. The number of groups that can be admitted to a project must be configurable per course.
21. A course must be managed by a responsible user (e.g. for receiving notifications and performing administrative tasks).
22. Project selection conflicts, such as a pre-made group versus a group formed out of the interest system, must be given priority based on a first come, first serve basis.

3.3.2. Non-functional requirements

1. The new BEPSys front-end needs to be written in the official language of the TU Delft, British English.
2. The design of the front-end of BEPSys needs to conform to the official huisstijl (design) of the university.
3. System must be deployed on a production environment running an Apache web server and a MySQL database server.

3.4. Stakeholder 2: Counsellor

An interview took place with the counsellor for the Computer Science bachelor program at the TU Delft, Eva de Haan, as the representative for the counsellor role of the system. The requirements were acquired during a formal interview at the start of the project, and during informal meetings throughout the implementation phase of the project. A total of four meetings occurred with the counsellor.

3.4.1. Functional requirements

1. Notification messages will be sent to all users on a 24-hour basis, when applicable. The types of notifications should be customizable.
2. Students should be approved to participate in the bachelor project course, and should automatically see a form to request entry through the counsellor if they do not meet the requirements.
3. A student who submits an electronic entry form should be able to be approved for entry to the class by the counsellor within the BEPSys system.
4. A counsellor should be able to view the completed courses for an entry requirement within BEPSys, and export it to another format, such as an CSV or XLSX document type.
5. A counsellor should be able to export all approved people for a course to another format, such as an CSV or XLSX document type.
6. The automated barrier for entry (classes or points required) should be customizable per course.
7. The approval to participate in a course should be done on a person-by-person basis, and not per project entry request.
8. A single user account must have the ability to obtain multiple roles within a course, such as ”Course Coordinator” and ”Coach”.

9. Deadlines for issues such as registration and presentations must be placed within the system.
10. The information of a user account, regardless of role, should allow editing of information such as email and password.
11. An integration with the official grading system of the TU Delft, OSIRIS, should exist in the new system.

3.4.2. Non-functional requirements

1. The entire BEP application and approval process should be paperless for all participants.

3.5. Stakeholder 3: TU Coach

Cynthia Liem is an assistant professor of the multimedia computing group at TU Delft, and acted as the representatives of the coaches in the BEPSys system. Coaches assist groups in their projects and are a neutral assistant for student questions. Her feedback was gathered through a single formal meeting during the research phase of the project.

3.5.1. Functional requirements

1. A single user account must have the ability to obtain multiple roles within a course, such as Course Coordinator and Coach.
2. A project should have the ability to be edited, and to have that edit approved by a course coordinator.
3. A project coordinator and company should get some sort of an update (email notification) when a group has been made and their project chosen.
4. Coaches need to be able to have a complete per-course overview of all existing projects and have the ability to indicate the desired projects to coach.
5. A company needs to have the ability to change their contact information with approval of a course coordinator.
6. Notification messages will be sent to all users on a 24-hour basis, when applicable. The types of notifications should be customizable.
7. A student group should retain the ability to hand-in a project with an already assembled student team, company, and coach.
8. Anyone should have access to see the number of students in a course, and the percentage of completed projects in a course.
9. A course coordinator should only have to approve a project when there is a coach who has approved it.
10. Projects within a course should be filtered by either name, date of entry, and other possible filters.
11. A project should have an area in which students, coach, and client can see the progress and upcoming deadlines or milestones.
12. A chat system should exist that allows the client and students to communicate without having to share e-mail or other contact information. This system should also support the sharing of files.

3.5.2. Non-functional requirements

1. A company should get an example project within the BEPSys system to utilize during their set-up phase.
2. A staff member should be able to state that they have too many projects to coach when being requested for approval.

3.6. Stakeholder 4: Company

A formal interview took place with a representative of the FeedbackFruits company during the research phase of the project. Their company had four projects in the old BEPSys system, of which two were taken and completed recently. Additional insight was provided by the FeedbackFruits development team, who did this in an ad-hoc fashion after the formal interview was completed.

3.6.1. Functional requirements

1. Notification messages will be sent to all users on a 24-hour basis, when applicable. The types of notifications should be customizable. A focus should be placed on making one of these notifications occur when a group has been found and the project can be started.
2. Projects must be reviewed and then be allowed to be moved to the next edition of the course if it is not accepted in the current/past edition.
3. A project should have the ability to be edited by the company that provided/created it.
4. Companies should be able to see projects which do not exist, but which would be taken by students due to their interest.
5. Example documents and approval forms which are currently presented in a PDF of DOCX format should be altered to exist as a form on the website. Newly created documents should support direct entry and display in the system.
6. Students must be able to provide a company with some small motivation words to ensure a company that the students matches their desired profile.
7. A chat system should exist that allows the client and students to communicate without having to share e-mail or other contact information. This system should also support the sharing of files.
8. Projects must be able to share files between student, client, and coach. These files can be both code and literary documents.
9. A project should have an area in which students, coach, and client can see the progress and upcoming deadlines or milestones.
10. A user of the BEPSys system must be guided through the process they are performing (e.g: creating a project, accepting a user) in a step-by-step manner, with an example.
11. A list of related projects based on keyword or aggregation must be shown when viewing an individual project.
12. A project must be able to have more than a single TU-Coach.
13. A company must be able to edit its description, logo, and other attributes pending approval of the coordinator of the course.

14. Members of an interest list must be able to invite others into that project/interest list.

15. Each user must be able to anonymously suggest questions to a project/company through an in-system chat system in the case of questions or uncertainties.

3.6.2. Non-functional requirements

1. A company should get an example project within the BEPSys system to utilize during their set-up phase.

2. A list of possible supervisors from the TU Delft for the role of coach must be provided, with tags to their area of expertise, to suggest possible coaches for a project.

3. A company must be able to place required deadlines within the BEPSys system for a project.

4. A student must fill out a small survey when registering for a course to guide them towards a possible project or area of interest.

5. The approval check marks that are in the current version of the BEPSys system must be clarified to ensure that students and companies understand they are related to class/academic approval, and not project approval.

3.7. MoSCoW Requirements

After all the interviews were completed, an aggregation was made and discussed with the client. Based on this aggregation, a formal MoSCoW[Atern, 2008] requirements list was created. The application also received the definition of ”done” as being upheld to the completion of all the ”must have” requirements from this aggregation.

3.7.1. Must Have

1. A single user account must have the ability to obtain multiple roles within a course, such as Course Coordinator and Coach.

2. The selection of a TU Delft coach for a project must support the ability to invite and register them into the system.

3. Groups must be able to claim a project once they match the required participant number.

4. The creation and maintenance of groups, consisting of multiple students, must be supported.

5. Groups must be able to invite, remove, and manage their members.

6. Related information for a project, such as the location of the software repository and other useful information, must be displayed within the system.

7. A company must not be able to see the projects or existence of other companies in the BEPSys database.

8. The new BEPSys needs to support multiple moments of entry, such as in the second and fourth quarter of the academic year.

9. The deletion of user accounts must not be able to lead to the removal or “orphaning” of their company or projects.

10. Requirements in the current BEPSys system must be kept and transferred to BEPSys 2.0.
11. The sender of email notifications must be configurable per course.
12. The number of groups that can be admitted to a project must be configurable per course.
13. A course must be managed by a responsible user (e.g. for receiving notifications and performing administrative tasks).
14. The first group that signs up for a project and meets the requirements will get the first chance to accept the project.
15. The new BEPSys front-end must be written in the official language of the TU Delft, British English.
16. The design of the front-end of BEPSys must conform to the official "TU Delft huisstijl" (design) of the university.
17. The system must be deployed on a production environment running an Apache web server and a MySQL database server.
18. A student who submits an electronic entry form must be able to be approved for entry to the class by the counsellor within the BEPSys system.
19. A counsellor must be able to export all approved people for a course to another format, such as an CSV or XLSX document type.
20. The approval to participate in a course must be done on a person-by-person basis, and not as a per-project unique entry request.
21. Deadlines for issues such as registration, deliverables, and presentation dates must be placed within the system on a per course basis.
22. The information of a user account, regardless of role, must allow editing of information such as email and password.
23. The entire BEP application and approval process must be paperless for all participants.
24. A project coordinator and company must be able to get a notification when a group has been made and their project chosen.
25. A company must be able to edit its description, logo, and other attributes pending approval of the coordinator of the course.
26. A student group must be able to retain the ability to hand-in a project with an already assembled student team, company, and coach.
27. Projects within a course must be able to be filtered by either name, date of entry, and other possible filters.
28. A project must have the ability to be edited by the company that provided/created it.
29. Example documents and approval forms which are currently presented in a PDF of DOCX format must be able to be altered to exist as a form on the website. New documents that are created should support direct entry and display in the system.

30. A user of the BEPSys system must be guided through the process they are performing (e.g: creating a project, accepting a user) in a step-by-step manner, with an example.
31. A project must be able to have more than a single TU-Coach.
32. Members of an interest list must be able to invite others into that project/interest list.

3.7.2. Should Have

1. Students without a pre-made group should be able to express their interest in a project to allow the formation of groups for these projects.
2. Companies should be able to view their previous projects that existed in BEPSys.
3. Companies should be able to edit projects and then submit these edits for approval to the project coordinator.
4. The new BEPSys front-end should be responsive, meaning the system works and displays correctly on both mobile and non-mobile devices.
5. Notification messages should be sent to all users on a 24-hour basis, when applicable. The content of these messages should be customizable.
6. A counsellor should be able to view the completed courses of a student for an entry requirement within BEPSys, and export it to another format, such as an CSV or XLSX document type.
7. A counsellor should be able to export all approved people for a course to another format, such as an CSV or XLSX document type.
8. The automated barrier for entry (classes or points required) should be customizable per course.
9. TU coaches should be able to have a complete per-course overview of all existing projects and have the ability to indicate the desired projects to coach.
10. All users should be able to see the number of students in a course, and the percentage of completed projects in a course.
11. A course coordinator should only have to approve a project when there is a TU coach who has approved it.
12. A project should have an area in which students, coach, and client can see the progress and upcoming deadlines or milestones.
13. A company should get an example project within the BEPSys system to utilize during the project set-up phase.
14. Projects should be able to be reviewed and then be allowed to be moved to the next edition of the course if it is not accepted in the current/past edition.
15. A list of related projects based on keyword or aggregation should be able to be shown when viewing an individual project.
16. The system must support the utilization of the TU Delft's single sign-on system to increase efficiency of account registration and validate the student and overseeing roles within the system.

3.7.3. Could Have

1. Students can be automatically approved to participate in the bachelor project through Osiris data, and can automatically see a form to request entry through the counsellor if they do not meet the requirements.

2. An integration with the official grading system of the TU Delft, OSIRIS, can exist in the new system.

3. A staff member can state that they have too many projects to coach when being requested for approval.

4. Projects can be reviewed and then be allowed to be moved to the next edition of the course if it is not accepted in the current/past edition.

5. Companies can see projects which do not exist, but which would be taken by students due to their interest.

6. Students can be able to provide a company with some small motivation words to ensure a company that the students matches their desired profile.

7. A list of possible supervisors from the TU Delft for the role of coach can be provided, with tags to their area of expertise, to suggest possible coaches for a project.

8. A company can place required project deadlines within the BEPSys system.

9. The approval check marks that are in the current version of the BEPSys system can be clarified to ensure that students and companies understand they are related to class/academic approval, and not project approval.

3.7.4. Would Have

1. Projects can share files between student, client, and coach. These files can be both code and literary documents.

2. Example documents and approval forms which are currently presented in a PDF of DOCX format can be altered to exist as a form on the website. New documents that are created should support direct entry and display in the system.

3. A chat system can allow the client and students to communicate without having to share e-mail or other contact information. This system can also support the sharing of files.

4. Each user can anonymously suggest questions to a project/company through an in-system chat system in the case of questions or uncertainties.

5. A student can fill out a small survey when registering for a course to guide them towards a possible project or area of interest.

4 Methodologies & Tooling

The methodologies & tooling section aims to describe the processes utilized by the project team during the development of the BEPSys 2.0 application. The development life-cycle is described, version and code control methodologies are explained, testing and task automation is mentioned, and tools utilized for clean testing and task automation are discussed.

4.1. Development Life-Cycle Model

The development life-cycle model refers to the software engineering methods utilized within the code base to indicate code maturity and readiness. During development, a feature is created and added into the development product. To ensure stability and the ability to show off the development progress to the client and coach, a release of new features has to be scheduled at set intervals which provides a stable base to continue development of the product.

The development life-cycle model implemented for BEPSys 2.0 was the ”dolphin model” (van Solingen [2015]). In this model, the dolphin is a metaphor for the development team, who must resurface often to acquire air. The development team is required to resurface to the client every so often and present the development progress, acquiring desired changes and feedback in the process. This model provides the benefit of continuous validation of features with the client, while also allowing for an iterative development process which allows for change in case of the required addition of features or client dissatisfaction.

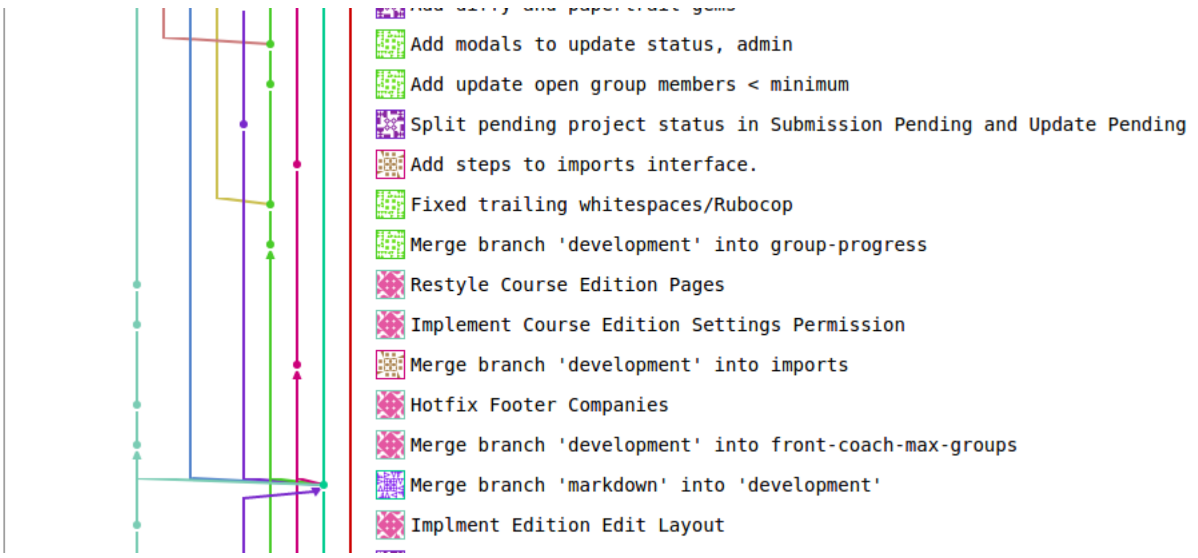
The dolphin model stands in direct contrast with the ”submarine model” (van Solingen [2015]), in which the submarine is a metaphor for the project team. The submarine delves underwater for a longer period of time, and only resurfaces when it is required for a mission. When comparing this to a product development iteration, the metaphor alludes to a project team interacting with a client exclusively at the beginning and ending of a project, never checking in to validate and verify their work.

The decision was made within the project team to work in iterations of one week, as this would provide ten moments of feedback with the client, while also providing enough time to implement a new feature set. The client was also consulted during moments of uncertainty, which was possible due to proximity of the client to the BEPSys 2.0 development office.

4.2. Version Control using Git & GitLab

Version Control refers to the tools utilized for management of code and stable releases. The ”Git” version control management system provides a method to share code among the team and keep all versions of the development code base in sync. Git allows a member to create a ”branch,” an area of code based on an earlier version that would not get the latest updates to allow isolated development, and upon completion of that development task to ”merge” the branch into another stable branch. The process of code reviews and merging can be found in section 3.3.

The ”master” branch acts as the most up-to-date approved version of the software at that moment in development. The master branch would form the basis for a ”development” branch which would be the branch from which all new features and implementations are created. The development branch aimed to be a stable code repository of all new work created during the week’s iteration. All branches for an iteration had to be started from the development branch, to ensure the most up to date code base was given to a member of the team when creating new features.



```
Add modals to update status, admin
Add update open group members < minimum
Split pending project status in Submission Pending and Update Pending
Add steps to imports interface.
Fixed trailing whitespaces/Rubocop
Merge branch 'development' into group-progress
Restyle Course Edition Pages
Implement Course Edition Settings Permission
Merge branch 'development' into imports
Hotfix Footer Companies
Merge branch 'development' into front-coach-max-groups
Merge branch 'markdown' into 'development'
Implment Edition Edit Layout
```

Figure 4.1: An snippet of the revision history of the source code under version control, illustrating how branching and merging were applied.

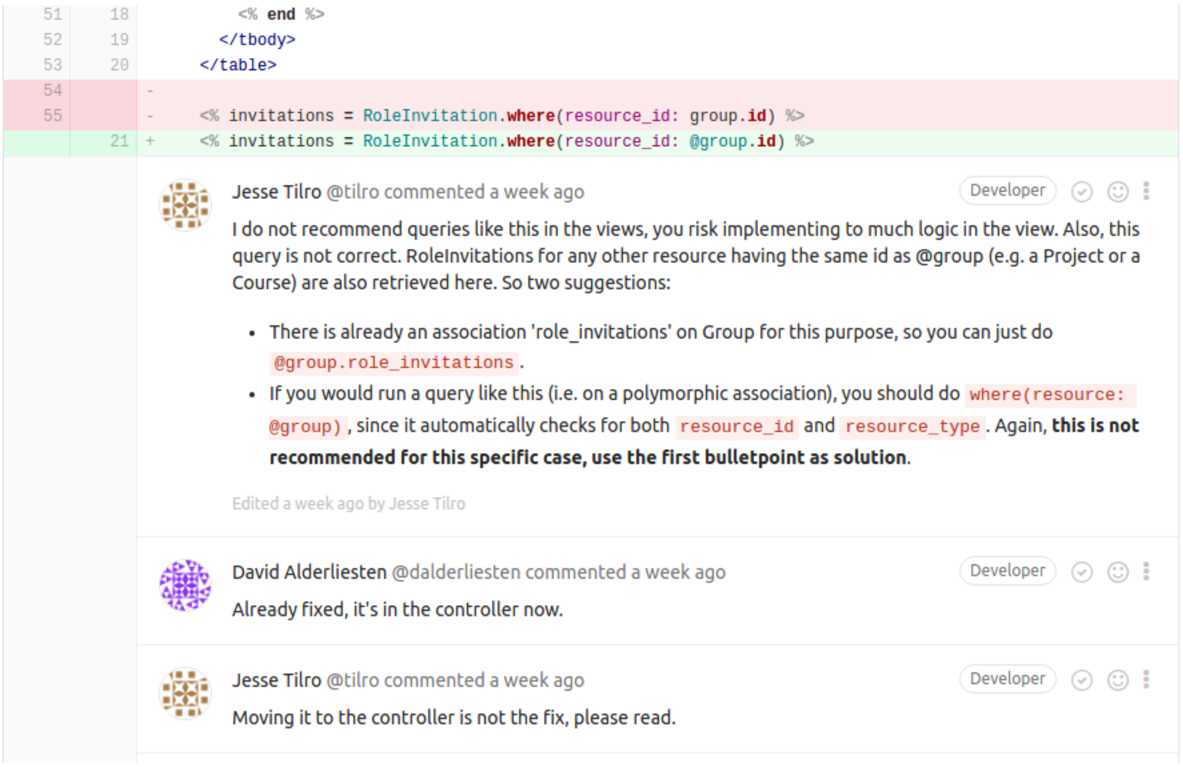
Upon completion of a development iteration, all code within the development branch would be submitted to the client for merging into the master branch. The master branch would therefore continue to be the most stable up-to-date approved branch, and upon completion of the merge a new development branch would be created from the new master.

The repository that was utilised to store the software was hosted on an internal server environment of the TU Delft, as required by the TU Delft’s regulations. The software utilised to manage the repository including merge requests, automated testing, and branching was Gitlab. These tools and environments were all provided by the client, and the utilisation of these tools and environment were required.

4.3. Modern Code Review through Merge-Requests

Merge-requests act as formal validation and verification processes in which a new feature or bug fix has been implemented and wishes to become a part of the stable development branch. A merge request consists of a code review in which at least two members of the team review another members’ code and ensure that all tests pass. A focus is also placed on the code quality, which must meet the required specifications given to the project, and a validation that all aspects of an implementation match the desired specifications for that implementation.

The team required two members to verify each merge request that was created, regardless of size or complexity of the branch under consideration. The standardised process was to review the code in isolation, and comment on any violations of code style or unnecessarily



```
        <% end %>
      </tbody>
    </table>
-
-   <% invitations = RoleInvitation.where(resource_id: group.id) %>
+   <% invitations = RoleInvitation.where(resource_id: @group.id) %>
```

Jesse Tilro @tilro commented a week ago Developer

I do not recommend queries like this in the views, you risk implementing to much logic in the view. Also, this query is not correct. RoleInvitations for any other resource having the same id as @group (e.g. a Project or a Course) are also retrieved here. So two suggestions:

- There is already an association 'role_invitations' on Group for this purpose, so you can just do `@group.role_invitations`.
- If you would run a query like this (i.e. on a polymorphic association), you should do `where(resource: @group)`, since it automatically checks for both `resource_id` and `resource_type`. Again, **this is not recommended for this specific case, use the first bulletpoint as solution**.

Edited a week ago by Jesse Tilro

David Alderliesten @dalderliesten commented a week ago Developer

Already fixed, it's in the controller now.

Jesse Tilro @tilro commented a week ago Developer

Moving it to the controller is not the fix, please read.

Figure 4.2: An example of a discussion taking place in a merge-request.

complex structures within the code. After the code was deemed acceptable, multiple local tests were done. These tests included automated self-written test suites through Jenkins, manual verification of functionality, and automated style offence checking with the utilisation of Rubocop.

4.4. Issue Tracking using JIRA

The tracking of issues and tasks, including the maintenance of the product backlog, was done utilising the JIRA software. JIRA allows issue tracking, assignment, user story logging, and the assignment of priority tasks for bug fixing.

4.5. Task Automation using Rake

A Rails project is bundled and integrated with the Rake gem, a task automation tool, which provides automation for many tasks such as running tests, performing database migrations, and updating dependencies. In addition, custom project specific tasks can be implemented that are integrated with the framework.

5 Technical Considerations

The original version of the BEPSys system was written in PHP, using the CakePHP framework [Bashirieh and Rahbari, 2013]. For the development of its successor the decision was made to discontinue the use of the existing code base, and to instead build the new system from scratch. It was decided upon to implement the new system in the Ruby language, using the Rails framework. This chapter outlines the reasons for starting design and development from scratch and utilising a different language and framework. The options for deployment and scaling with the selected technology are discussed, along with the decisions related to the migration of data in the old BEPSys system to the new iteration.

5.1. Starting Over

After an analysis of the code base of BEPSys 1.0 and taking into account the elicited requirements, it was concluded in conjunction with the client that starting the design and implementation of BEPSys 2.0 from scratch was in the best interest of both stakeholders and developers. Starting over increased the chances of meeting the requirements regarding the product quality, and allowed the development team to provide a greater guarantee of removing and avoiding (security) flaws that were present in the current implementation.

5.1.1. Improving Maintainability

Maintainability is considered to be one of the eight main characteristics that make up the product quality of a software system, as standardized in ISO/IEC 25010. Aside from its importance with respect to software quality, there was an additional emphasis on maintainability for the BEPSys 2.0 project. As elaborated upon in the requirements, the collection of stakeholders for the system was diverse, yielding equally diverse requirements. These requirements are prone to change, since the purpose of BEPSys is to support the TU Delft's administration of a course approval and management process that is subject to intermittent changes. An example of such a change is the admission criterion utilised for students to enrol for the bachelor project, which were indicated to be subject to change in future versions of the course. These changes could vary from minor entry point changes (where a student would have to attain a certain amount of European Credit Hours to gain entry) to entire course requirement changes (all classes from the first and second year would have to be passed before enrolment was accepted). The current BEPSys application lacks support for these long-term changes.

The quality of the current product did not meet the expectations either. The number of 'hotfixes' and 'urgent fixes,' alongside the functional changes in the commit history after the delivery of the initial version of the system, reflect this. A more flexible and more generalized implementation is urgently required to ensure a more maintainable system. The fresh start and new framework are thus aimed at improving the product quality, especially with respect to maintainability.

The BEPSys application is currently implemented without the employment of third-party software dependencies. Software packages for the PHP language could have been managed by utilising the Composer, for which a dependency file recorded in the application. In addition, specific plugins for the CakePHP framework are available, providing off-the-shelf functionalities for CakePHP projects. The third-party code that is included, consisting primarily of JavaScript assets, are integrated directly into the code base without any dependency management, which leads to a less maintainable software repository that is vulnerable to security issues due to a lack of automatically managed updates.

The lack of version controlled dependencies - or rather the lack of dependencies at all - yields implementations of subsystems that can be described as “reinventing the wheel.” This refers to writing code to solve a problem that has already been solved by more experienced developers and made available in some form, usually as open source software or libraries. An analysis of the initial code base revealed that the aforementioned redundant implementation was rife within the current BEPSys application.

The user authentication subsystem in BEPSys is handled by the ”UsersController” component. This component is responsible for handling actions like the login, logout, register, reset and deletion of user accounts. This is implemented using the Authentication component of the CakePHP framework. However, this component provides a rather low-level interface, therefore rendering the controller actions cluttered and having a relatively high complexity. This might lead to security vulnerabilities, such as the ones that were found in the password reset and account confirmation mechanism during an audit of the original code base.

5.1.2. Preventing Flaws in BEPSys 2.0

Implementing a subsystem such as user authentication is a time consuming and risky endeavour. When there are reusable open source dependencies that have proven themselves to be functional and secure in practice, utilisation of these dependencies is preferable in the new application.

As mentioned in the research report, the new user authentication subsystem will rely on the Ruby gem ”Devise”. This gem is contributed to by many developers, and is rich with features related to the required account system requirements. Unless specific customization is required, the application does not need controllers for the purposes of user authentication: these are embedded into the gem. This means zero lines of controller code in the code base, as opposed to the 321 lines of unreliable authentication code in the original PHP version of the system.

The undesired insertion or update of values for certain attributes will be prevented using “Strong Parameters”, which is utility in Rails to filter only allowed key value pairs from the parameters sent along with HTTP-requests. This prevents users from updating values in columns of a database table for which they are not authorized.

In addition, the ”CanCanCan” gem is to be employed for structured user authorization, which previously was either done using manual role checks all over the system, or not done at all. Using this gem, user authorization is centralized in an ”Ability” class, in which permissions to create, read, update and destroy models - or specific model instances - are dynamically determined for a given user instance.

The research report, found under appendix B, lists many other evaluated and selected modules (mainly Ruby gems) that not only assist in quality assurance, but also significantly increase the speed of the development process, mitigating the overhead of starting from scratch.

5.2. Rails Framework

The following section outlines the decisions that led to the selection of the Rails framework for the development of the new BEPSys 2.0 application.

5.2.1. Model-View-Controller

Rails implements the Model-View-Controller (MVC) pattern, which is a compound software design pattern that can be decomposed into three more fundamental design patterns: the Strategy, Composite and Observer design pattern. The purpose of this compound design pattern is to strictly separate key responsibilities in a system over different classes of components (i.e. controllers, models and views) of the system.

5.2.2. Representational State Transfer

Representational State Transfer (REST) is a software architectural style that emphasizes scalability of component interactions, generality of interfaces and independent deployment of components (Fielding and Taylor [2000]). By imposing constraints on the software architecture, certain quality attributes such as performance, scalability, simplicity, modifiability, visibility, portability, and reliability can be improved. For these purposes, in contrary to the original implementation, BEPSys 2.0 is intended to provide a RESTful interface. Rails supports this, as exemplified by the default resourceful routes.

5.2.3. Language and Framework Preferences

Ruby as a language was selected by the development team due to its aim to centralize programmer productivity, such as by offering an intuitive, readable, high-level syntax. Within the context of the Rails framework, this is especially noticeable in the different internal domain specific languages (DSL) that are offered by tools integrated with the framework. The language allows the focus to be placed upon conforming to the functional requirements as opposed to syntactical enforcement, speeding up the development process.

The chain of tools integrated with the Rails framework provides developers with quick ways to manage the project. This concerns command line utilities such as ”Rake” for task automation (such as running tests and database migrations), Rails for generation of projects and code scaffolds and Capistrano for fully automated deployment.

The philosophy behind the Rails framework includes two main principles, namely “Don’t Repeat Yourself (DRY)” and “Convention Over Configuration”. The implications of the former principle is obviously a more maintainable code base. However, the latter principle has the negative side effect of a rather steep learning curve. Nonetheless, a month before the start of the project the developers started learning Ruby on Rails through readily available online tutorials and documentation, mitigating the effect of the hurdles these numerous conventions cause for novice Rails programmers. The benefit of this principle is ultimately that building applications is faster because of the negligible overhead of configuration throughout the application. The Ruby and Rails communities are also helpful, providing many forms of detailed documentation, and many reusable modules in the form of Ruby gems.

Rails also comes integrated with the MiniTest testing framework, extended with an abstraction layer. This testing framework allows for structured testing of all the different classes of components of the rails application, at different testing levels (such as unit and integration tests). Alternatively, frameworks like RSpec and Cucumber can be used with Rails as well. With the stable release of Rails 5.1[1][2] on 21 April, 2017, full integration of system tests (i.e. browser-automated acceptance tests) were introduced - right in time to be utilised for the

[1]`http://weblog.rubyonrails.org/2017/4/21/Rails-5-1-rc2/`

[2]`http://edgeguides.rubyonrails.org/5_1_release_notes.html`

project. The value of these tests and the way they were employed are further discussed in section 8.1.2 on page 30.

5.2.4. Justifying Maintainability

It was discussed with the client that students involved in the future maintenance of BEPSys are not likely to be familiar with Ruby on Rails due to a lack of its existence within the bachelor curriculum offered at the TU Delft. The involved stakeholders have little to no experience with Ruby on Rails, either. Therefore, one could argue that the choice to employ Ruby on Rails is interfering with the maintainability of the system. The development team believes that this argument is ultimately unsustainable. First, PHP is not part of the computer science bachelor curriculum either, nor is the framework CakePHP, thus rendering it equally likely that students are unfamiliar with the currently employed technology. This argument can be generalized to many web technologies, frameworks, and languages (such as Django, Meteor, Laravel, Symfony, Zend, Sinatra, WordPress, Elixir, Phoenix).

Moreover, bachelor students in computer science at the TU Delft are taught about relatively abstract concepts on programming languages and software engineering methodologies, ensuring the ability to quickly familiarize with new software technologies. Therefore, one could argue that being introduced to a different language and/or framework should not pose any unreasonable challenges and thus should not introduce any significant hurdles with respect to maintainability. The development team was required to teach themselves the new frameworks, and the resulting product shows the ability to learn the language and framework, and proceed to develop with it.

5.3. Migrating data to the new production environment

The migration section outlines the research and decisions that were made which led to the decision to not migrate current BEPSys data to the new application.

5.3.1. Technical Considerations

The current implementation of BEPSys stores data objects with fields which were determined to be relevant at the time of implementation. Such data includes student records (name, netID, student number, date of birth), project information (name, company, contact information), and other minor data objects that are required for the system (users who have a special role, such as the course coordinator or counsellor). As outlined in section 5, the design of BEPSys 2.0 aimed to generalize all data objects (splitting company information from projects to allow for a single company entity to have multiple projects) and to create flexible parameters (allow multiple courses, forms, rule sets). Due to the restrictions of the previous data model, alongside the requirements for flexibility and shift of scope focus, it was decided that migration of data to the new production environment would not be done.

5.3.2. Client Preferences

After multiple discussion moments with the client, it was concluded that the client was willing to accept both a data migration or no migration. Due to the desire to make BEPSys 2.0 flexible, and the lack of support for many of the options within the data model, the client accepted the proposal to not migrate any data over to the new production environment. The client did insist that information would be provided to all companies currently in the BEPSys system about the incoming change, including the ability to fetch their current project data before the removal of the old system.

5.3.3. Migration Procedure

Since it was decided that no data migration to the new production environment would take place, no procedure was developed. It was agreed upon to inform all companies in the existing BEPSys database of the incoming application change, and to provide them with the opportunity to re-register in the new system. No further migration related actions or procedures took place during the development of BEPSys 2.0.

6
Design

The design section outlines choices that have been made related to database design, data modelling, software architectural decisions, and front-end design choices. Section 6.4 on page 26 outlines group formation theories that were researched to aide in the formation of groups for projects.

6.1. Decisions

The methodologies section aims to clarify the reasoning and processes behind decision made for the project. The original version of the BEPSys system was written in PHP, using the CakePHP framework [Bashirieh and Rahbari, 2013]. For the development of its successor the decision was made to discontinue the use of the existing code base, and instead build the new system from scratch. Additionally, it was decided to implement the new system in the Ruby language, using the Rails framework. This section aims to outline the reasons for starting design and development from scratch and utilizing a different language and framework. The options for deployment and scaling with the selected technology are also discussed in this section.

6.2. Modelling

To ensure that future changes and requirements can be added into the new BEPSys system, and to ensure that all specifications and requirements are met, different models were created as a guideline for the back-end. The flows of different processes within the system were modelled using business process model and notation (BPMN). The database was designed using an EER diagram.

6.2.1. Process Flow Modelling

Figure 5.1 displays a process flow diagram which describes the process of the creation of project. This process is complicated due to the ability for multiple company entities to exist, as they can be created on a person-by-person basis. The administrator has the option to merge companies such that all projects from those companies are seen under the name of one of the merged companies, creating an overview of a company with multiple users.

Figure 5.2 displays the process of joining a group for a project. After a coordinator has approved a user to participate in a course (edition), a user can request to join a group or can be invited by the group leader to join the group.

Figure 5.3 displays the process of enrolling in a course (edition), including the permission that must be granted to the user from a counsellor.

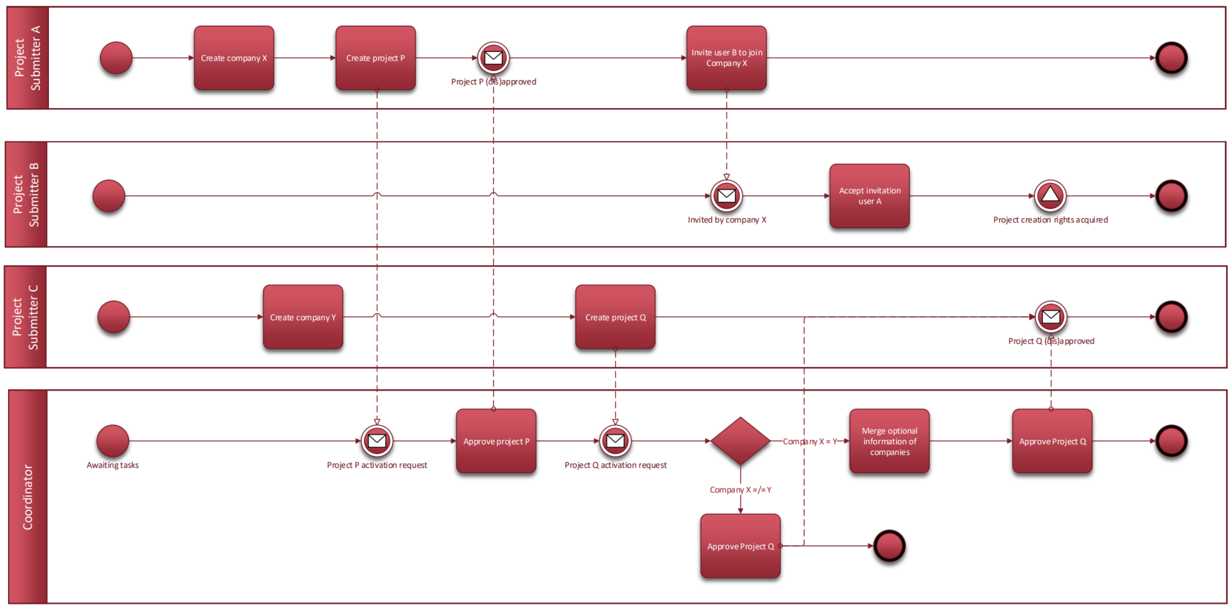


Figure 6.1: Process flow model for project creation

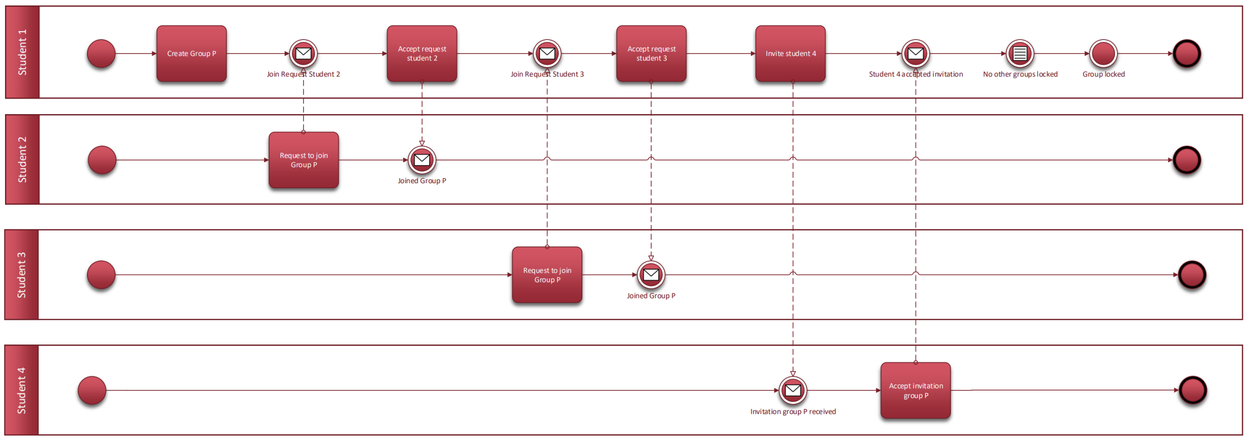


Figure 6.2: Process flow model for group joining

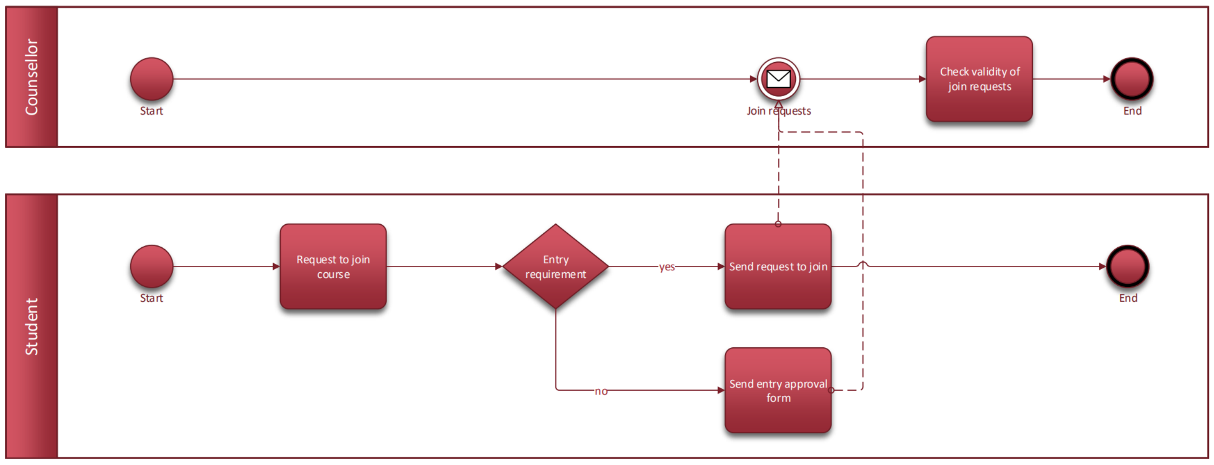


Figure 6.3: Process flow model for course enrolment

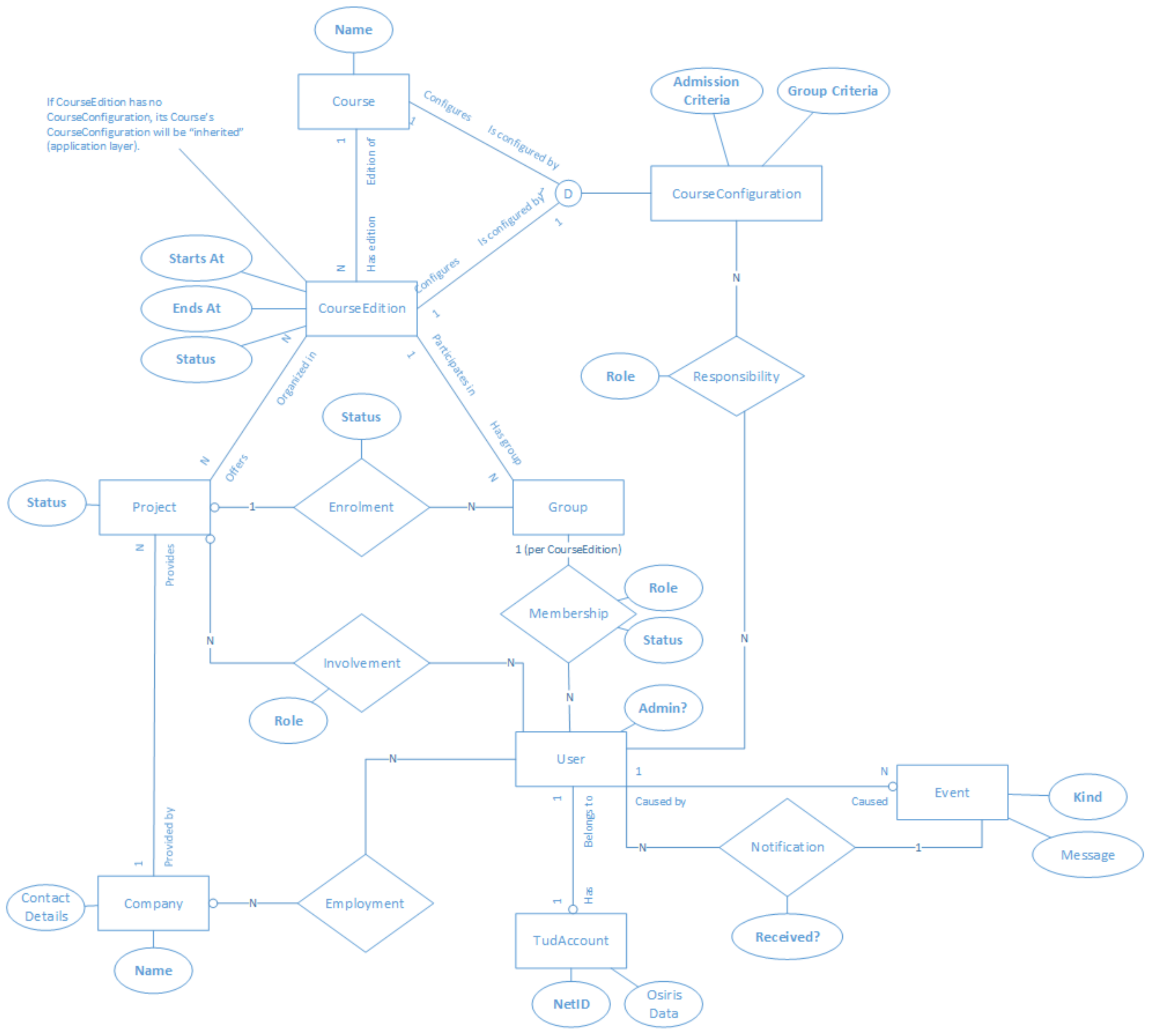


Figure 6.4: Initial data model

6.2.2. Data Modelling

Data modelling refers to the design of the data and the interactions that take place between objects of data. To describe the initial data model, a conceptual data model was designed. To comply with the requirements, this model primarily expands the data model of BEPSys 1.0 and aims to conform to it wherever possible. Extensions to the model include separating student groups as an entity, separating companies as an entity, adding support for course editions under a course to separate active time periods of courses, and a data model for the system notifications with support for logging. Figure 5.4 shows the initial data model.

The ubiquitous "role" and "status" attributes were added to relations that were already in place as a result of the required associations between different entities (regardless of the user role allocation mechanism). This makes for efficient reuse of existing relations for role allocation purposes. In order to scope user roles to entities which are not directly associated with the User, a more generic role allocation mechanism was put in place revolving around the Role entity which uses Rails polymorphic associations in the application layer to be associated with different entities.

6.2.3. Schema Model

The schema model outlines the interactions between data objects within the system. The schema model served as a guiding diagram to ensure that all individually developed components would interact correctly as per the agreed upon model. Due to the size, the complete schema model can be found under appendix A.

6.3. Front-end

This section outlines the design process behind the front-end's graphical and layout design, including wireframes, layout concepts, and interaction design.

6.3.1. Wireframes

The development of a front-end in an iterative manner is an inefficient and slow process, as the design would have to be developed, programmed, and discussed for each iteration. A decision was made to develop a set of wireframes in Balsamiq's Mockups[1] application, which provide a somewhat stylized concept within a drag-and-drop interface for front-end design. The wireframes would assist in developing the main look and feel of the application.

6.3.2. Layout

The main focus of the layout was to provide a singular style for all aspects of the application, meaning that menu's and information display was identical regardless of role, data object, or location. The general template of the layout was to match that of the TU Delft, meaning a full width navigation bar at the top of the page and a full width footer with a navigational sitemap and logo at the bottom of the page. The content of each individual page would be displayed between these two objects. On the course page, an overview will be given of all courses that exist in the system. When selecting a course and selecting a course edition, a list of projects would be displayed. An impression of this layout for the course selection is provided by the wireframe in figure 5.5, and an impression of this layout for the project selection of a course edition is provided in figure 5.6.

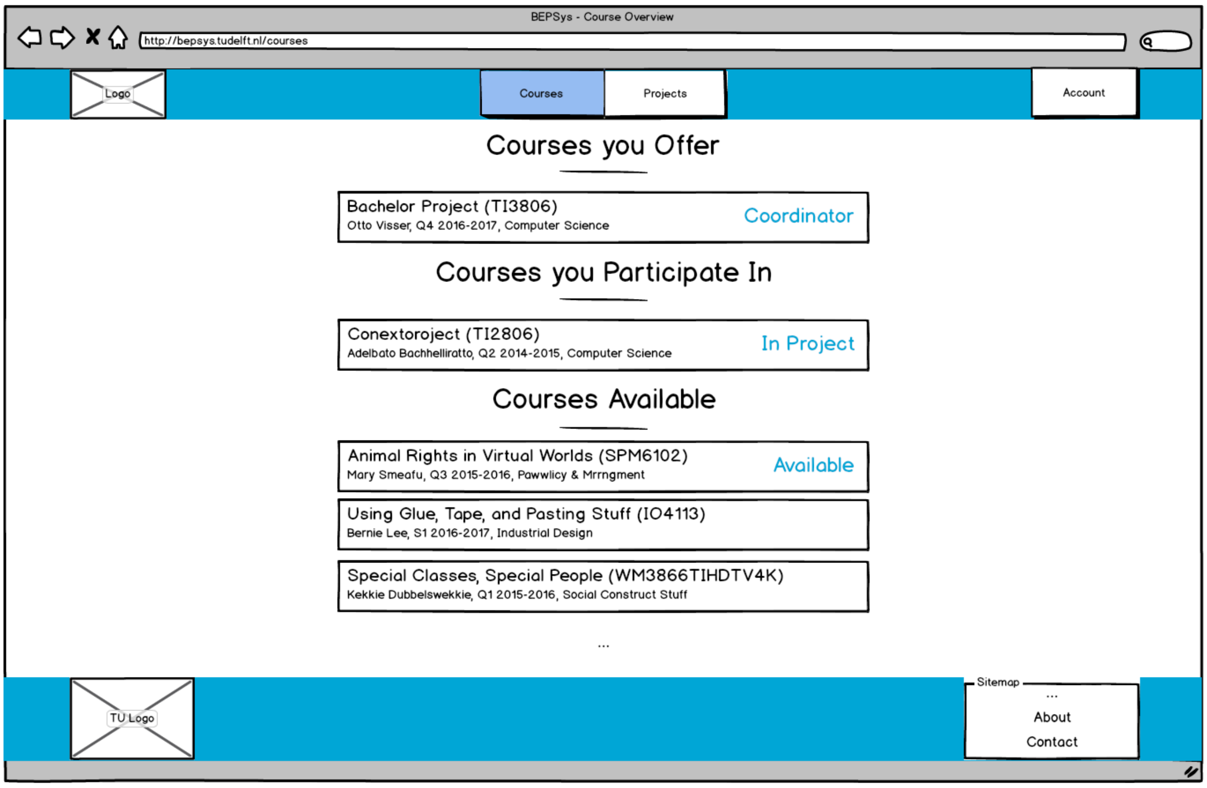
Since each data object in the application would require multiple views (such as projects, which require views for information, groups, and management), it was decided to implement a contextual menu on each page that contained a data model with depth. A wireframe layout of such a layout can be seen in figure 5.7. Note the navigation bar in figure 5.7 that is located under the course title and edition period, which acts as a central navigation method through objects with the level of aforementioned depth that require multi-level navigation.

Utilising these mock-ups, a general look and feel to the BEPSys 2.0 application was developed and agreed upon. The front-end would rely on a simple interface with full-width elements to distinguish navigation from page content.

6.3.3. Colour Scheme & Design

The colour scheme for the front-end of the application was predetermined by a client requirement. The client expressed a desire for the new application to adhere to the TU Delft's colour and style guidelines ("huisstijl"), which dictate colouring for multiple elements and accents, as well as header and display properties. The navigation and footer colours were determined to be blue as dictated in the style guide, as to mimic the TU Delft's website style as much as possible for a uniform user experience across platforms. All effects and animations are focussed

[1]https://balsamiq.com/products/mockups/



Courses you Offer

Bachelor Project (TI3806)
Otto Visser, Q4 2016-2017, Computer Science
Coordinator

Courses you Participate In

Conextoroject (TI2806)
Adelbato Bachhelliratto, Q2 2014-2015, Computer Science
In Project

Courses Available

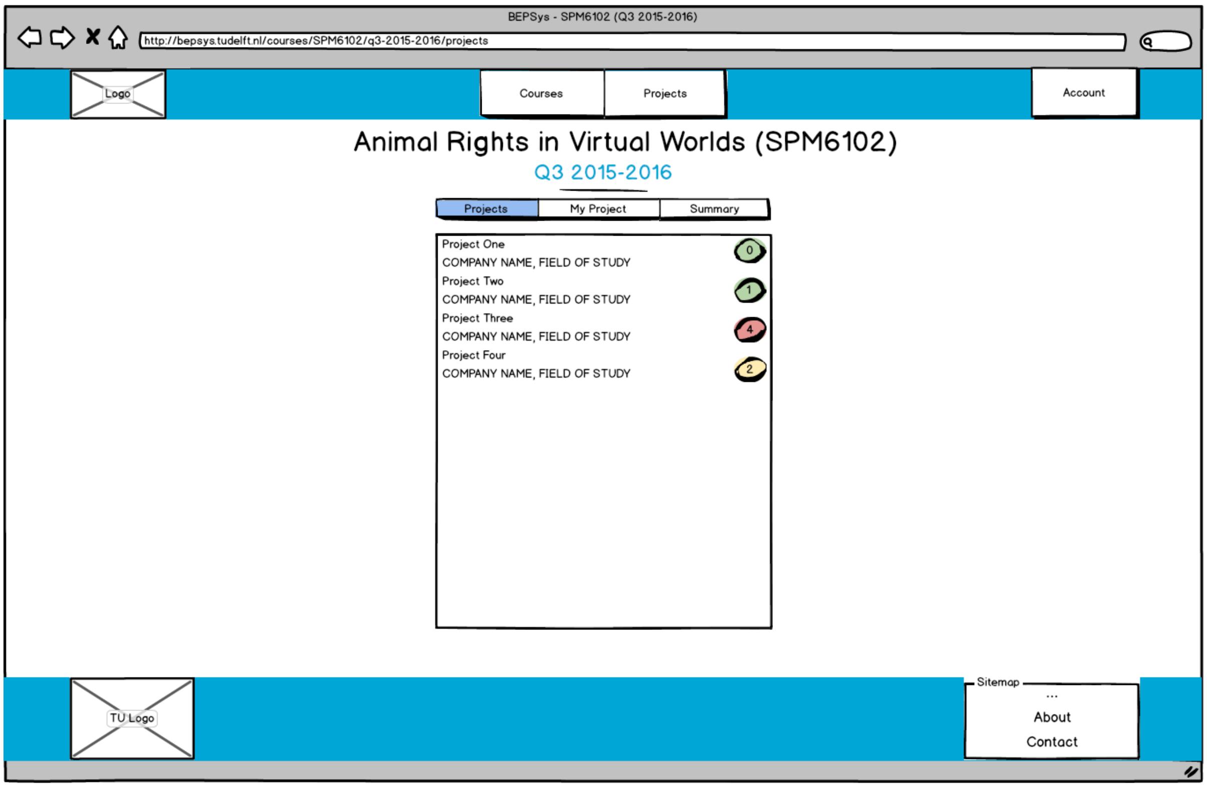
Animal Rights in Virtual Worlds (SPM6102)
Mary Smeafu, Q3 2015-2016, Pawwlicy & Mrrngment
Available

Using Glue, Tape, and Pasting Stuff (IO4113)
Bernie Lee, S1 2016-2017, Industrial Design

Special Classes, Special People (WM3866TIHDTV4K)
Kekkie Dubbelswekkie, Q1 2015-2016, Social Construct Stuff

...

Figure 6.5: Wireframe layout for course selection


BEPSys - SPM6102 (Q3 2015-2016)
http://bepsys.tudelft.nl/courses/SPM6102/q3-2015-2016/projects
Logo
Courses
Projects
Account

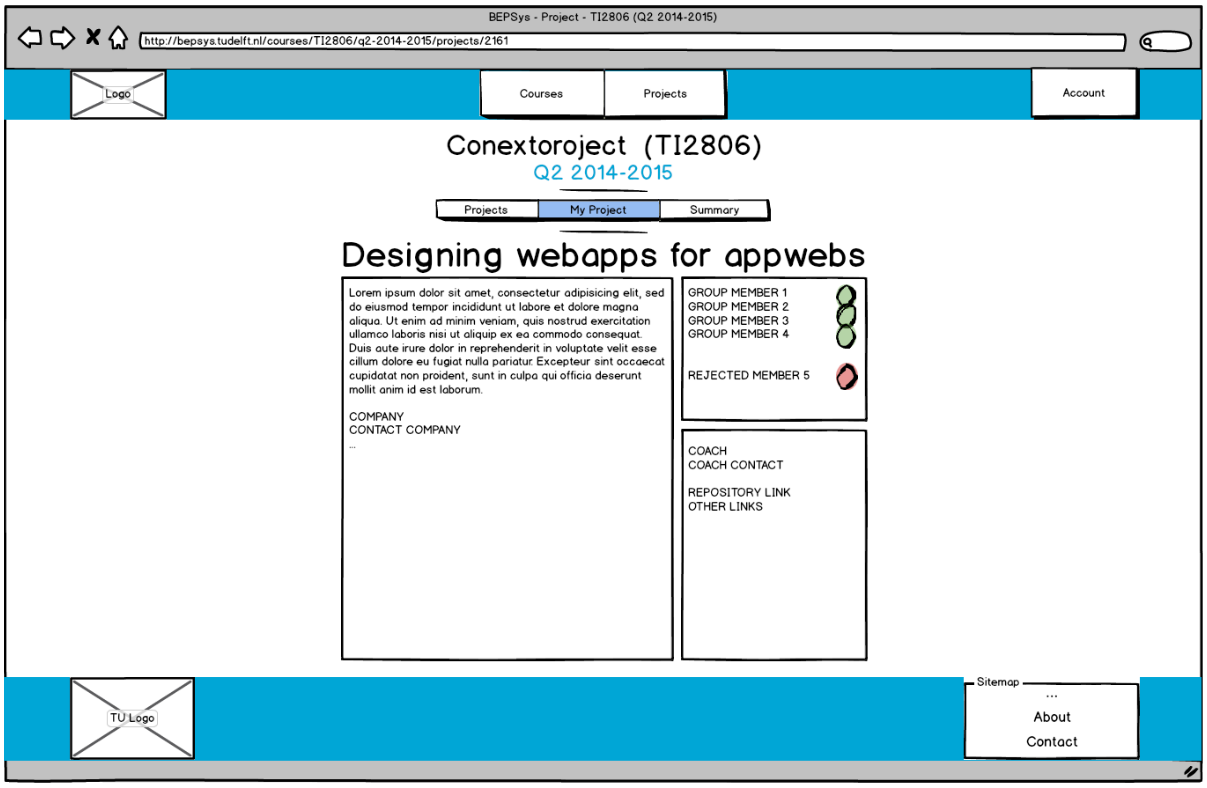
Animal Rights in Virtual Worlds (SPM6102)
Q3 2015-2016

Projects
My Project
Summary

Project One
COMPANY NAME, FIELD OF STUDY
0
Project Two
COMPANY NAME, FIELD OF STUDY
1
Project Three
COMPANY NAME, FIELD OF STUDY
4
Project Four
COMPANY NAME, FIELD OF STUDY
2

TU Logo
Sitemap
...
About
Contact

Figure 6.6: Wireframe layout for project selection within a course edition



Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

COMPANY
CONTACT COMPANY
...

Figure 6.7: Wireframe layout for project information

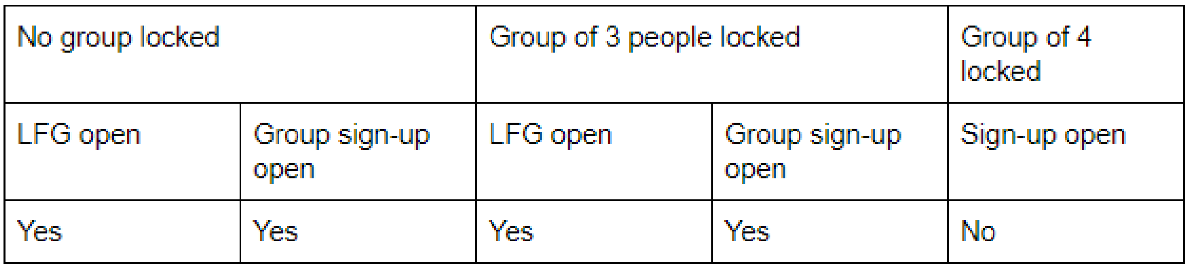
on matching those found on the TU Delft's website, including the sidebar navigation discussed in section 5.3.2. A table with all required colours and properties is found in appendix C.

6.4. Group Formation

Starting a project in the BEPSys application requires a group. In this section an assessment is made of different methods of group formation and outlines the decisions that led to the design of groups and group formation in BEPSys 2.0.

6.4.1. Formation Aspects

Before looking at the benefits and drawbacks of a group formation system there are two ways of dealing with group formation that can be distinguished [Cantador and Castells, 2012]. One method is "intentional" group formation in which users can create their own groups. The alternative refers to "non-intentional" group formation which requires an automatic identification system in which groups are created automatically. Non-intentional systems can be very useful if one wishes to solve difficulties in creating groups that have certain requirements. The "optimal" group could suggest group members that all have the same talent which makes the group very strong in one area of expertise. A good group could also suggest group members with different talents and specialisations to ensure together the group is divergently talented. To form diverse groups there are associated optimization problems where the aim can be to maximize group diversity and minimize difference among the groups. According to [Mahenthiran and Rouse, 2000] it can also be said that when people choose their own group they will have less "slackers", which is defined as people who are graded somewhat lower than the average of the group grade due to the inability to perform within the group. The group performance and satisfaction can be enhanced when people work together with friends rather than when they are allocated together with other people.



No group locked		Group of 3 people locked		Group of 4 locked
LFG open	Group sign-up open	LFG open	Group sign-up open	Sign-up open
Yes	Yes	Yes	Yes	No

Figure 6.8: Group status business logic table

6.4.2. Group Formation Design

It was decided upon to go for an intentional group formation system where a person can choose to join a project. In the system, a group size of 3 to 4 students is taken into account as per the wishes of the client. The minimum and maximum number of group members can be configured by a course coordinator for each individual course edition. The grouping system has three phases: open, locked and approved. In an open phase any student is allowed to sign up for a group or join the project-based ”looking for group” list. When a group reaches a minimum of 3 interested people (or the minimum amount of people for a group), the system locks that group to be considered for the project. People can still sign up for other groups, but when other groups reach 3 members then they will be placed on a waiting list for the project. The leader of the locked group has the option to finalize the group after which no more people can request to join the group. The sign-up procedure is then closed for everyone for that individual project and the locked group is selected. The maximum number of groups that can be accepted for a project will be configurable per course edition.

The system utilises a “First Come First Serve” (FCFS) procedure in which the first group to reach 3 members (or the minimum amount of members for a group) will be locked first. In the locked phase the waiting list will also have a FCFS approach which suggests that the first group on the waiting list will get locked in the case of the current locked group getting rejected or opting not to do the project. Groups have a “looking for team mate”-option where it will be visually clear that a group is looking for extra group members. Other users can simply click to join the group after which a request is sent to the group leader.

A locked group will get the option to commit to a project. Every group member should commit individually to the project in the locked phase to ensure absolute certainty to commit to the project. This prevents the issue in which the group leader can accept a new member and commit to a project without the others of the group agreeing with the decision of the leader. It was decided to let group leaders commit on their own as it will be reviewed in a later iteration of the development process. When a group has committed to a project, a final check must be done by the project submitter. The project submitter will have the option to accept the group or not. If the group is not accepted then the sign-up phase is open again and the current locked group is unlocked and tagged as “rejected”. However, if the group gets accepted then the project can finally start its process.

An outline for the group phases and business logic as outlined in this section can be found in figure 5.8. Note that ”LFG” refers to the interest lists, and is an abbreviation for ”Looking for Group,” which was the original title of the interest list system.

7 Implementation

The implementation section describes the developed application and its back-end, front-end, and deployment environments. The focus therein lies with changes between the design and actual development phases. It also outlines attempted implementation of technologies that would not make it to the final application due to time or resource constraints.

7.1. Back-end

The back-end section covers implementation related details and issues that are related to Ruby on Rails and the database environment.

7.1.1. Implemented Data Model

The data model and associated schema as designed during the research phase was implemented in the application. A copy of the schema can be found in appendix A. Minor attributes such as course images and company logo support were added after the design phase, but all changes made were minor and encompassed a scope of a few variables added to certain data models.

7.2. Front-end

The front-end section covers implementation related details and issues that are related to the layout, design, and interface implementation that is shown to the users of the application.

7.2.1. Administration

After completion of the wireframes during the design phase, it was decided upon to have a physical split between the front-end for users, and the view for administrative tasks. This resulted in an administrative panel which did not adhere to design guidelines and was focused entirely on providing simple access to all data objects and administrative tasks in the application.

7.2.2. Custom theme for the Bootstrap layout framework

The web interface for the front-end and administrator panel was built using the Bootstrap layout framework[1]. The Bootstrap framework accelerates front-end development by providing many general, pre-built components and layout utilities. In addition, these components and utilities are responsively styled, which eases the process of making the interface available on different

[1] `https://getbootstrap.com/`

screen sizes (such as mobile phones and tablets). In order to meet the front-end design guidelines that have been introduced in the preceding chapter on ”Design,” a custom theme was developed for the Bootstrap framework using the colours as outlined in appendix C. In order to make this theme maintainable and reusable, the Syntactically Awesome Stylesheets (SASS[2]) version of Bootstrap was used. SASS is a CSS pre-compiler and syntax extension that allows for colour variables across CSS files. This allowed the default Bootstrap styling to be modified and tweaked by simply editing SASS variable values.

7.2.3. Responsiveness

Due to the Bootstrap implementation as outlined in the previous subsection, the design of both the administration panel and the front-end is entirely responsive. The tweaking of the responsiveness, including verification, was done through manual testing. All devices are supported, although phones with a screen diameter under four inches will have issues displaying certain pages. It was decided to not take those devices into account, as they are a decreasing trend and are not found in phone trends today. The major interest and usage group for BEPSys 2.0 consists of students, and a common trend is that students have smartphones with a screen diameter and resolution that exceeds these parameters.

7.2.4. Asynchronous Interface

Research was done into the possibility of an asynchronous interface, in which data is displayed instantly based on live updates and instant reactions to user input. A possible component to support this feature was React.JS, a Javascript library which allows for simple implementation of a reactive interface. Whenever a user performed an action related to group management or administrative tasks, the page would not have to reload or be refreshed manually, but all data on the page would by dynamically adjusted through the utilisation of listeners. Due to time constraints, it was not possible to implement this library into the BEPSys 2.0 application.

Due to the added value of such a library, and due to the interface having been designed with a library such as React.JS in mind (all components in a view are separated and independently loadable), the development team wishes to advise any future programmer to implement such a reactive library into the BEPSys 2.0 application.

7.2.5. Single Sign-On

The TU Delft provides each staff member and student with an official TU account, which can be utilised to sign in to authorised applications using their ”single sign-on” system. The incoming iteration of this system utilises a security assertion mark-up language (SAML) which has been integrated into the BEPSys 2.0 application.

Acquiring authorisation to utilise an application with the single sign-on environment requires a formal request to be sent to the TU Delft’s authority for personal and sensitive data. This request was sent in the second week of the project, and a resubmission occurred in the sixth week of the project. Both requests were met with silence, meaning the actual link with the single sign-on system and the BEPSys 2.0 SAML integration did not take place. This was entirely due to a lack of responsiveness on the part of the TU Delft’s central services, and there was no possible alternative course of action to be taken by the development team.

For temporary purposes, an integration was made with Okta[3] to simulate logging in through the single sign-on. A temporary registration page for student and internal staff members (which would not be required if the single sign-on environment was active) was added to temporarily circumvent this issue.

[2]`http://sass-lang.com/`
[3]`https://www.okta.com/`

8 Testing & Quality Assurance

The process of testing is an important complement to the software development life-cycle, especially when focussed on improving and assuring the quality of the software product. Furthermore, testing is context dependent. Therefore, a specific testing strategy and policy were constructed for the testing process of the web information system. This chapter discusses the technologies and methods involved in the automation of software testing for verification purposes, how acceptance testing was applied for validation purposes, and the studies that were employed to improve the human computer interaction aspect of the system.

8.1. Software Verification

Software verification amounts to confirmation by examination and through provision of objective evidence that specified requirements have been fulfilled (Kaner and Fiedler [2013]). Using dynamic and static testing, and different testing techniques at different levels, the system was verified to meet the elicited requirements as documented in chapter 3 on page 4.

8.1.1. Dynamic Testing

Within the BEPSys2 application are two main object types that were tested: models and controllers. Test Cases were deemed most suitable in testing individual methods found in models. Testing these methods does not require any integration tests to be run, as most methods can be tested by providing a simple input value and checking the given output to an expected output.

Integration testing was found most useful when testing controllers. Controllers contain a lot of redirects to different URL addresses, and many asserts that validate whether a website page was successfully loaded or was showing a redirect or error page. The IntegrationTest[1] class in Rails provides a convenient list of assert methods which support these test cases.

Rails also provides a simple manner of providing test data. Mock data for tests called fixtures[2] can be easily stored in a YAML file and called from the appropriate test files.

8.1.2. Browser Automated Testing

Since Rails version 5.1, system tests are available natively. Based on the Capybara gem and fully integrated with the framework, system tests use an interchangeable module for browser-automation (using specialized software such as the Selenium Webdriver). This way, the system can be automatically tested against acceptance criteria, including Javascript functionality

[1]http://api.rubyonrails.org/v5.1/classes/ActionDispatch/IntegrationTest.html
[2]http://api.rubyonrails.org/v5.1.1/classes/ActiveRecord/FixtureSet.html

and adherence to visual criteria for the layout, using browser-automation techniques. From within these tests, the entire application and framework API's are available. This means that tests can launch the browser, perform some actions through the web interface in the same manner as a real user, and validate whether relevant records in the database are being manipulated as expected (an example of this could be filling in input fields within forms and pressing submission buttons to validate object creation and error handling).

Additionally, system tests can run on a headless server, such as by using a virtual frame buffer. Tests were configured to run inside en X virtual frame buffer (XVFB) on the Jenkins continuous integration server (storing screenshots in case of failed tests to allow for manual checking to resolve errors and test failures).

A major contributing factor for employing system tests into the BEPSys 2.0 application is the fact that they provide reliable regression tests. Due to strict security regulations for the system, dependencies had to be kept up-to-date automatically. Dependency updates may often break functionality, especially in the case of major updates changing a crucial API. Running system tests in continuous integration assists in assuring that a succeeding build on the continuous integration server yields a correctly functioning build in the production environment.

8.1.3. Test Coverage Analysis

The development team wanted a metric related to the percentage of tested objects and classes within the code base. From the available coverage metrics a decision was made to utilise line coverage. A code coverage analysis tool for Ruby, SimpleCov[3], was utilised to determine the percentage of line coverage in the code base. This tool was used as a means to see the test coverage at a given point in time. The team generally aimed at achieving a minimum line coverage of 80% for models and controllers. Line coverage for view objects, which rely on HTML and CSS, were not created.

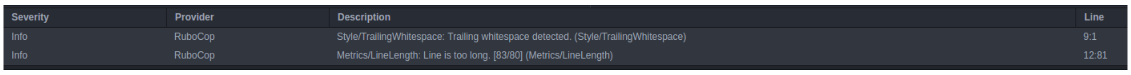
8.1.4. Static Analysis using Rubocop

To ensure uniformity in code style (indentation, tabbing, whitespace, bracket placement), and to detect specific types defects in the system, static analysis was used. The static analysis tool Rubocop[4] was used for this purpose. Rubocop checks for violations of rules that are organized in rule sets called ”Cops”. General Ruby language cops, as well as a specific cop for the Rails framework, are available.

The Rubocop configuration was left at the strict default settings, which are in line with the Ruby Style Guide constructed and maintained by the community[5]. Some exceptions to specific rules for specific files were added to the configuration only in cases where the rule was considered not applicable, along with an elaborate motivation. These motivations have been left in the code base for other developers to see in case of maintenance or changes.

In order to integrate static analysis into the development workflow, a linter plugin for the used text editor Atom was installed locally. This plugin ensures that Rubocop runs in the background on opened files, and warnings are displayed inside the editor whenever violations are introduced. An example of such local Rubocop violations can be seen in figure 7.1. Rubocop was also configured to run on the continuous integration server as part of the build and testing process. In order for the CI build to pass, the code would have to be clean from violations. A build would fail if a Rubocop style violation took place.

[3]`https://github.com/colszowka/simplecov`
[4]`https://rubocop.readthedocs.io/en/latest/`
[5]`https://github.com/bbatsov/ruby-style-guide`



Severity	Provider	Description	Line
Info	RuboCop	Style/TrailingWhitespace: Trailing whitespace detected. (Style/TrailingWhitespace)	9:1
Info	RuboCop	Metrics/LineLength: Line is too long. [83/80] (Metrics/LineLength)	12:81

Figure 8.1: An example of a local rubocop violation

8.1.5. Security Testing

Automated testing was utilised to identify and eliminate security vulnerabilities. Refer to section 10.3 on page 37 for details about the tools and techniques employed for these purposes.

8.1.6. Continuous Integration using Jenkins

To ensure that tests were executed within a clean environment and without the ability to manipulate tests locally, an automated test system was utilised. This system, Jenkins, runs all system tests and other automated tasks (such as Brakeman and Rubocop) to ensure that the submitted code is up to the agreed standard within the development team. The utilisation of an automated system for this process ensures that tests and implementations that are unstable or heavily dependent on local changes can be caught, effectively removing submission bias and manipulation from the testing process.

8.2. Software Validation

Software validation concerns the evaluation of the system to check whether the user needs and requirements are met. User Acceptance Testing (UAT) was primarily used to support the software validation. Developers cannot identify all quirks and bugs on their own, thus requiring user testing to locate additional bugs and issues.

Four stakeholders representing the users of the system were asked to perform UATs: a student, a study counsellor, a coordinator, and a staff member. The test sessions with these stakeholders were prepared by setting up test scenarios in which the user had to solve the assignments given by the developers. The documentation and instructions for these scenarios can be found in appendix D.

During these scenarios the user was requested to think out loud, allowing the developers to document possible issues or bugs that the user came across. Users were not permitted to ask anything regarding the assignments unless they could no longer make any progress and got stuck. It is undesired that users are stuck at an activity while this issue could have been found during user test sessions before deployment.

The user test session did not only involve the thinking-out-loud process. Feedback questions were asked in order to found out even more about what the user has in mind when looking at the system.

The results of the user tests session were processed by dividing the feedback of the users into separate issues. These issues were evaluated one by one with the development team to prioritize them and to ensure that most bugs and issues were fixed and that newly requested features were implemented with the best efficiency possible. The prioritization of issues was defined by looking at how much the user was impacted by an issue. A high priority would be given to bugs or missing functionality. A lower priority was given to minor and visual issues, such as the neatness of the layout in the front-end.

9 Deployment

New technologies were employed for developing BEPSys 2.0, as discussed and justified in earlier sections. In order to implement Single Sign-On integration, the system was required to be hosted inside the infrastructure of the TU Delft. For these reasons, a new production environment for the application had to be configured.

9.1. Production Environment Configuration

As required by the product owner, the new BEPSys 2.0 application will be deployed on an Ubuntu system running an Apache web server and a MySQL database server. The server will be periodically and automatically rebooted to apply security updates as required by the TU Delft's regulations. Different approaches for configuring the server architecture to support the Rails application were studied.

Rails is integrated with an internal middleware stack conforming to Rack, which is a minimal, modular, and adaptable interface between Ruby supporting web servers and Ruby frameworks. Different Rack application servers are available, of which Phusion Passenger, Puma, and Unicorn were studied for implementation and deployment. Puma comes bundled with Rails by default, and was used as a development server.

For Phusion Passenger, an Apache module is available that allows Apache to serve Rack applications by utilising a child process for the Passenger server.

Alternatively, a Rack server serving the Rails application could run as an independent process. By configuring a virtual host, the Apache web server could function as a reverse proxy delegating HTTP requests to the Rack server. To ensure that the Rack server resumes on reboot, the initial idea was to add an ”@reboot” rule to the CRON table. However, as a more reliable approach, the Rack server can be made available and enabled as a service managed by the init system (such as ”systemd”).

9.2. Scalability

Like with many other frameworks, Rails applications can be horizontally scaled, which amounts to the conversion of the server architecture into a three-tier architecture. Such an architecture distributes the load balancer, the Rails application instances, and the database instances over different servers. If multiple distinct database instances are necessary, multi-master or master-slave replication techniques can be employed to synchronize the databases. The second server configuration approach mentioned previously, namely that of running the Rack server as an independent process, best supports horizontal scaling due to the ability to add additional application instances to the architecture and distribute them over different systems. Therefore, this approach was used for configuring the production environment.

9.3. Configuration Automation using Ansible

Ansible[1] is an IT automation language and engine, having different use cases such as application deployment, continuous delivery, and orchestration. The use case for which Ansible was employed for BEPSys 2.0 was configuration automation. From a control node, the configuration of remote machines can be managed over an SSH connection in a fully automated way. Ansible has an agentless architecture, meaning that no installation of special software on remote machines is required in order for them to be controlled; industry standard protocols like OpenSSH are utilised. In an Ansible script - called a ”Playbook” - the needs for the BEPSys 2.0 production environment were recorded in a detailed and declarative manner. The Playbook utilises different user created, self-contained modules called ”Roles” shared in the Ansible community. This script was then run against the production server to allow the application to be deployed.

9.4. Deployment Automation using Capistrano

Deployment to the staging and production environments for the BEPSys 2.0 system was fully automated using the Capistrano gem. This tool provides an extensible and abstract interface for scripting deployment related tasks which are ultimately translated into shell commands. These shell commands are tunnelled over an SSH connection with the remote environment (such as a VPS or a dedicated server).

The Rails framework comes with an additional gem that sets up Capistrano for deploying Rails projects specifically, therefore requiring minimal configuration by the developers. Each deployment yields a new timestamped directory on the server in which the specified version of the system is cloned from the Git repository. Subsequently, assets that are compiled and files are shared across different versions are symbolically linked into the directory (such as user uploads). If the deployment process succeeds, the symlink pointing to the ‘current’ version (the version that exists on the web server) is updated. This allows for an easy rollback to a previous version using a single command if necessary.

[1]`https://www.ansible.com/`

10 Security

The BEPSys 2.0 application handles information for which confidentiality, integrity, and availability must be assured. This section focuses on security issues that arose in the design and development of BEPSys 2.0, and the design choices that were made to tackle these issues are elaborated upon.

10.1. Testing for and preventing OWASP top 10 security flaws

The Open Web Application Security Project (OWASP) classified a set of ten security vulnerabilities in 2013 due to the increasing impact technology partakes in crucial infrastructure such as ”financial, healthcare, defense, energy, and other critical infrastructure” [OWASP, 2013j]. This section discusses nine of the ten flaws present in the OWASP top 10 together with mitigations for these flaws that were employed in the BEPSys 2.0 application.

10.1.1. Injection

Injection attacks occur when unsanitised user input is used in a command or query. An adversary can craft his or her input in such a way that some form of code execution is achieved that is undesired by the developer. This can result in remote code execution on the server or unauthorized access to data [OWASP, 2013a]. The Rails framework has built-in protection mechanisms against injection attacks such as SQL injection, a vulnerability which gives an attacker the ability to inject own SQL queries. By using Rails' ActiveRecord query interface SQL injection flaws are prevented as nowhere in the application raw SQL queries have to be used with untrusted user input.

Stakeholders have additionally requested file and document sharing functionality for groups, which creates a new attack vector. Uploads should only be possible for certain file types and files should be placed in a data directory outside of the application web directory. It should further be ensured that uploaded files can only be written to the intended data directory [OWASP, 2006].

10.1.2. Broken authentication and session management

Flaws in the user authentication and authorization systems occur when this functionality is wrongly developed, implemented, or used. This can result in the compromise of user accounts as a result of password, token, or session theft / manipulation [OWASP, 2013b]. Authentication and authorization issues are limited by relying on existing gems that provide authentication and authorization functionality in an abstracted manner. With these functionalities abstracted away in a small number of gems the developers merely had to focus on correctly configuring and guiding the gems and not on the actual implementation of authentication and authorization features. Gems used for authentication and authorization include Devise, CanCanCan and Rolify.

10.1.3. XSS: Cross-site scripting

Cross-site scripting attacks are possible when a user can provide input that is printed in one way or another to the web browser of a user. This makes it possible for an attacker to inject Javascript code into the user's browser. An attacker could steal session cookies [OWASP, 2013c] or perform other actions related to unwanted information manipulation. The Rails framework automatically prevents cross-site scripting attacks by escaping data directly printed to the browser by default.

It was not possible to consider under all circumstances to print escaped data. Company representatives and TU staff members can style project submissions in BEPSys 2.0 using Markdown[1]. Output from the markdown-to-html parser could not be escaped as this would prevent the parsed markdown from being correctly rendered as html. Without a form of escaping, this would have created a Cross-Site-Scripting opportunity. The solution to this problem was escaping the entered markdown text before it is rendered to html, ensuring that no XSS attacks could occur.

10.1.4. Insecure direct object references

References to files and pages that should not be visible to all users can be exposed. If an adversary can change the URL path of a resource to that of a resource that is not protected, unauthorized data can be leaked [OWASP, 2013d]. All actions in BEPSys 2.0 were build up on the "need-to-know" principle. This means that, by default, a user has access only to a small number of pages and actions and that access to additional pages and actions have to be explicitly defined. This approach ensures that a user can only perform a minimal amount of required actions.

10.1.5. Security misconfiguration

Insecure configurations can cause security issues. The consequences of a misconfiguration(s) depends on the type of misconfiguration in the application [OWASP, 2013e]. A security misconfiguration for Ruby on Rails that has to be dealt with in the production environment is preventing the contents of POST requests to be logged. The contents of POST request can contain sensitive information such as login credentials [Facca, 2017]. Additionally, the web application is only accessible in production via HTTPS to prevent users from accessing BEPSys via the unencrypted and thus less secure HTTP protocol.

10.1.6. Sensitive data exposure

Sensitive information needs to be properly protected from unauthorized access and indexing. Proper protection and encryption ensures that in the case of a breach or unauthorized access it is not possible to read sensitive data [OWASP, 2013f]. In the BEPSys 2.0 system passwords are hashed and salted. The Devise gem used for authentication by default uses bcrypt, a strong algorithm for password hashing [OWASP, 2011]. In a comparison between MD5 and bcrypt, MD5 hashes could be cracked 2.5 million faster than a bcrypt hash [Gosney, 2012].

10.1.7. Missing function level access control

Each function call requires an authorization check to ensure that a user has the correct privileges for accessing a function or resource. This prevents unauthorized access to pages and

[1]https://daringfireball.net/projects/markdown/

unauthorized information edit requests [OWASP, 2013g]. In BEPSys 2.0 access rights are defined for each user per controller action as ‘Ability’. It is not always possible to control authorization using Abilities, in those cases authorization takes place on a controller level.

10.1.8. CSRF: Cross-site request forgery

A web application might not sufficiently check whether an HTTP request is deliberately performed by a logged-in user. If a user is logged in and the vulnerable receiving web application accepts any HTTP request coming from the user without any verification, an adversary can forge requests made to the vulnerable receiving web application by making the user visit a specially prepared website. If this prepared website contains special Javascript code or an embedded link to the vulnerable web application then unauthorized requests can be performed [OWASP, 2013h]. This vulnerability is automatically prevented by the Rails framework if forms are used. Every time a form is created a unique CSRF token is generated that is sent along with the entered data in the form.

10.1.9. Using components with known vulnerabilities: Ruby Gems

Ruby on Rails and gems have experienced vulnerabilities in the past. Installing the latest security patches prevents the exploitation of known vulnerabilities in the server platform [OWASP, 2013i]. The ”bundle-audit” gem is used to check for security patches and the ”brakeman” gem is used to test the application for security vulnerabilities. Both gems were used in the continuous integration pipeline.

10.2. Security Gems

Existing libraries for authentication and authorisation are used instead of building this functionality from scratch. This prevents overhead and the introduction of vulnerabilities in the code base. Devise (a Ruby account gem) and CanCanCan (a Ruby access and permissions manager gem) were used for authentication and authorisation respectively. Both gems belong to the most popular gems for these purposes [TheRubyToolbox, 2017a,b].

10.3. Secure Development

Security issues have been considered into the design of BEPSys 2.0 from the onset. Role and rights management is included in the design, security issues were anticipated and actively tested for, and remediations were documented in advance. Code is kept maintainable and the Ruby on Rails Security Guide is taken into consideration. A clean code base reduces the number of bugs and thus also the number of vulnerabilities. All developers are aware of the most common security vulnerabilities and source code is audited at every merge-request to take security into account.

Throughout the development of BEPSys 2.0, the web application was checked for design limitations and security vulnerabilities. The application was tested for vulnerabilities using static source code audit tool Brakeman. In addition, the dependencies of the system were checked for vulnerabilities using bundle-audit, a tool that checks currently installed versions of gems in the application against an advisory database.

Other security vulnerabilities like flaws in user authorization were tested for using specific dynamic tests. Manual code analysis took place at each pull request and prevented vulnerabilities from being introduced in the stable release. At the end of the project the application is again audited, both with manual source code analysis and with manual penetration testing to ensure that the majority of vulnerabilities are removed before the application is deployed.

11 Ethical Considerations

The following section outlines ethical decision and issues that were discussed and handled during the development of the BEPSys 2.0 application.

11.1. Handling of Confidential Information

The BEPSys 2.0 application will handle multiple sources of confidential information, including student contact and personal information. Federal and TU Delft regulations mandate that the handling of this private information is done with great care, and requires applications utilising this data to be secure and to uphold encryption standards.

Another aspect is that not all users within the system may view all information. Students and staff members are allowed to see most information in the application, but an external account (from a company or an institute) may not view any student information, nor information from another company. Thus, an isolation of data as per the ”need to know” principle is required to adhere to all the regulations and requirements.

The implementation of this revolved around the CanCanCan permissions, in which only the super user (administrator) receiver permission to manage everything in the system. All other roles (student, staff, coordinator, counsellor) have hand-picked and tested permissions in which each data object within the system has indexing, viewing, reading, editing, creating, destroying, and managing permissions that are set and tweaked per role option. This prevents unwarranted data access, and ensures that all users of the application only see information that adheres to the ”need to know” principle.

11.2. Status Confirmations

The BEPSys 2.0 application is utilized for the bachelor project course within Computer Science to display and handle official confirmations related to academic status. This data is submitted to official records of the TU Delft, and can have legal implications for both student and staff. If a student is rejected from participating in a course edition, this is considered to be binding due to the application having been validated and approved by the appropriate counsellor. This decision must only be taken when it is certain that a student does not meet the entry requirements. This situation is identical for approved students, whom must be only be accepted when it is certain they meet the entire requirements.

To facilitate this, the BEPSys 2.0 application contains multiple checks and validations on the importing of student data and utilises modals to confirm the intent of a counsellor when editing an individual user. These modals pop-up and prevent accidental button activations of such confirmation actions, ensuring a counsellor or course coordinator must be certain they wish to perform the desired action upon the selected user(s). These systems aim to prevent possible (legal) conflicts with status and approval in order to ensure a streamlined student confirmation process.

11.3. Group Formation

Although related to the confirmations mentioned in the previous section, consideration has gone into the group management procedure and regulations. There was a period in time in which there was an ethical discussions related to the ability to remove members from groups. It placed the power of legal decisions in the hand of students (namely, whether or not a person would get approved within a group for a project).

After discussions with the client, it was decided to not have any guards against student group management. The client made it clear that a lot of time had gone into assisting students who wished to have their group managed in the previous iteration of BEPSys, costing the course coordinator(s) of the Bachelor Project a lot of time. Although some risks are associated with allowing group members to manage each other, the inclusion of logging and safeguards in the system should prevent accidental sweeping decisions from being taken, thereby averting the issue.

12 Conclusion

The following section outlines the status of the product upon finalisation of the development, and suggestions for future development and extensions of the system. The section also outlines problems with the development and possible improvements in the reflections subsection.

12.1. Product Status

The product that was delivered on the 6th of July 2017 adhered to the requirements as outlined in chapter 2. The majority of the ”must haves” were implemented, the majority of the ”should haves” were implemented, and a sizeable amount of ”could-” and ”would haves” were implemented. Feedback gathered from the client was also implemented during the final week of development, resulting in a significant amount of non-written requirements being implemented. With regards to essential functionality, the new BEPSys 2.0 application supports:

1. Registration for student, staff, externals, and coach users.
2. The signing-on to the application using the TU Delft single sign-on environment.
3. Registration of a new company and projects within a company.
4. Enrolment and counsellor approval for all courses.
5. Registration of multiple courses and course editions under courses.
6. The creation and management of groups with regards to students and coaches.
7. Management of groups, including approving and rejecting groups or projects.
8. The sharing of files between group members and client.
9. The validating of project submissions before showing them to users.
10. Privacy for all users and companies, including an adherence to the ”need to know” principle.
11. Overviews for all roles and objects, split as required.
12. Exporting and Importing of all administrative information.

The delivered product supports all features that are required for the Bachelor Project course, and the roll-out and utilisation of the application appear allotted to start in the first quarter of the 20167-2018 academic year at the TU Delft. The development team remains prepared to assist and make changes in case of requirements that arise during deployment as agreed with the client.

Although not all requirements could be met, most of these issues resulted due to time constraints. With further development the application can continue to be easily developed due to good maintainability.

Interest for the BEPSys 2.0 project management application has arisen from multiple external parties, including the central student council of the TU Delft (represented by the party ”Lijst Beta”), and other degree programmes at the TU Delft. The application could secure additional funding from parties within the TU Delft to continue development and to be expanded with features they require or wish to see.

12.2. Future Development

As a suggestion for future development, the BEPSys 2.0 application mainly requires a focus on additional usability features. These features include extended notifications, linking the single sign-on of the TU Delft with the temporary SAML implementation of the application, and support for a reactive interface through libraries such as React.JS.

If BEPSys 2.0 is rolled out to additional faculties, a prioritisation has to be placed upon expanding the administration panel with flexible features, such as adaptive forms for enrolment and flexible course entry requirement validation. Another such focus could take place with an integration attempt with the TU Delft grading system, OSIRIS, to allow for immediate approval if a student matches these parameters for a course edition.

12.3. Process Reflection

During the process, multiple issues arose which, upon reflection, could have been avoided. The reflection subsection aims to outline a subset of the encountered problems, along with an analysis of possible solutions.

During the first week of the project, it was agreed upon that all team members would arrive at the office at the TU Delft each day at an agreed upon time (barring issues with transportation), in order to allow for a small daily discussions to ensure all team awareness of ongoing processes and to prevent duplicate efforts. Although this was adhered to during the first week, the process of conducting a centralised meeting was not consistent throughout the project. Two members in the team were unavailable during multiple days each week, certain members had regular issues reaching the office on time, and certain events would receive precedent over attending the meetings. This periodically led to a lack of communication during crucial moments. This led to frustration among team members, and some members felt others were slacking or not contributing as much to the collective effort. These issues could have been avoided if the team had constructed and agreed upon a more strict policy for scheduling meetings and office hours.

Regular meetings took place with the client in order to solicit advise and additional requirements if necessary. After the fourth week, this resulted in an ever increasing set of requirements that would eventually lead to scope creep. The team attempted to catch-up to this scope creep, which eventually caused an overwhelming amount of tasks to be completed and many requirements and possible features to be cut from development. A lesson to be learned from this is to better prioritize and select issues, and more frequently and elaborately communicate, amongst team members as well as with stakeholders. This would allow for more realistic planning.

User testing took place late in the development cycle. This resulted in a hassle for selecting possible testers, of which a few were unable to make it due to scheduling issues. The user testing revealed that many minor changes had to occur. These minor changes and requests would eventually build-up to become a subset of close to 100 issues, which all had to be taken care of in the final week of development. Changes ranged from colour and layout changes to additional features that were ”must haves.” These issue overwhelmed the development team and caused the removal of a lot of allotted time for improving software quality and polishing and preparing the system for deployment. If the user testing with the client had taken place in an earlier stage, solving these problems could have been planned for and would have prevented a required rescheduling of all issues in the product backlog.

A significant setback occurred due to administrative delays related to the single sign-on submission. As mentioned in chapter 7 on page 28, support for the single-sign on authentication protocol was built into the system, but was not linked with the TU Delft's single sign-on system. This delay occurred due to problems at the administrative and oversized branch of the TU that is responsible for handling identity management. If the team had spent additional time pursuing the client and the responsible branches to handle the request sent, it could have been sped-up and perhaps implemented within the allotted time frame of the project.

12.4. Process Recommendations

Based on the reflection in the previous section, and based on the experience of the development team, a set of process recommendations that are encouraged for future development of the platform include:

1. Ensure all team members are present during all meetings and at the allocated time.
2. Ensure client and coach meetings occur on regular intervals, and provide moments for all stakeholders to test the application freely to reveal weaknesses.
3. Ensure that new requirements require a definition from the client or relevant stakeholder, and ensure to prioritise them appropriately to prevent scope creep.
4. Ensure that administrative tasks, including sending requests to a large central organisation such as the TU Delft, are sent on time.
5. Ensure that features are properly tested before integration.
6. Prioritise features not through commitment to a schedule, but based on what the product needs at that time in development.

Bibliography

Atern (2008). Moscow prioritisation. Web.

Bashirieh, S. and Rahbari, N. (2013). Bachelor project system (bepsys). Technical report.

Cantador, I. and Castells, P. (2012). Group recommender systems: New perspectives in the social web. *Intelligent Systems Reference Library*, 32:139–157.

Facca, B. (2017). Zen rails security checklist. Web.

Fielding, R. T. and Taylor, R. N. (2000). *Architectural styles and the design of network-based software architectures*. University of California, Irvine Doctoral dissertation.

Gosney, J. (2012). Password cracking hpc. Web.

Kaner, C. and Fiedler, R. L. (2013). *Foundations of Software Testing*. Context-Driven Press.

Mahenthiran, S. and Rouse, P. (2000). The impact of group selection on student performance and satisfaction. *International Journal of Educational Management*, 14(6):255–264.

OWASP (2006). Unrestricted file upload. web.

OWASP (2011). Password storage cheat sheet. web.

OWASP (2013a). Top 10 2013-a1-injection. web.

OWASP (2013b). Top 10 2013-a2-broken authentication and session management. web.

OWASP (2013c). Top 10 2013-a3-cross-site scripting (xss). web.

OWASP (2013d). Top 10 2013-a4-insecure direct object references. web.

OWASP (2013e). Top 10 2013-a5-security misconfiguration. web.

OWASP (2013f). Top 10 2013-a6-sensitive data exposure. web.

OWASP (2013g). Top 10 2013-a7-missing function level access control. web.

OWASP (2013h). Top 10 2013-a8-cross-site request forgery (csrf). web.

OWASP (2013i). Top 10 2013-a9-using components with known vulnerabilities. web.

OWASP (2013j). Top 10 introduction. web.

TheRubyToolbox (2017a). Rails authentication. web.

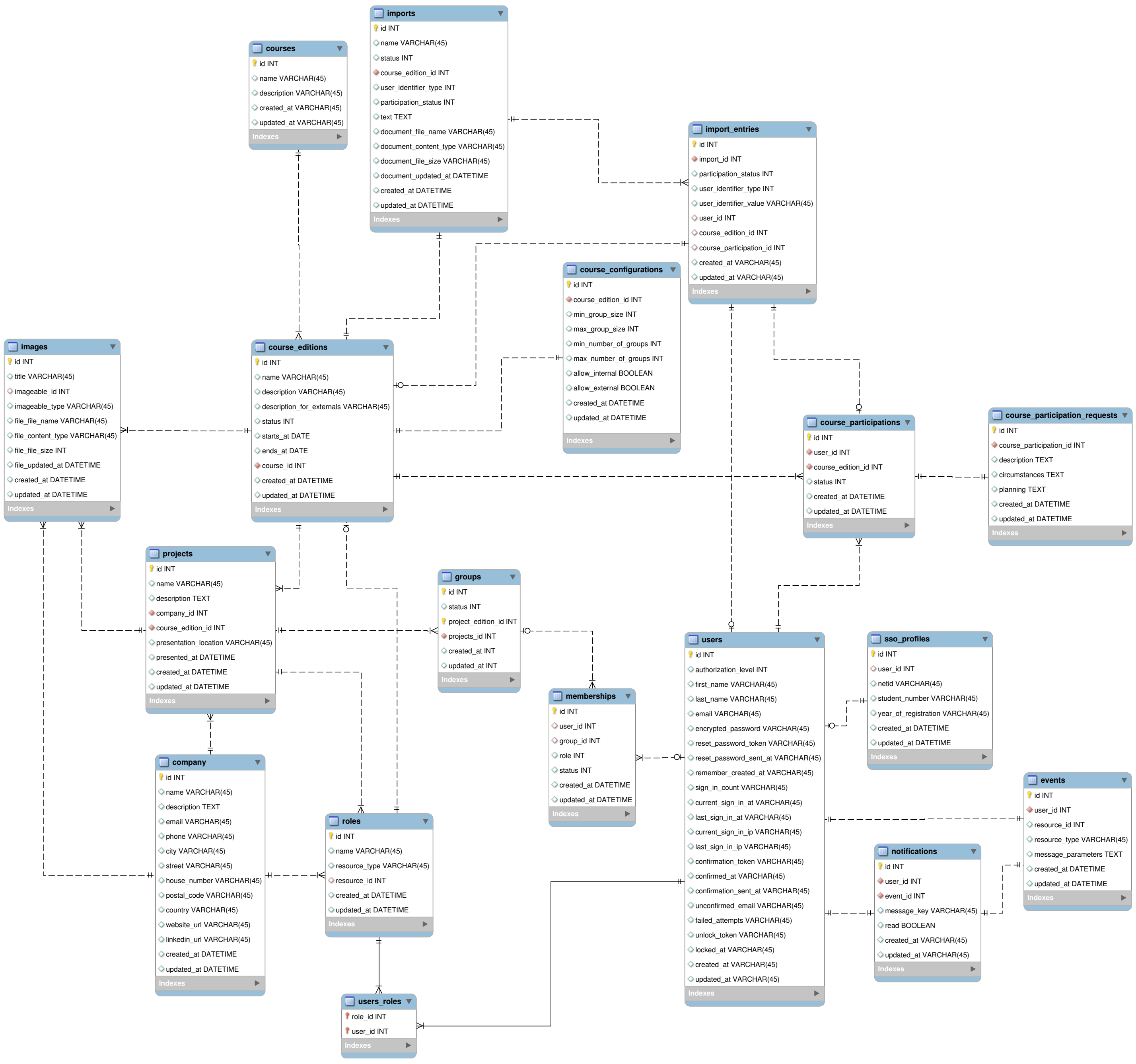
TheRubyToolbox (2017b). Rails authorization. web.

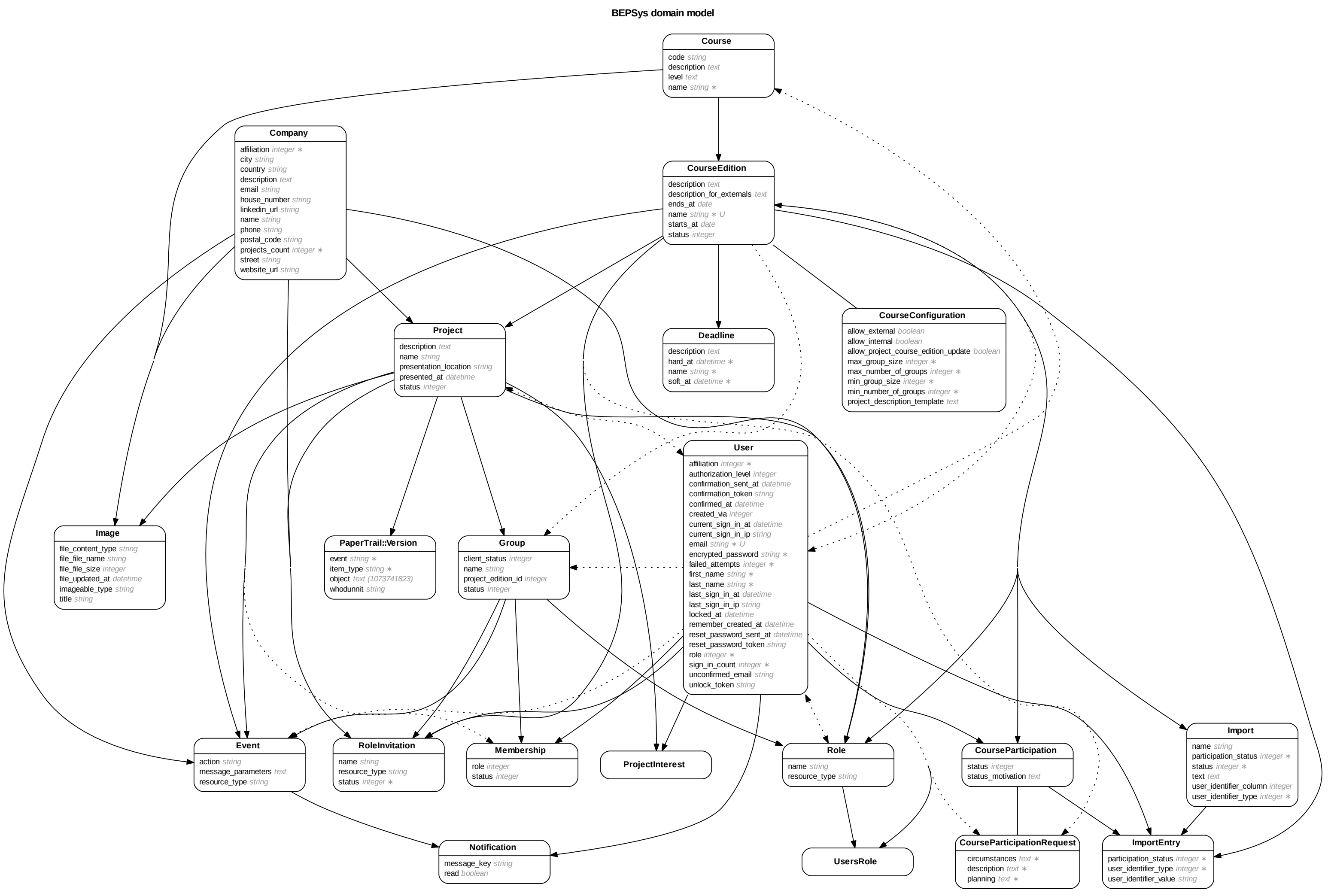
van Solingen, R. (2015). The power of scrum.

Appendices

A
Data Models

The following pages include the final implementation models of the database schema for the system. The first model was manually designed and kept up-to-date using MySQL Workbench. The second model was automatically generated based on the model components in the application using the rails-erd gem.


imports
id INT
name VARCHAR(45)
status INT
course_edition_id INT
user_identifier_type INT
participation_status INT
text TEXT
document_file_name VARCHAR(45)
document_content_type VARCHAR(45)
document_file_size VARCHAR(45)
document_updated_at DATETIME
created_at DATETIME
updated_at DATETIME
Indexes
courses
id INT
name VARCHAR(45)
description VARCHAR(45)
created_at VARCHAR(45)
updated_at VARCHAR(45)
Indexes
import_entries
id INT
import_id INT
participation_status INT
user_identifier_type INT
user_identifier_value VARCHAR(45)
user_id INT
course_edition_id INT
course_participation_id INT
created_at VARCHAR(45)
updated_at VARCHAR(45)
Indexes
course_configurations
id INT
course_edition_id INT
min_group_size INT
max_group_size INT
min_number_of_groups INT
max_number_of_groups INT
allow_internal BOOLEAN
allow_external BOOLEAN
created_at DATETIME
updated_at DATETIME
Indexes
images
id INT
title VARCHAR(45)
imageable_id INT
imageable_type VARCHAR(45)
file_file_name VARCHAR(45)
file_content_type VARCHAR(45)
file_file_size INT
file_updated_at DATETIME
created_at DATETIME
updated_at DATETIME
Indexes
course_editions
id INT
name VARCHAR(45)
description VARCHAR(45)
description_for_externals VARCHAR(45)
status INT
starts_at DATE
ends_at DATE
course_id INT
created_at DATETIME
updated_at DATETIME
Indexes
course_participations
id INT
user_id INT
course_edition_id INT
status INT
created_at DATETIME
updated_at DATETIME
Indexes
course_participation_requests
id INT
course_participation_id INT
description TEXT
circumstances TEXT
planning TEXT
created_at DATETIME
updated_at DATETIME
Indexes
projects
id INT
name VARCHAR(45)
description TEXT
company_id INT
course_edition_id INT
presentation_location VARCHAR(45)
presented_at DATETIME
created_at DATETIME
updated_at DATETIME
Indexes
groups
id INT
status INT
project_edition_id INT
projects_id INT
created_at INT
updated_at INT
Indexes
memberships
id INT
user_id INT
group_id INT
role INT
status INT
created_at DATETIME
updated_at DATETIME
Indexes
users
id INT
authorization_level INT
first_name VARCHAR(45)
last_name VARCHAR(45)
email VARCHAR(45)
encrypted_password VARCHAR(45)
reset_password_token VARCHAR(45)
reset_password_sent_at VARCHAR(45)
remember_created_at VARCHAR(45)
sign_in_count VARCHAR(45)
current_sign_in_at VARCHAR(45)
last_sign_in_at VARCHAR(45)
current_sign_in_ip VARCHAR(45)
last_sign_in_ip VARCHAR(45)
confirmation_token VARCHAR(45)
confirmed_at VARCHAR(45)
confirmation_sent_at VARCHAR(45)
unconfirmed_email VARCHAR(45)
failed_attempts VARCHAR(45)
unlock_token VARCHAR(45)
locked_at VARCHAR(45)
created_at VARCHAR(45)
updated_at VARCHAR(45)
Indexes
sso_profiles
id INT
user_id INT
netid VARCHAR(45)
student_number VARCHAR(45)
year_of_registration VARCHAR(45)
created_at DATETIME
updated_at DATETIME
Indexes
company
id INT
name VARCHAR(45)
description TEXT
email VARCHAR(45)
phone VARCHAR(45)
city VARCHAR(45)
street VARCHAR(45)
house_number VARCHAR(45)
postal_code VARCHAR(45)
country VARCHAR(45)
website_url VARCHAR(45)
linkedin_url VARCHAR(45)
created_at DATETIME
updated_at DATETIME
Indexes
roles
id INT
name VARCHAR(45)
resource_type VARCHAR(45)
resource_id INT
created_at DATETIME
updated_at DATETIME
Indexes
events
id INT
user_id INT
resource_id INT
resource_type VARCHAR(45)
message_parameters TEXT
created_at DATETIME
updated_at DATETIME
Indexes
notifications
id INT
user_id INT
event_id INT
message_key VARCHAR(45)
read BOOLEAN
created_at VARCHAR(45)
updated_at VARCHAR(45)
Indexes
users_roles
role_id INT
user_id INT
Indexes


BEPSys domain model
Course
code string
description text
level text
name string ∗
Company
affiliation integer ∗
city string
country string
description text
email string
house_number string
linkedin_url string
name string
phone string
postal_code string
projects_count integer ∗
street string
website_url string
CourseEdition
description text
description_for_externals text
ends_at date
name string ∗ U
starts_at date
status integer
Project
description text
name string
presentation_location string
presented_at datetime
status integer
Deadline
description text
hard_at datetime ∗
name string ∗
soft_at datetime ∗
CourseConfiguration
allow_external boolean
allow_internal boolean
allow_project_course_edition_update boolean
max_group_size integer ∗
max_number_of_groups integer ∗
min_group_size integer ∗
min_number_of_groups integer ∗
project_description_template text
User
affiliation integer ∗
authorization_level integer
confirmation_sent_at datetime
confirmation_token string
confirmed_at datetime
created_via integer
current_sign_in_at datetime
current_sign_in_ip string
email string ∗ U
encrypted_password string ∗
failed_attempts integer ∗
first_name string ∗
last_name string ∗
last_sign_in_at datetime
last_sign_in_ip string
locked_at datetime
remember_created_at datetime
reset_password_sent_at datetime
reset_password_token string
role integer ∗
sign_in_count integer ∗
unconfirmed_email string
unlock_token string
Image
file_content_type string
file_file_name string
file_file_size integer
file_updated_at datetime
imageable_type string
title string
PaperTrail::Version
event string ∗
item_type string ∗
object text (1073741823)
whodunnit string
Group
client_status integer
name string
project_edition_id integer
status integer
Import
name string
participation_status integer ∗
status integer ∗
text text
user_identifier_column integer
user_identifier_type integer ∗
Event
action string
message_parameters text
resource_type string
RoleInvitation
name string
resource_type string
status integer ∗
Membership
role integer
status integer
ProjectInterest
Role
name string
resource_type string
CourseParticipation
status integer
status_motivation text
Notification
message_key string
read boolean
UsersRole
CourseParticipationRequest
circumstances text ∗
description text ∗
planning text ∗
ImportEntry
participation_status integer ∗
user_identifier_type integer ∗
user_identifier_value string

B
SIG Analysis Results

At the end of the sixth week, the source code was uploaded to the Software Improvement Group (SIG). SIG analysed the code in order to rate the maintainability of the system. In response, SIG reported that BEPSys 2.0 scored 4 out of 5 stars on their model for maintainability, which is above average.

The maximum score was not achieved due to a low score in the area of code duplication. For this category the redundancy in the code was analysed, identifying pieces of code that could be removed after refactoring. For assuring maintainability, removing redundant code is desirable because changes to this code would otherwise need to be implemented multiple times, in different places in the codebase.

The major flaw that was identified, were the methods used for filtering parameters in controllers that were used for managing the same resource (but which were organized in different namespaces). These methods often shared a number of parameters. SIG stated that they understood the presumed reason for this rather redundant approach; in the opinion of the developers this was related to the accepted parameters, which should be customizable per controller to allow different subsets of a model's attributes to be managed via different controllers for authorization purposes. Nonetheless, SIG still advised to try to refactor the implementation by sharing parameters between controllers through an inheritance hierarchy.

The team agreed that sharing the parameters between different controllers is a manner of reducing duplication. An important principle in object-oriented software design is to favour composition over inheritance. Therefore, it was preferred to share the methods - or rather their abstractions allowing customization in each specific controller - through modules instead of super classes.

Besides the issues identified by the SIG, the team was confident that code duplication was largely eliminated. The utilised framework was designed according to the principle "Don't Repeat Yourself," and implementations were continuously refactored in order to create reusable abstractions and modular components.

C
BEPSys 2.0 Colour Table

The following page contains the color schema provided by the ”TU Huisstijl” style guidelines that was adapted for the design of BEPSys 2.0.

Table C.1: BEPSys 2.0 color table

Colour	RGB Value
Black	0, 0, 0
White	255, 255, 255
Blue	0, 166, 214
Light Blue	110, 187, 213
Purple	29, 28, 115
Yellow	225, 196, 0
Red	226, 26, 26
Green	0, 136, 145
Gray	107, 134, 137

D User Testing

The following pages contain the documents utilised during the user testing phase of the development of BEPSys 2.0.

Student

In the test scenario no account has to be created. Use the following login credentials:

User without course enrolment
User: student1@student.tudelft.nl
Pwd: test123

User with course enrollment, without project or group
User: student2@student.tudelft.nl
Pwd: test123

Please think out loud. Describe all the steps that you are going to take and what you would expect after certain button presses.

Scenario 1: Account information update
User: student1@student.tudelft.nl
Pwd: test123
Task: Change your last name

Questions:
- What did you like and dislike about the homepage?
- What did you like and dislike about the account information updating process?

Scenario 2: Register for correct course edition
User: student1@student.tudelft.nl
Pwd: test123
Task: Register for a course edition

Questions
- What did you like and dislike about the course enrolment process?

Scenario 3: Project interest list
User: student2@student.tudelft.nl
Pwd: test123
Task: Join the interest list of a project

Questions
- What did you like and dislike about the project interest process?

Scenario 4: Group joining and coach invitation

User: student2@student.tudelft.nl

Pwd: test123

Task: Join a group and Invite a coach

Questions

- What did you like and dislike about the project joining process?
- What did you like and dislike about the coach invitation process?

Company

In the test scenario no account has to be created. Use the following login credentials:
User: test@external.nl
Pwd: test123

Please think out loud. Describe all the steps that you are going to take and what you would expect after certain button presses.

Scenario 1: Account information update

Task 1: Login with the provided login credentials
Task 2: Change your last name

Questions:

- What did you like and dislike about the homepage?
- What did you like and dislike about the account information updating process?

Scenario 2: Course navigation

Task 1: Login with the provided login credentials
Task 2: Navigate through the courses and find relevant information

Questions:

- What did you like and dislike about the course (edition) overview?

Scenario 3: Project creation

Task 1: Login with the provided login credentials
Task 2: Create a new project (and thus also a new company)

Questions:

- What did you like and dislike about the company creation process?
- What did you like and dislike about the project creation process?
- What did you like and dislike about the project styling option using Markdown?

Scenario 4: Group approval

Task 1: Login with the provided login credentials
Task 2: Go to an unapproved project
Task 3: Approve the project

Questions

- What did you like and dislike about the group approval process?

Coordinator

In the test session no account has to be created. Use the following login credentials:
User: admin@tudelft.nl
Pwd: test123

Please think out loud. Describe all the steps that you are going to take and what you would expect after certain button presses.

Scenario 1: Free navigation through application

Task: Freely navigate through the admin interface of the application

Questions

- What did you like and dislike in general?

Scenario 2: Creating course editions

Task: Create a course edition

Questions

- What did you like and dislike about the course (edition) creation process?

Scenario 3: Updating student names

Task: Update the name of a student

Questions

- What did you like and dislike about the account update functionality?

Scenario 4: Assigning Coordinator privileges

Task: Make user Staff1 (coordinator@tudelft.nl) coordinator for
'Test Course-Edition 1'

Questions

- What did you like and dislike about the process of assigning roles to users?

Scenario 5: Approving project proposals

Task: Approve a project proposal

Questions

- What did you like and dislike about the project approval process?

Scenario 6: Approving groups

Task: Approve a group

Questions

- What did you like and dislike about the group approval process?

Counsellor

In the test scenario no account has to be created. Use the following login credentials:
User: admin@tudelft.nl
Pwd: test123

Please think out loud. Describe all the steps that you are going to take and what you would expect after certain button presses.

Scenario 1: Import list of approved students

Task 1: Login with the provided login credentials
Task 2: Import a list of approved students

Questions

- What did you like and dislike about the import process?

Scenario 2: Approve students

Task 1: Login with the provided login credentials
Task 2: Approve and reject several students

Questions

- What did you like and dislike about the student approval process?

E Infosheet

The following page contains the infosheet that was delivered alongside the technical report.

THE INFORMATION SHEET WAS REMOVED FROM THE PUBLIC VERSION OF THE REPORT DUE TO CONFIDENTIAL INFORMATION REGARDING SECURITY FLAWS AND CONTACT INFORMATION.

F

Research Report

The following pages contain the unaltered research report as provided to the client and coaches at the end of the research phase - the first two weeks - of the project.

RESEARCH REPORT REMOVED FROM PUBLIC REPORT DUE TO SECURITY INFORMATION REGARDING A CURRENTLY ACTIVE SYSTEM AT THE TU DELFT.