Mymoodle

A PBL extension for Moodle

Student report

Aalborg University
Department of Computer Science
Selma Lagerlöfs Vej 300
9220 Aalborg East
http://www.cs.aau.dk

Title:

MyMoodle – a PBL extension for Moodle

Theme:

Application Development

Project Term:

P6, spring 2012

Project Group:

sw609f12

Students:

Anders Eiler
Esben Pilgaard Møller
Bjarke Hesthaven Søndergaard
Martin Sørensen

Supervisor:

Saulius Samulevicius

Copies: 6

Pages: 133

Finished: June 4th, 2012.

Synopsis:

MyMoodle is an attempt to implement Aalborg's Problem Based Learning Model in Moodle. The project seeks to integrate project groups and associated tools in Moodle, while keeping the Moodle core untouched. MyMoodle currently enables each project group to create blackboards, tasks, and meetings, while keeping track of these on a timeline. The project was conducted using modified Scrum of Scrums as development method. It should be seen as a partial solution, as it is the intention it will be continued by other students next year.

Anders Eiler

Esben Pilgaard Møller

Bjarke Hesthaven Søndergaard

Martin Sørensen

Preface

We would like to thank our supervisor Saulius Samulevicius for helping and supervising throughout the project.

The Summary that is required of this report can be found in Appendix E

Quotations are the words of another person along with a source. The source of the citation will either be in the text immediately before or after, or could potentially be incorporated into the quote as shown in the example below:

> “ *This is an example of a quotation.* ”
>
> — X, p. Y

References are references to sections, figures, code snippets, chapters or parts written elsewhere in the report. This could look like the following:

This is explained in Section X.Y.

Code examples are written in a special environment so they are easy to read and recognize. Examples can be seen in Code snippet 1. Whenever there is a sequence of three dots (“...”) in a code snippet, it means that we have omitted some content, which is not important in that specific context.

```
1  <?php
2
3  function print_numbers($from, $to) {
4
5      echo "Hello World! Here are the numbers from " . $from . " to " . $to;
6      for($i = $from; $i <= $to; $i++) {
7          echo "\n" . $i;
8      }
9
10 }
11 print_numbers(1,100);
```

Code snippet 1. Code example of a hello world script written in PHP.

Contents

I Shared Part & Introduction **1**

1 Introduction **3**
1.1 E-Learning . . . 3
1.2 The Aalborg PBL Model . . . 4
1.3 Moodle . . . 5
1.4 Problem Definition . . . 6

2 Related Work **7**
2.1 Learning Management Systems . . . 7
2.2 Relevant Student Reports . . . 8

3 Initial Requirements **11**
3.1 Discussion with Expert Users . . . 11

4 System Definition **15**
4.1 Decomposing MyMoodle . . . 15

5 Methods & Tools **17**
5.1 Development Method . . . 17
5.2 Tools . . . 22

II Analysis & Design **25**

6 Analysis **27**
6.1 Problem Domain . . . 27
6.2 Requirements . . . 30

7 Development Method **33**
7.1 Implementation of Scrum . . . 33

8 Project Group Library **37**
8.1 Implementing Project Group Library . . . 38

9 System components **41**
9.1 Timeline Block . . . 41
9.2 Tasks Block . . . 42
9.3 Types Admin Tool . . . 46

III Implementation **47**

10 General Moodle **49**
10.1 Custom Pages . . . 49
10.2 Moodle Form . . . 50
10.3 Structure of a Block . . . 52
10.4 XMLDB . . . 54
10.5 Admin Tools . . . 55

11 Timeline **59**
11.1 Presentation of a Timeline . . . 59
11.2 Timeline Functionality . . . 61
11.3 The Filters . . . 64

12 Tasks **67**
12.1 Creating Tasks . . . 67
12.2 Editing Tasks . . . 68
12.3 Types Admin Tool . . . 71

13 My Moodle **75**
13.1 PGLib . . . 75
13.2 System Integration . . . 75

IV Testing **79**

14 Testing Approach **81**
14.1 Black-box Testing . . . 81
14.2 White-box Testing . . . 83
14.3 Testing Approach . . . 84

15 Test Plan **85**
15.1 Test Phases . . . 85

16 Test Report **91**
16.1 Code Coverage Results . . . 91
16.2 Dynamic Black-box Test Results . . . 91
16.3 Integration Test . . . 93
16.4 User Feedback . . . 94

V Epilogue **97**

17 Discussion **99**

18 Conclusion **101**

19 Future Work **103**
19.1 From Proof of Concept to Production . . . 103
19.2 Implementation of Suggested Changes . . . 103
19.3 Expanding Functionality . . . 103

Bibliography **105**

Appendix **111**

A Demo Meeting **111**
A.1 Lea Gustavson 111
A.2 Mathilde Gammelgaard 112

B Dynamic Black-box Test Cases **115**

C Dynamic Black-box Testing Results **125**

D Integration Test Results **131**

E Summary **133**

Part I

Shared Part & Introduction

Introduction 1

Today, computers are heavily integrated with the educational system. Different educational methods are supported by different information communication systems, called e-learning systems. This observation has lead us to investigate how the method of our university, the Aalborg Problem Based Learning (PBL) model, can be implemented into the e-learning system being used at our university, namely Moodle.

Compared to the previous semesters, the bachelor project is differently structured, since the goal of this project is to collaborate on a multi-group project. At the beginning of the semester two larger development projects were proposed, each of which were then divided into sub-projects. This meant that every "sub-group" working on a larger project was supposed to work together towards creating a single system. The goal of the multi-project described in this report was to develop an extension for Moodle, which 14 students chose to assign themselves to. The large multi-project was then divided into smaller sub-projects consisting of three to four students. This part of the report is shared among every sub-group and contains the same content in every report. In this part "we" refers to all 14 members of the multi-project.

1.1 E-Learning

The term e-learning covers all forms of electronically supported learning. E-learning is often associated with distanced learning and out-of-classroom teaching, but can also be used to support traditional teaching in classrooms. In short e-learning is defined as learning that is facilitated and supported via Information and Communications Technology (ICT). The cooperation between the teachers and students, and among the students themselves, can similarly be partially or completely conducted via ICT [2][3].

At Aalborg University e-learning is employed in a variety of forms. There are courses taught exclusively online in the Master of Problem Based Learning (MPBL) education [15], and regular courses that make use of online quizzes and more as a method of teaching.

1.1.1 Learning Management Systems

E-learning is often conducted through the use of a Learning Management System (LMS). An LMS is loosely defined as a software system that administrates, tracks, and reports on teaching. It is considered robust if it contains the following functionality [42]:

- Has centralized and automated administration.
- Uses self-service and self-guided service.
- Is able to assemble and deliver learning content rapidly.
- Consolidates teaching activities on a scalable web-based platform.
- Supports portability and standards.
- Has the ability to personalize content and reuse knowledge.

LMSs have many forms, each suited to a specific target group. The general characteristics of an LMS are [47]:

- Student registration and administration.
- Management of teaching events such as scheduling and tracking.
- Management of curricula and obtained qualifications.
- Management of skills and competencies (mostly for corporate use).
- Reporting of grades and approved assignments.
- Management of teaching records.
- The ability to produce and share material relevant to courses.

The purpose of an LMS is to handle and administrate all study-related activities at a learning institution.

Moodle (Modular Object-Oriented Dynamic Learning Environment) [13] is currently the primary e-learning platform at Aalborg University (AAU). Its main purpose is to allow lecturers to share course-relevant material with students and to serve as a calendar service containing dates of lectures, meetings etc. Unfortunately, Moodle does not support PBL, which is the learning method used at AAU.

1.2 The Aalborg PBL Model

The Aalborg PBL Model is a term coined to describe AAU's problem based learning model. It originates from the philosophy of the university's staff. They were interested in giving the students an active role in obtaining knowledge, as opposed to the lecture-based learning method used at many universities. Furthermore, they wanted to give the faculties a more active role in the students' learning experience than what the lecture setting provided. From this, the Aalborg PBL Model was developed. This section is based on [35] with supplementary literature from [48, pp. 9-16].

The Aalborg PBL Model consists of six core principles, which outline the students' learning process. These are:

- **Problem orientation** - A problem relevant to the students' field of study that serves as the basis for their learning process.
- **Project organization** - The project is the medium through, which the students address the problem and achieve the knowledge outlined by the curriculum.
- **Integration of theory and practice** - The staff at the university is responsible for teaching the students to connect their project work to broader theoretical knowledge via the curriculum.

- **Participant direction** - Students define their problem and make decisions relevant to the completion of the project themselves.
- **Team-based approach** - Most of the students' project related work is conducted in groups of three or more students.
- **Collaboration and feedback** - Peer and supervisor critique is used continuously throughout a project to improve the students' work. The aim is for the students to gain the skills of collaboration, feedback, and reflection by following the Aalborg PBL Model.

From this point on we will use the term "project group" to describe a team of students working on a project in cooperation.

1.3 Moodle

Moodle is an e-learning platform for creating dynamic web sites for courses. It is written in PHP and supports SQL databases for persistent storage. Moodle is originally developed by Martin Dougiamas in 2002 and is released under an open source license (GPLv3+) [8][55]. It is currently maintained by a community of developers. Due to its modular design, the functionality of Moodle can be extended with plugins developed by the Moodle community. The version of Moodle that we have decided to use is Moodle version 2.2.

Moodle is built around the concept of courses, and most activities in Moodle are centered around them. Courses can be divided into categories. Topics, resources, activities, and blocks can be added to courses [41]. Topics are an integrated part of courses, and resources can be links to external sources or uploaded files. Activities and blocks are plugins, which can be added to a course to provide additional functionality. An activity is added by placing a link on a course page to where the functionality of the activity lies. A block can be shown visually on the course page. An example of a Moodle course page can be seen in Figure 1.1

As Moodle is built around courses it does not provide much functionality to support the Aalborg PBL Model discussed in Section 1.2.

The individual groups will elaborate further on Moodle if needed. Based on the content described so far we define our problem in the following section.

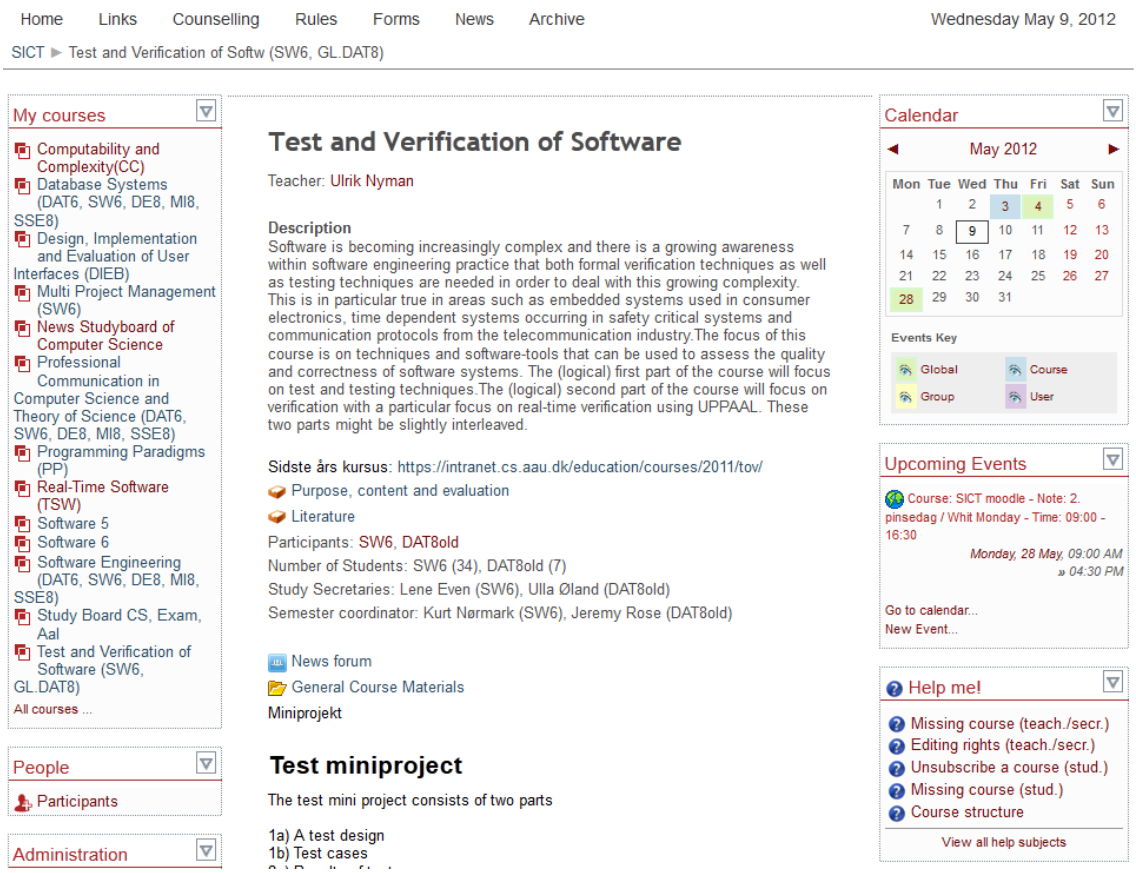

Figure 1.1. A Moodle course page for the Test and Verification course

1.4 Problem Definition

The problem that this project is concerned with is the following:

Moodle does not fully support the work method used at AAU. Moodle is built up strictly around courses, and does not support the concept of project groups. To accommodate for this, students must use other tools for project group work. This project deals with how support for the Aalborg PBL Model can be implemented in Moodle. This involves researching other systems than Moodle for information on how they accommodate project groups, and conducting interviews with students and administrative personnel of different faculties to gather requirements for such an extension to Moodle.

Related Work 2

In this chapter existing LMSs will be examined to see how they support the Aalborg PBL model to gain inspiration for how it could be implemented in Moodle. Furthermore, we will look at student reports from the previous year in order to draw on their experiences with multi-projects, since they worked with a similar topic.

2.1 Learning Management Systems

In this section we will examine a number of existing LMSs. The systems that will be examined are SharePointLMS, Litmos, and Mahara. We will examine how they handle students, courses, and the concept of groups. The objective is to gain an understanding of how an extension to Moodle could be structured, and which tools might be relevant to include in the system.

2.1.1 SharePointLMS

SharePointLMS [16] is an LMS based on the Microsoft SharePoint platform. It offers the basic functionality of an LMS such as course management, student assessment tools such as quizzes and course certificates, conference tools, and document sharing.

In terms of PBL, SharePointLMS offers some relevant features, such as creation of groups. Within a group it is possible to enable features such as a chat, an internal mail, a calendar and an online conference tool for meetings that could be useful when working in a PBL context. The only drawback is that groups cannot be created as independent entities in the system, but have to be created in the context of a course. This does not fit well into the Aalborg PBL model, where a clear distinction exists between courses and projects.

2.1.2 Litmos

Litmos [11] is a lightweight LMS with focus on being easy to set up and use. Its main features are creation of courses including multimedia content such as audio, assessment of students, and surveys to gain feedback on courses.

In contrast to SharePointLMS, Litmos supports creation of groups as independent entities and even creation of groups within a group. A group can be assigned to a course, which could be utilized to model the way courses are structured at AAU. This could be achieved

by creating one group containing all students on a given semester and then assigning this group to the relevant courses. Within this group a number of sub-groups would be created, representing project groups. Litmos has a built-in mailing system and is also integrated with Skype, which could be used to communicate within the groups.

2.1.3 Mahara

Mahara [12] is technically not an LMS, but a Personal Learning Environment (PLE), meaning that it is more learner-centered, as opposed to LMSs, which are typically more institution-centered. However, Mahara still has some features that are relevant to the Aalborg PBL model.

Mahara aims to be an online portal where students can share their work and be members of communities within their area of interest. However, it does not come with a built-in calendar, which makes planning of projects an issue when using only Mahara. Mahara compensates for this by providing a sign-on bridge to Moodle, allowing users to access their Moodle accounts directly from Mahara without having to sign in again, and vice versa.

The aspect of Mahara relevant to the Aalborg PBL model is its social networking feature, which allows users to maintain a list of friends and create groups. Within a group it is possible to create a private forum as well as share files.

Overall Mahara provides a good platform for communicating within project groups, but it lacks support for planning and coordinating the projects.

2.1.4 Comparison

None of the examined LMSs provide complete support for the Aalborg PBL model. However, they do provide a variety of features that, if combined, would provide the functionality one would expect a PBL-oriented LMS to have.

The group structure found in Litmos could be combined with the features found in SharePoint to create a portal where students could organize their group work. The functionalities typically provided in LMSs appear to be mostly forums, internal communication and planning tools. None of the examined systems consider the aspect of communicating with a supervisor.

2.2 Relevant Student Reports

During the spring semester of 2011 a group of students at AAU were given the task of solving a problem similar to ours. They had to develop an LMS as a multi-project consisting of four groups, with the goal of fulfilling the needs of the students and faculty of AAU. However, the solution they developed was created from scratch, unlike our solution, which is an extension of an existing system. Furthermore, the focus of the earlier project was to develop a solution which fulfilled the needs of teachers and students in relation to courses, whereas our project focuses on developing a solution to better facilitate PBL.

Despite the differences between our project and theirs, we concluded that it would be beneficial for us to examine their development methods and try to learn from their experiences, since the multi-project approach, target group analysis and product testing aspects still remain highly relevant for us.

We examine the following reports: *E-LMS - Authentication, Database, and Educator* [38], *E-Learning Management System: Implementing Customizable Schedules* [36] and *E-LMS - Administration, Calendar, Model, Education, Courses* [53].

The following is a list of relevant and interesting observations the groups made in their reports:

- Each group was allowed to pick their own development method, which lead to problems later in the development process as some groups used an agile development method, while others used a traditional method. Integration testing became a problem as the groups were not synchronized and were always in different stages of their development process. Agreeing on a shared development method would have been preferable.
- Each group sent a representative to meet with representatives from other groups at least once every fortnight, where each representative presented their progress since the previous meeting. As these meetings were primarily for status updates, very little coordination took place between the groups.
- Every month there would be a large meeting where all the students and supervisors were present and the current status of the system was presented.

Even though we cannot use the product that was developed by the students last semester, we can still use their experiences to improve this project and not make the same mistakes as them.

Initial Requirements 3

To determine how to enable Moodle to support the Aalborg PBL model we conducted two informal discussions with E-Læringssamarbejdet ved Aalborg Universitet (ELSA) [5] and the Master in Problem Based Learning (MPBL) department [15] at Aalborg University. This section accounts for the results of the discussion. Based on the results we decide how to solve the problem explained in conceptual terms.

3.1 Discussion with Expert Users

We conducted meetings with ELSA and MPBL, in which we discussed which improvements we could make to Moodle in regards to the Aalborg PBL model.

3.1.1 ELSA

- Type: Informal meeting
- Participants: Lillian Buus [40], Marie Glasemann [43], Mads Peter Bach [34], and us
- Date: February 6, 2012

ELSA is responsible for the technical, organizational and pedagogical support of Moodle at Aalborg University. [5] However, the actual development on Moodle is outsourced to the different IT departments of the university. ELSA receives all feature requests and is responsible for providing support to the departments of the university. Because of this, they have keen knowledge about the problems and shortcomings of Moodle. We are only interested in improving Moodle in relation to the Aalborg PBL model, thus we have chosen five of the subjects ELSA proposed that we deem relevant to this project:

Automation of input data

A course is automatically generated from a template. This empty course now needs to be filled with content, which requires manual work. This is primarily done by the administrative personnel and the lecturers. ELSA would like this to be generated automatically based on central data. ELSA would prefer that new entities introduced by us were also generated automatically.

Maintenance of data

Courses are, as mentioned above, maintained by the administrative personnel and the lecturers. ELSA mentions that we should consider who should maintain the data introduced by features implemented by us.

Overview of data

When a person is enrolled to several courses, the task of finding and entering the course page becomes difficult. ELSA would like a feature in Moodle that renders a more simple overview, which eases the task of finding the wanted item. We need to ensure that similar problems do not occur in any features we develop.

Sharing of data

The ability to share files and relevant material is a central concern when engaged in project group work. ELSA would like features to be more general such that they could be used in different contexts. For example, this could be the ability to share quiz questions between courses.

Archiving

ELSA currently archives all courses and their related material. The archiving is done by closing all courses for enrollment and then copying all data from the Moodle installation. This approach does not work well, since students complain that they cannot enroll or unenroll from courses during the backup process.

3.1.2 MPBL

- Type: Informal meeting
- Participants: Jette Egelund Holgaard [46], Morten Mathiasen Andersen [33], and us
- Date: 08-02-12

The MPBL department defines itself as: "Master in Problem Based Learning is a fully online and highly interactive e-Learning programme for faculty staff at institutions who want to change to Problem Based and Project Based Learning (PBL)". [15] At the department of MPBL they educate staff from other educational institutions in how to conduct PBL. Moodle and other communication technologies are used in the education process. The duration of the online education is two and a half years. To gain knowledge about how Moodle is used in professional and PBL contexts we conducted a meeting and discussed how Moodle can be improved to better support PBL. We chose to consider four of the subjects MPBL suggested:

Joining of tools

MPBL explains that they are using Moodle as a part of their education but it does not support any functionality that aids PBL. To facilitate the online education other technologies such as Skype [19] and Adobe Connect [1] are used. MPBL would like the necessary tools to be available in Moodle to avoid having to use several external tools.

Nearness

When working as a group it is advantageous to be in the same room. When that is impossible, which is often the case for MPBL, tools must create the feeling of nearness. A virtual meeting place could create the wanted feeling of nearness and thus the foundation of working as a group is set.

Collaboration

Management and sharing of documents between the members of a group is a common issue. A need exists in Moodle to better handle sharing of documents. The ability to comment on documents is a common way to give feedback on the work of others. Currently, document sharing is conducted via emails, Dropbox [4], and Google Docs [7].

Planning

When group members are geographically separated it is essential to coordinate and plan. MPBL would like Moodle to contain a tool for planning the collaboration process.

System Definition 4

Based on our Problem Definition, Initial Requirements, and Related Work, we define the system that we develop in this project, MyMoodle, as follows:

> *MyMoodle is an extension for Moodle that allows Moodle to support the Aalborg PBL model.*

Since MyMoodle is an extension it adds functionality without changing existing functionality.

4.1 Decomposing MyMoodle

Since the project is large it is decomposed into sub-projects. This is due to the educational context in which the project is being conducted. The educational context requires us to work as one large group divided into four sub-groups.

Each sub-group will have its own dedicated project, of which the goal is to make a part of MyMoodle. The parts from each sub-group will be integrated with each other, and the development process and results of each part will be documented in the corresponding sub-group's report [23].

The final system, as defined above, should enable PBL in Moodle. We will divide the system into sub-systems to ensure that we cover the core principles of the Aalborg PBL model as defined in Section 1.2. The core principles we find relevant to this project are:

- Project organization
- Participant direction
- Team-based approach
- Collaboration and feedback

These principles are important for MyMoodle, and must be considered in each sub-system. However, each sub-system prioritizes the core principles differently. Notice that two of the core principles are not considered. The core principles, problem orientation and integration of theory and practice, are important concepts, but we do not believe that we can cover

these in an LMS. These must be covered by the students themselves with guidance from their supervisor.

There are several ways to decompose MyMoodle to integrate the core principles of the Aalborg PBL model in Moodle. To support the principles of project organization and team-based approach we decide to implement the concept of project groups in Moodle. One sub-system, called Project Group Management, implements this concept. To implement collaboration and feedback we need a tool for planning, collaboration between the members, and communication with the supervisor(s). The principle of participant direction is implemented in general by all parts.

Each sub-system is described in the following sections.

4.1.1 Project Group Management

A project group is a team of students working on a project. It should be possible for students and their supervisors to work together on a project. It should be possible to create and manage project groups in a simple and intuitive manner. The nearness request made by the MPBL department in Section 3.1.2 should be satisfied in this sub-system by giving participants of a project a virtual meeting place.

4.1.2 Project Planning

The students can plan the way they want to organize their projects themselves. This sub-system is responsible for allowing students to make a schedule for a project and assign tasks to each other. A student that is participating in several projects concurrently should be able to get a collective overview of his schedule. This also corresponds to the request made by ELSA to allow sharing of data described in Section 3.1.1, along with the planning request made by MPBL department in Section 3.1.2.

4.1.3 Intra-group Collaboration

During a project, the participants must be able to collaborate regardless of geographic location. The collaboration must be efficient such that deadlines are met and progress is made. In Section 3.1.2 MPBL describes that they would like to have better collaboration between students in Moodle.

4.1.4 Supervisor Communication

The participants must be able to communicate with, and receive feedback from, their supervisor. This sub-system should accommodate the MBPL departments requests for collaboration and planning described in 3.1.1

Methods & Tools 5

In this chapter we describe the development methods and tools that are considered for this project. A choice is made for tools and development methods along with reasons for the choices.

5.1 Development Method

For the collaboration between the four groups to work, it is important that we have a common understanding of the development method we are using. The groups that were developing an LMS last year each used different development methods, which caused problems, see Section 2.2. In this section different development methods are presented and one is chosen for this project.

5.1.1 Considered Methods

We consider a selection of traditional and agile development methods. In general, traditional methods follow a schedule planned upfront, where every task is handled as a single unit and the result is not changed afterwards [52, sec. 2.7]. This method is inspired by the methods used in the construction industry.

Projects using agile methods are developed in iterations, where some part of the product is developed in every iteration [49, p. 25]. This should help with adapting to changes that the end-users or customers might pose during the development.

Some methods can have characteristics from both agile and traditional. The development methods considered in this project are: Extreme Programming, Scrum, and Waterfall. These are presented in the following sections.

Extreme Programming

Extreme Programming (XP) is an agile development method that consists of 12 recommended core practices [49, p. 137]. These include, but are not limited to: Frequent refactoring, pair programming, and the whole team working together in a single room. The core of XP is to find every practice that is considered good and taking it to the extreme, e.g. since code reviews are good, do them all the time through pair programming. Kent

Beck, the creator of XP, states that in general all the practices of XP should be applied because they compensate and support each other [49, p. 156-157].

There are roles assigned to different people involved in the project [49, p. 145]. These roles are: customer, programmer, tester, coach, tracker, and consultant. All these roles are important, but Larman, the author of [49], stresses that an on-site customer or at least an on-site customer proxy is needed [49, p. 152-156].

When using XP, the development is done in iterations lasting one to three weeks each. An iteration should not be planned until right before the start of it. At the start of the iteration the on-site customer should help prioritize which features should be implemented in the given iteration and the programmers estimate the time to implement them. This process is called the "Iteration planning game".

Scrum

This section describes the agile development method Scrum, and is based on [49, chap. 7,pp. 109-136]. Scrum is, as XP, an agile development method that utilizes a number of iterations of development cycles. These development cycles are known as sprints. A sprint usually has a length of 30 calendar days. A sprint backlog is created prior to the sprint. This backlog consists of the features, ordered by priority, that should be implemented during the sprint. The sprint backlog cannot be changed during the sprint. If some of the features in the sprint backlog are not implemented during the sprint they are moved to the product backlog, which is a list of the features that should be implemented in the future.

In Scrum there are different roles. There should be a product owner, whose task is to meet the costumer's and end-users' interests. These interests should be formalized and prioritized in the product backlog. There is also a Scrum master. A Scrum master serves as a link between the development team and any external individuals. It is the Scrum master's task to ensure that the development team is not disturbed. The development team is usually small, and their task is to design, analyze, and implement the features from the backlogs. Every day, the Scrum team holds a short meeting where they say what they have done since the preceding meeting, what they plan on doing the present day, and any problems they are having.

A variant of Scrum for more teams is called Scrum of Scrums or Large Scrum [44, pp. 23-30][49, p. 111]. In such a project there is one complete team consisting of several smaller sub-teams that will hold ongoing Scrum of Scrums meetings during each sprint to synchronize their work. The Scrum of Scrums meeting is similar to the daily Scrum meeting with a representative from each sub-team, except that they talk about what the sub-teams have done and should do instead of what individuals have done and should do.

Waterfall

A project following the Waterfall method is divided into a number of phases. The number of phases in a Waterfall development process varies for each implementation. One version of the waterfall model defined at [50] divides the waterfall model into six phases:

- **Requirement gathering and analysis** - The initial phase where all possible requirements are gathered from the end-users of the system. Their validity and whether their implementation is feasible is analyzed. The result of the phase is a requirement specification document, which is used in the next phase.
- **System design** - Based on the requirements a design for the system is constructed. The result is a system design specification document.
- **Implementation and unit testing** - Based on the design document the system is divided into modules and units. The coding is started in this phase. Upon completion every unit is tested.
- **Integration and system testing** - All the developed units are combined into the final system. The integration of the units is tested. The complete system is tested to ensure it satisfies the requirement specification document. After a satisfiable number of tests have been conducted the system is delivered to the customer.
- **Deployment** - Depending on the system some initial setup and configuration of the software may be necessary before the end-users can use the system. The result of this phase is a deployed system that is set up to the end-users' needs.
- **Operations and maintenance** - A theoretically never-ending phase. After the software is delivered issues may arise from practical use of the software and bugs may be discovered. When issues arise they are dealt with as long as the software is in use.

Waterfall models are strictly traditional, since in their pure form they do not allow to move back to a phase once the next is started. This suggests a big and heavy upfront design plan that is to be followed until the project is finished.

Of the methods presented, Waterfall is the one that requires the most amount of documentation. To complete a phase, some document must be created to be used in the following phases. These documents can vary from an architectural design document to source code.

5.1.2 Choosing a Development Method

The following list shows the characteristics of this project, which will be used to determine the development method for the project.

1. **Four groups (14 persons in total)**
   This semester the students are divided into two major groups, one of which is working with Moodle. At AAU, a group limit of four members is imposed during the sixth semester, therefore the Moodle project has been divided into four sub-groups.
2. **Diverse target group**
   The relevant target groups are: students, supervisors, and administrative personnel.
3. **No on-site costumer**
   Because this is a university project, there are no on-site customers. Instead, there are several contact persons such as supervisors, students, and administrative personnel, who are available to test the product and provide feedback.

4. **Hard deadline**
   Because this is a semester project the project must be handed in when the semester ends.
5. **Pass on project**
   This project will be passed on to the sixth semester students next year.
6. **Known framework and platform**
   The Moodle platform is open source and documentation is available on most relevant topics.
7. **Education environment**
   Since this is a university project, we are working in an educational environment.
8. **Not full-time development**
   Because this is a semester project there are lectures beside the project, therefore the development time is limited.
9. **No manager/product owner**
   Because this is a semester project, there is no project manager or product owner as there would be in a corporate environment.
10. **No shared working room**
    We do not have a room available where all four groups can work simultaneously.
11. **Low criticality**
    If the system fails it will only affect the comfort of the users.

Barry Boehm and Richard Turn, authors of [37], have identified five factors that can be used to determine whether to use a traditional or agile development method. These factors are: personnel, dynamism, size, culture, and criticality.

The personnel factor covers the composition of personnel based on the *extended Cockburn method skill rating scale*, where people are divided into five categories based on their methodological skills. The methodological skills are divided into five levels: -1, 1b, 1a, 2, and 3 [37, p. 34]. The levels of the extended Cockburn scale are defined as follows: A person with level -1 is unable or unwilling to follow a shared development method. With training a person with level 1b can perform procedural development method steps such as writing a function while conforming to coding standards. A trained level 1a person is able to perform discretionary development method steps such as using design patterns to solve a problem. A person with level 2 can alter a development method to handle a new but similar situation. A person with the last level, 3, is able to alter a development method or create a new one to handle a new unfamiliar situation.

Dynamism is the anticipated percentage of changes in requirements that will occur during the project. The size is simply the number of people in the development team. The culture factor is a scale of how much the team prefers chaos over order. Criticality is a scale of how much a system failure will influence the real world. This is based on the Cockburn Scale used to differentiate between Crystal methods [49, pp. 36-37].

A polar chart showing the factors for our project can be seen in Figure 5.1. As seen, most of the points are closer to the center than the periphery, this indicates that an agile method is preferred over a traditional one.

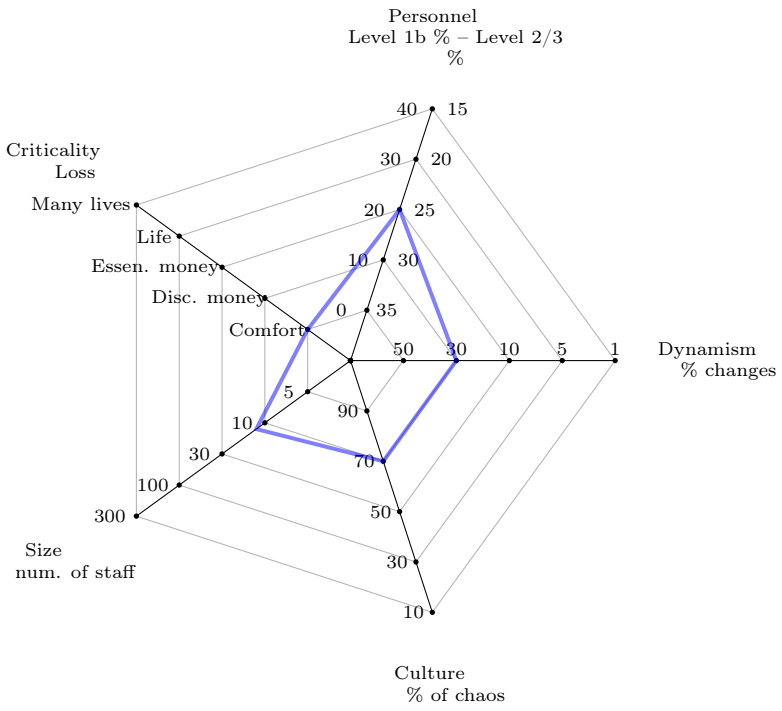

Figure 5.1. Radar chart showing the deciding factors for choosing the development method of our project

The personnel score is positioned in the middle because we consider ourselves above level 1b, but none or very few of us qualify as level 2 or 3.

We will use a development approach similar to Scrum of Scrums. The reason for this is three-fold. First of all, we have a diverse target group (item 2), which may make a big upfront analysis and design difficult to perform. This lead us to choose an agile method due to the risk of "I know it when I see it" (IKIWISI). IKIWISI is when the customer does not know what she wants before she sees it. Our choice is also supported by the polar chart showing the development factors in Figure 5.1.

Secondly, we are 14 members divided into four groups (as item 1 states), which is not handled very well in other agile methods, such as XP, that dictates that all the developers should be in the same room. This is not the case for us as item 10 states.

The third reason is that we have a hard deadline (item 4), which means that we have to hand in our project at a specific date. Scrum of Scrums suggests that iterations (sprints) are time-boxed, which is ideal for us since we can cut less important features instead of missing the deadline. This is also supported by item 5, because the end product is a working release although some features may have been cut. The features cut may then be suggested to the group of students, who are to take over this project next year.

5.1.3 Refining Scrum

As items 9 and 3 in the previous section state, we have neither a project manager nor an available on-site customer. We will handle the missing on-site customer by having shorter iterations and contacting the customers whenever an iteration is over. Scrum of

Scrums [21] dictates that there should be a Scrum master in each sub-group. Since none of us have used Scrum before, none of us are qualified to be a Scrum master. We are in an educational situation (see item 7) so we will strive to allow every member to be Scrum master for some time period. This may not be ideal, but we consider it to be more important that every member of the sub-groups tries to have the responsibility of a Scrum master than having only one member trying it and learning it well.

5.2 Tools

A series of tools are used in the creation of this project. These tools are described briefly in this section. The tasks that must be handled by tools are: version control, bug tracking, code documentation and testing.

5.2.1 Version Control

All of us have been using subversion (SVN) for version control in previous projects. SVN is a centralized solution [20] with a single repository the group members can update from and commit changes to. However, this project differs from previous projects with respect to the organization of groups; we are not one group of x individuals, but rather one group divided into sub-groups. This has lead us to choose a distributed solution rather than a centralized one.

The solutions considered are the distributed systems Git [6] and Mercurial (Hg) [9]. These systems are quite similar and the main difference is that Hg is simpler than Git and Git is more flexible than Hg [22]. A few of us have been using Hg and none of us have used Git, which leads us to choose Hg, such that we have a little knowledge of the chosen system.

5.2.2 Bug Tracking

We need to have some way of communicating and tracking the defects or bugs that we will discover during our project. The most important requirement for the tool is that it should be able to track the bugs and make them easily available to the team that will continue this project next year. The tools considered for bug tracking are Bugzilla [39] and Eventum [25]. These tools are very similar in their features. When a bug is discovered a bug report must be added in the tool. In the tool, the members of a team can see the bugs and their status, and mark them as fixed when appropriate.

Since we have used Bugzilla as part of the course Test and Verification, choosing Bugzilla will save us the overhead of having to learn a new tool. Furthermore, the development of Eventum seems to have been discontinued since the beginning of 2009 [24], which means that if defects exist they are unlikely to be fixed. In conclusion, we choose to use Bugzilla to track our bugs.

5.2.3 Code Documentation

We do not wish to use a tool that requires us to write documentation externally from the code. We plan to use a tool like PHPDocumentor [45] or PHPXref [54] to handle our documentation, since both of these read comments in the source code and use it for

documentation. This gives us the ability to write code and documentation in the same file. The syntax both of these tools use is the same, which means that as long as the syntax is used either tool can be used to generate documentation.

The difference between the two tools is that the focus of PHPDocumentor is to give a more diverse set of final documents (different HTML and PDF templates), where the focus of PHPXref is to show references between classes, functions, etc. We use both tools such that we and the group continuing this project next year can choose to read the documentation they prefer.

5.2.4 Testing

We want the tests to remain available after our work on the project ends, since the groups that are to take over this project next year should be able to reuse our test cases. We will use the built-in testing framework of Moodle [14], which is based on SimpleTest [17]. We do not consider other tools, since it reduces the number of tools we have to use by using one which is already a part of Moodle. This should ensure that the test cases can be used next year and be part of the final product, should it be released to the public.

Part II

Analysis & Design

Analysis 6

This project consists of four parts that are split amongst the four project groups. The four parts are:

- Project organization
- Participant direction
- Team-based approach
- Collaboration and feedback

In this group, we will be working with project organization.

This subpart of the system was concerned with how students plan and manage their projects in cases where they do not have a group room. To gain an understanding of the requirements for such a planning tool, interviews were conducted with a number of students. Based on these interviews the problem domain was described and defined, and a list of user stories created. From these user stories a set of requirements and a product backlog, for use in the sprints, was created.

6.1 Problem Domain

In this section the problem domain will be examined and described, and from this the main objects and activities will be identified.

6.1.1 Analysis of Problem Domain

The problem domain for this project is universities using the Aalborg PBL model. The users of this sub-system will be students who do not have a physical group room and therefore have to rely on other methods for organizing their project work. What had to be examined were the tools provided by the university, and how the students currently organized their group work. Interviews have been conducted with three students with the aim of obtaining a general understanding of the work methods used in their semester projects. Similarly the tools provided by the university were examined in regards to their functionality and the students feelings towards them. The tools in question: Moodle and Mahara have been described in Section 2.1.

Users

The users of the system would be students at a university where the Aalborg PBL model is used. As a representative of the general student in such an environment, the students at AAU were considered the users of the system. As the system was meant to be used by all students, their different level of computer skills and physical work environment had to be taken into account. As an example not all students had a physical group room wherein they could organize their work. Similarly students had different levels of willingness and ability to look for and learn to use new tools.

Provided Tools

Mahara is presented to some students during their first year as a tool for them to use in their projects. From the interviews it seemed no instructions on how to use the system were provided. One student expressed she had looked at it, but found it too complicated. Another student had used it, but found it lacked some features and subsequently stopped using it. Students not using Mahara may stem from their lack of computer skills, or, as one expressed, that it seemed messy and difficult to use. Two students' opinion do not reflect all students' opinion of Mahara, however as these students have not used Mahara, their project groups consequently have not either.

As mentioned in Section 1.1.1 Moodle does not provide any general support for the Aalborg PBL model. Moodle is consequently not used in their daily work. They did, however, express a desire to use Moodle if it provided the necessary support for group based work. The interviews showed a common request; having all required functionality in a single tool, as they found it frustrating to use more than one.

Students' Work Methods

The systems used by the students for their group work were: Dropbox and Facebook. Dropbox was used as a file archive where all the material related to the product was uploaded. They kept track of tasks and decisions with a Word document. These Word documents were typically a summary of their meetings, so if one member had several tasks she might have to open and read several documents to gain an overview. They did not use any system for checking whether deadlines were met or not.

They had created a group on Facebook, and used this group as a communication portal. If one group member had an urgent message she would use text messages to reach the other members.

Internal meetings within the groups were arranged either at a preceding meeting, via Facebook, or phone. No shared calendar was used, therefore some group members would occasionally note a wrong time for the meetings and miss them. One student mentioned they solved this by having a group member take a screenshot of her calendar and uploading it to Dropbox. This gave the group a "shared" calendar. However, if a date for a meeting was changed a new screenshot would have to be uploaded which the students found annoying and troublesome.

In general the methods used by the interviewed students seemed improvised, and were

based on systems they were already familiar with. No extensive work had been put into finding new and better methods by the students.

Objects

Based on the interviews the main objects in the problem domain could be identified. It was clear that some shared calendar or timeline was needed in the system for keeping track of tasks and meetings. The two objects: tasks and meetings had to be represented in the system, as these were the two main activities in the interviewed students' group work. As summaries were written for every meeting, a summary object could be attached to every meeting once it had concluded. For the students without a physical group room there was also the question of booking a room for their meetings, both internal meetings as well as meetings with the supervisor.

Similarly some tasks would have a deadline that had to be set and kept track of.

The work done in the group must also be represented within the system. The result of every task should be either a modified or new document. Therefore an association between files and tasks should be present in the system.

Activities

The activities in the system were based on the conducted interviews. They were related to the various objects in the problem domain.

For tasks the related activities involved: setting up a task, assigning it to a person, setting an optional deadline, uploading the solution to the task, and editing and deleting tasks. Furthermore there was a request for a reminder when a deadline is due, therefore an activity would involve optionally choosing a reminder method.

Meeting activities would involve: setting a date, booking a room, adding the meeting to the timeline, adding participants to the meeting, and optionally sending a reminder to the meeting participants.

Files related to tasks would need to be uploaded.

6.1.2 Summary of the Analysis

Based on the analysis of the problem domain it was apparent that the tools provided by the university were not sufficient to support group work for all students. One of the major issues were how the students organized their group work. A solution to this would be to include an organizational tool in the solution developed in this project. The tool could be a timeline tool, where the students can add all activities related to their group work, e.g. meetings, tasks etc. As the tool will be implemented in Moodle it will be within an existing system used by the students, solving the issue of the students having to learn and use several tools for their group work.

Along with the visual timeline additional tools for supporting the timeline must be developed as well. The primary tool could be one for managing specific tasks assigned to different members of the project group, e.g. writing a section for their report.

This report will therefore deal with the implementation of a timeline tool and the required supporting tools that can be integrated in Moodle.

6.2 Requirements

To develop our system we must understand the requirements of the users first. We will base these requirements on the functionality requested by the potential end-users during the interviews. These requirements will be the basis of this subpart of MyMoodle, and thus it is important that they match the wishes of the customers. Based on the interviews we will first create user stories, see Section 6.2.1, and then distill these into a product backlog, described in Section 6.2.2, which is a fundamental part of the Scrum process.

6.2.1 User Stories

This section will cover the user stories that the product backlog described in Section 6.2.2 will be based on. User stories is a concept used in Scrum to describe the needs and desires of users and create a basis for the system definition. Section 6.1.1 identified the needed objects and activities. The connection between objects and activities can be seen in Figure 6.1. The data in Figure 6.1 was based on the interviews, and was used to gain an overview for creating the user stories.

Activities \ Objects	**Task**	**Meeting**	**Timeline**
Add object	X	X	X
Edit object	X	X	X
Delete object	X	X	X
Set a date/deadline	X	X	
Add users to object	X	X	
Email reminders	X	X	
Horizontal view of other objects			X
Vertical view of other objects			X

Figure 6.1. Table of Objects and Activities.

Within our project our user was the student, we used a general structure of user stories as follows:

> " *As a [user] I want [functionality].* "

The following user stories are in no particular order but will receive priorities in the product backlog:

- As a student I want to be able to see a horizontal timeline that makes it easy to see events within the next week/month that is related to a given project, course, semester, etc.
- As a student I want to be able to see a vertical timeline that expands downwards and makes it easy to see every event that is related to a given project, course, semester, etc.

- As a student I want to be able to add project tasks with a title and optionally deadline and note.
- As a student I want to be able to see all deadlines for project tasks in a coherent way.
- As a student I want to be able to categorize project tasks so I can quickly identify different types of project tasks via color coding.
- As a student I just finished a project task and want to mark it done by accessing the timeline.
- As a student I want to assign myself and other group members new project tasks.
- As a student I want to be alerted via email if I have any project tasks assigned to me with an exceeded deadline.
- As a student I want to see meetings with the group and with the supervisor on the timeline.
- As a student I want to be able to see a list of tasks, and be able to filter the list to show available, taken, and completed tasks, as well as tasks with a deadline.
- As a student I want to be able to sort the list of tasks, for example by deadline.

6.2.2 Product Backlog

The product backlog is an ordered list of requirements for the product in the form of user stories. The entire team can enter items into the product backlog, but it is up to the product owner to prioritize them. The priority of items in the product backlog should be decided based on criteria such as dependencies, value, date needed, etc. Each of the items should also have an estimate of how "big" the task is, i.e. how long it will take to complete.

Because of the iterative nature of the Scrum process the product backlog was not finished at the start of the project. Rather, it was continually worked on and expanded during the process, as new tasks appeared, and priorities shifted. At the start of each sprint the product backlog was used to create the sprint backlog. In general the sprint backlog was created with the highest priority items in the product backlog, while ensuring that the combined time needed for the tasks did not exceed the sprint duration.

Figure 6.2 shows the product backlog as it was at the end of the project, with all of the items and associated priorities.

No.	User story	Priority
1	As a student I want to be able to see a horizontal timeline that makes it easy to see events within the next week/month that is related to a given project, course, semester, etc.	10
2	As a student I want to be able to add project tasks with a title and optionally deadline and note.	10
3	As a student I want to be able to see all deadlines for project tasks in a coherent way.	9
4	As a student I want to assign myself and other group members new project tasks.	8
5	As a student I want to see meetings with the group and with the supervisor on the timeline.	8
6	As a student I want to be able to categorize project tasks so I can quickly identify different types of project tasks via color coding.	7
7	As a student I just finished a project task and want to mark it done by accessing the timeline.	7
8	As a student I want to be able to see a vertical timeline that expands downwards and makes it easy to see every event that is related to a given project, course, semester, etc.	6
9	As a student I want to be able to see a list of tasks, and be able to filter the list to show available, taken, and completed tasks, as well as tasks with a deadline.	3
10	As a student I want to be alerted via email if I have any project tasks assigned to me with an exceeded deadline.	3
11	As a student I want to be able to sort the list of tasks, for example by deadline.	2

Figure 6.2. The complete product backlog.

Development Method 7

In Section 5.1.2 Scrum of Scrums was chosen as the development method for the multi-project. As a result Scrum was used as the development method within each project group. Scrum does however contain elements which do not fit into a student project. To accommodate this it was modified to fit within the context and scope of a student project.

7.1 Implementation of Scrum

The implemented practices of Scrum were determined by their relevance in a student project, and the resources available within the project group. As an example the role of Scrum master requires a person to have undergone professional training. Similarly not all the meetings found within Scrum were sensible within the scope and context of the project.

7.1.1 Meetings

The Scrum Checklist [44] defines six meetings as part of Scrum:

1. *Estimation Meeting*
2. *Sprint Planning Meeting - Part 1 (Analysis of sprint backlog)*
3. *Sprint Planning Meeting - Part 2 (Design of sprint backlog)*
4. *Daily Scrum*
5. *Sprint Review*
6. *Sprint Retrospective.*

In the project the estimation meeting and the two planning meetings were combined into one. In this combined meeting all the items in the sprint backlog were analyzed, designed and their size estimated. Dependencies between sub-groups were discovered in the planning meeting and discussed at Scrum of Scrums meetings. The planning meetings were combined into one in an effort to optimize the planning process by reducing the parts of the meeting irrelevant in a student project.

Daily Scrum meetings were held as outlined in [44] with a few exceptions. Due to lectures they were not held at the same time everyday, and some days not at all. The Scrum of Scrums meetings were similarly not held every day. This was decided upon between the groups as it was not deemed necessary to have a coordination meeting every day.

Sprint retrospectives were conducted at the end of every sprint. The sprint retrospectives included some aspects of the sprint review, more specifically every group presented their achievements in the sprint. A new element was that the next sprint was planned at this meeting. Planning the sprint in the multi-project group involved selecting its length, the date for the next Scrum of Scrums, and the date for the sprint review meeting. The planning was required to fit everything around lectures and other study activities not related to the project.

The sprint review was replaced with a meeting at the end of the implementation where the system was presented to potential end-users. This was done as there were no dedicated customers or end-users, and finding people with time to attend a meeting at the end of every sprint was not feasible. Instead these meetings were held as one at the end of the implementation.

7.1.2 Roles

Scrum is a development method with many distinct roles. These are Scrum master, customer, team, product owner, manager, and end-user. Implementing all of these roles was not feasible in this project. As mentioned in Section 5.1.2 no one was available to take on the role of Scrum master. The role was therefore handled collaboratively between the group members, so everyone had the chance to attend at least one Scrum of Scrums meeting and be responsible for daily Scrum meetings. The responsibilities of the Scrum master were reduced to ensuring the formalities of the daily Scrum meeting were upheld and attending the Scrum of Scrums meetings. The roles of customer and product owner were similarly difficult to implement. There was no customer that had requested the product, and similarly there were not enough resources to set aside a person to be a dedicated product owner. The two roles were partially filled by the students interviewed in the analysis phase. The tasks handled by them were to prioritize the product backlog and participate in an end-user meeting where the final system was presented to them. From this they also implicitly filled the role of end-user. The role of manager was not assigned, as there are no corporate interests to take into account in a student project.

7.1.3 Artifacts

Scrum uses a set of artifacts to support and manage a project. Artifacts is a term describing tools and intermediate work products used to manage the change typically found in an agile project. In [44] 5 artifacts are mentioned:

- **Impediment Backlog** - A list of impediments which might lower the productivity of the Scrum team. It is the Scrum master's task to remove or limit their influence on the Scrum team.
- **Product Backlog** - A list of all the functionality a Scrum team wishes to implement in a project. The items are sorted based on the business value.
- **Selected Backlog** - A set of functionalities taken from the product backlog the Scrum team wishes to implement in a sprint.
- **Sprint Backlog** - A set of user stories derived from the selected product backlog that represents the work needed to implement the desired functionalities in a sprint.

- **Sprint Release** - A functioning piece of software released at the end of every sprint that could potentially be delivered to a client.

The impediment backlog was not used in the project. There was no Scrum master to maintain it and since no one had significant experience with Moodle or multi-projects predicting and identifying impediments was difficult. Internal impediments were handled at the daily Scrum, while impediments related to the other project groups were handled at the Scrum of Scrums meetings.

The other items were used according to their description.

Project Group Library 8

Since this system was part of a multi-project we had to be able to interact with the other projects. It was decided to implement this interaction by having a layer that every sub-system interacted with. This layer, called PGLib (*Project Group Library*), can be seen in Figure 8.1. The parts we worked with are the Timeline and the Task, shown to the left in Figure 8.1, and collaborated with the other sub-groups on the PGLib. The rest is divided amongst the other sub-groups as follows: one group worked with the Supervisor, another with the Blackboard, and the last with the Admin and the Project Group View.

The PGLib is a part of the plugin called Project Group. The Project Group plugin provides all features with respect to the notion of project groups explained in Section 1.2. This means that the Project Group plugin provides two things:

- A library, PGLib, that contains the functionality needed for project group specific blocks.
- A Project Group View that provided the same as the course view, which is standard in Moodle, i.e. a "home page" for each project group with blocks that are relevant to project groups, i.e. Timeline, Tasks, Supervisor, and Blackboard.

All blocks communicate with PGLib rather than each other as seen in Figure 8.1. This is the result of a design decision to have one place that contains all common functionality. In the figure it can be seen how the Project Group View uses most of the blocks, thus the Project Group plugin depends on the blocks and vice versa. Therefore the plugins cannot function as standalone plugins and instead the entire system is distributed as a combined plugin called MyMoodle.

PGLib is not a layer separating the blocks and Moodle. Therefore PGLib is not used to forward function calls to Moodle functions or general usage of Moodle's core. This is represented in Figure 8.1 with the encapsulation of all blocks in a box that interacts with Moodle. This implies that each block may interact with Moodle's core without accessing PGLib first.

Therefore PGLib only contains functionality that concerns project groups and supplied cross-blocks functionality.

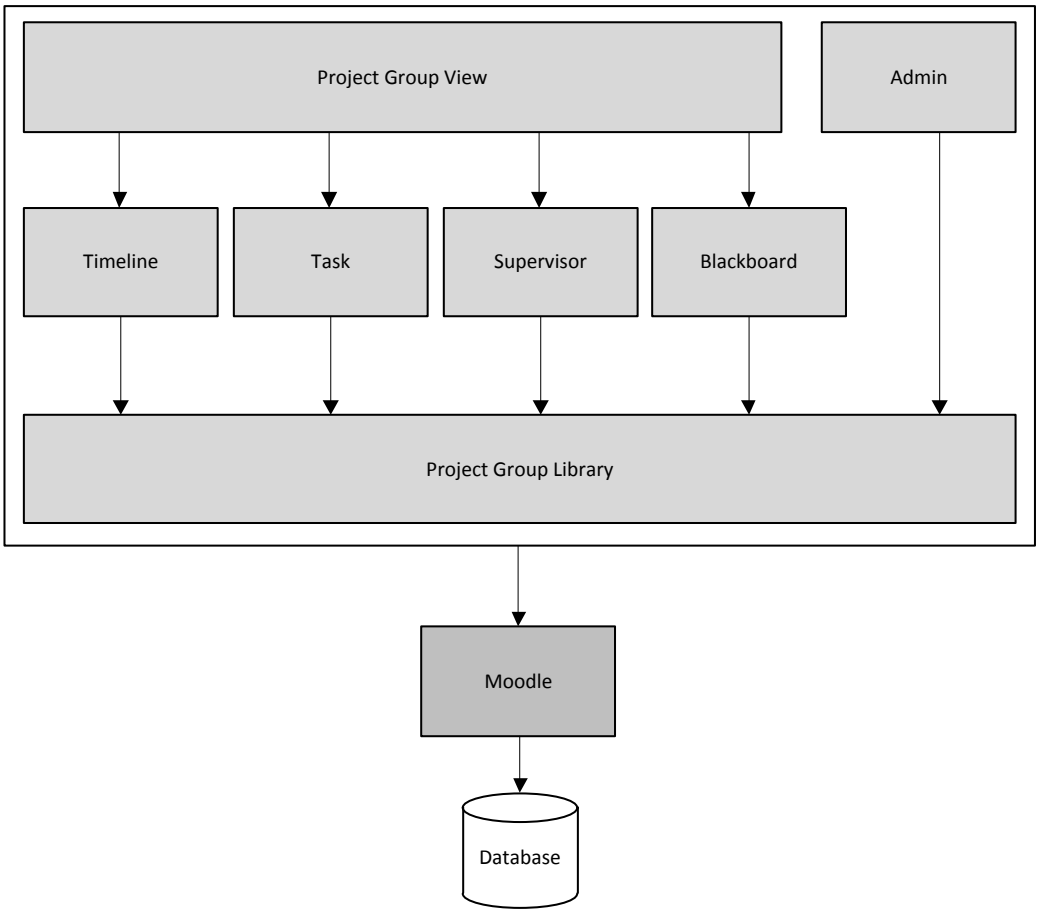

Figure 8.1. Multi-project design overview.

8.1 Implementing Project Group Library

PGLib was implemented by creating a project group folder in Moodle's `/local/` folder. Moodle's `/local/` folder is for plugins that do not suit the standard plugin formats: module, block, auth, enroll, etc. [29]. There are some examples of plugins where no standard plugin fits. These are found in Moodle's online documentation:

- Applications that extends the system level of Moodle, such as hub servers, amos servers, etc.
- Custom admin settings.
- Extending the navigation block.

There are functional differences between local and normal plugins, these are, again from Moodle's online documentation:

- Executed last during install/upgrade at all times.
- Best candidate for event handling.

- Can add admin settings to any settings page.
- Does not necessarily need a UI compared to other plugins, which are usually visible.

Another benefit of local plugins is that they are loaded along with most of Moodle's core, meaning if the Moodle core is available then the local plugin's library is also available. This means we can use functionality in the local plugin's library without worrying about including it beforehand, making it simpler to use the functionality.

From now on, whenever we talk about PGLib, we talk about the `lib.php` file located in the `/local/projectgroup/` folder. We did not place all functionality in the `lib.php` file in order to avoid conflicts. Instead files were created according to the naming scheme `lib_{block}.php`, where `{block}` was equal to the block's name without the "`block_`" prefix. For instance the library for tasks' functionality was put in a file called `lib_tasks.php`. This file was then included by the `lib.php` file.

9 System components

The timeline tool will consist of three individual components. The timeline implemented as a block, a tasks block to manage the tasks associated with a project, and an admin tool for managing the types associated with tasks. Types will be explained in Section 9.1.2. The concept of blocks in Moodle is described in Section 10.3 and admin tools in Section 10.5. These three components will implement the functionality requested by the interviewed persons, as seen in the product backlog in Section 6.2.2. Some of the components will share user stories and will need to work together to implement the solution. The timeline block is the main component and will visually present the activities related to a project. It must be noted that the components implemented as blocks must be compatible with courses in order to be in accordance with the standards set by Moodle.

9.1 Timeline Block

The timeline block handles the visual representation of the activities, such as tasks and meetings, created on a course or project group. The timeline must able to filter the information shown, so the user can customize her timeline. Filtering could involve choosing a timespan or selecting specific activities to show. A method must be implemented to create and store filters in the database, and to extract the correct data from the database based on said filters.

Most of the visual features required by the user stories will be taken care of by the timeline, and it will also provide shortcuts to other functionality, such as completing a task. Thus the timeline will take care of the user stories: 1, 3, 5, 7, and 8 of the product backlog in Figure 6.2.

9.1.1 Filters

Each timeline can be set up to display a unique set of activities, e.g. tasks and meetings. The activities displayed on a timeline are defined by a set of filters. Filters can be defined based on a list of properties for each type of activity. Such a list could contain creation date, deadline, completion date, and title. In addition a type of activity can be ignored by the timeline. To accommodate for new types of activities being introduced in the future the filters should be implemented modularly, for the timeline to be easily extendable.

An example of a filter could be; show all meetings between two dates, as well as all tasks with a deadline after a given date.

The filters will be customized for every type of activity.

9.1.2 Database Model

Since the timeline is designed to be implemented as a block, all settings for the timeline will be saved using Moodle's built-in data structure for blocks.

Retrieving Data for Different Activities

Each type of activity that is displayable on a timeline will have its own library file in PGLib. Based on the filters for a given timeline, the data will be displayed by two types of functions. Each activity type will have its own fetch function that given a filter will return an array containing the requested activities. These fetch functions are called by a controller function that merges the received data and displays them on the timeline. This gives a procedure as follows:

1. A timeline is requested and the associated filters are fetched from the database.
2. Based on these filters, the controller function will identify the activity types that are to be displayed.
3. The appropriate fetch function for each activity type will be called with the filters as the argument.
4. Each fetch function will return the activities matching the filters.
5. The controller function will merge the returned activities, sort them by date, and display them on the timeline.

9.1.3 Presentation

The user stories describe a desire for displaying the timeline either horizontally or vertically. The timeline will be flexible, meaning its time span is based on the activities it is displaying. The timeline will also be designed to display the beginning of each week, the week number, and indicate the current date. Different types of activities will be displayed on the timeline using different colors to make it possible to distinguish for instance meetings and tasks from each other.

9.2 Tasks Block

The task block handles the creation, editing and deletion of tasks.

The task block will seek to implement the user stories: 2, 3, 4, 6, 9, 10, and 11 of the product backlog in Figure 6.2. The functionality will be implemented through a public interface with the following methods:

- `create($data)` - Takes the associative array `$data`, seen in Figure 9.1, as argument containing the data of a task and adds the task to the database.

- `edit($modified_data)` - Takes the associative array `$modified_data`, seen in Figure 9.2, as argument containing the modified data of a task and updates the database with the new information.
- `delete($id)` - Takes the id of a task as argument and deletes the task and its associated data from the database.
- `complete($id)` - Takes the id of a task and updates its status to completed in the database.
- `fetch($identifier, $id)` - Takes a string and an id, both of which can be left empty. It will fetch the field for a task in the column name which matches `$identifier`. If `$identifier` is set to `'id'` it will fetch all the data for the task with the given id. If no arguments are given all tasks will be fetched from the database. It returns the associative array `$fetced_data`, seen in Figure 9.3.

The interface makes use of three specified associative arrays: `$data`, `$modified_data` and `$fetched_data`. These arrays must be formatted correctly for the functions to work properly. The definition of `$data` can be seen in Figure 9.1, `$modified_data` in Figure 9.2 and `$fetched_data` in Figure 9.3. The definitions should be understood as: *str created_ by* means there must be a field with the index `created_by` which contains a string.

```
$data = {
str created_by,
str title,
int type,
str desc,
date deadline (null allowed),
int[ ] assigned_users
}
```

Figure 9.1. $data.

```
$modified_data = {
int id,
str title,
int type,
str desc,
date deadline,
int[ ] assigned_users,
int status
}
```

Figure 9.2. $modified_data.

```
$fetched_data[int id] = {
str created_by,
str title,
int type,
str desc,
date deadline,
int[ ] assigned_users,
int status
}
```

Figure 9.3. $fetched_data.

9.2.1 Database Model

A task is a general term used to describe all work-related activities in a project. This could for example be writing content for the report, programming a component, or spell-checking a part of the report. These tasks have certain common attributes associated with them which must be reflected in the model. These are:

- **id** - Each task must have a unique id, as explained in Section 10.4.
- **Title** - Each task should have a title to serve as a short description of the task.
- **Description** - Optionally a description with more information can be added to the task.
- **Type** - Each task should have a type representing the type of work the task is concerned with. This could be writing or spell-checking a part of the report. There should be standard types, and it should be possible for a project group to create their own types should they want to.
- **Deadline** - It should be possible to set a deadline for a task. It should, however, be possible to create a task without a deadline.
- **Status** - Each task should have a status such as: *Not started*, *In progress*, etc. to keep track of the current progress of all tasks.
- **Assigned users** - It should be possible to assign users to a task. It should, however, be possible to have a task with no assigned users.

Furthermore every task needs some metadata which is only used by the system:

- **Timestamp** - A timestamp representing when the task was created.
- **Completion Timestamp** - A time representing when the task was completed.
- **Created by** - The user who created the task.

A `tasks` table containing most of these attributes will be created in the database. A field named `activity_id` is added as a foreign key to the activities table, linking a task to a course or project group.

The field not included in the `tasks` table is **Assigned users**. As a task can have zero to infinitely many people assigned to it. It is not sensible to represent this with fields in the `tasks` table, as it would require a restriction to be placed on how many users can be assigned to a task. Therefore an additional table named `task_users` is created. This table will contain the id field required by Moodle. Furthermore it will contain two fields: `task_id` as a foreign key to the id field of a task in the `tasks` table, and `user_id` as a foreign key to the id field of a user in the table `user`. The `user` table is part of the Moodle core and contains all registered users.

The model of tasks can be seen in figure 9.4.

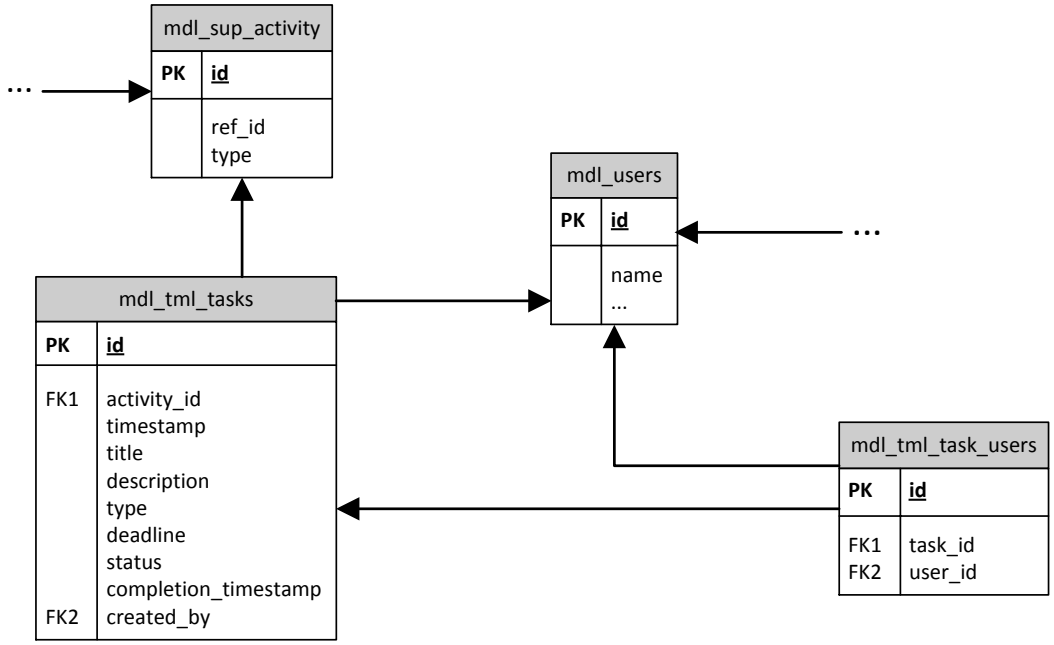

Figure 9.4. Database design for tasks.

Types

The `tasks` table field type represent the work activity related to the task. The issue with types is that some project groups might need to create their own types. The table `tasks_types` will be created to handle types. It will contain predefined types and user defined types. The table will contain the following fields:

- **id** - Field required by Moodle.
- **ref_type** - This field denotes whether the type is related to a course or a project group.
- **ref_id** - The id of the course or project group the type is associated with. This field is NULL for predefined types.
- **name** - The name of the type, e.g. Program.
- **color** - During the interview phase a request was made for each type being associated with a color. A type's color is stored in this field.

- **predefined** - Indicates whether the type is predefined or not. For predefined types the ref_id is ignored, and thus every predefined type with the right ref_type is shown for every course/project group.

When a task is added to a project group or course all predefined types will be fetched from this table along with all the user defined types associated with the course or project group. The types table can be seen in Figure 9.5. Relations to a table with ... indicates the table is used by other sub-systems of MyMoodle.

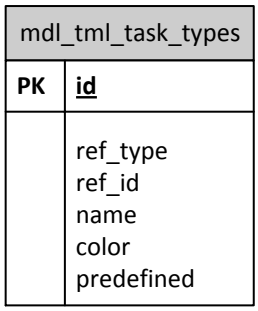

Figure 9.5. Database design for types.

9.3 Types Admin Tool

In Section 9.2.1 the data model for tasks was defined so that every task must be associated with a type. The intention is that the system can be suited to the students using it by creating predefined types that fit their field of study. To achieve this functionality an additional tool must be implemented. The tool will be implemented as an admin tool such that the staff is able to provide a set of predefined types. The only user story this tool is intended for is user story 6 in the product backlog in Figure 6.2.

The tool will consist of two Moodle forms. One for creating types and one for deleting them. Moodle forms are described in Section 10.2. To be in accordance with the model for types explained in Section 9.2.1, every type will be assigned a color and a name. Furthermore to account for blocks being compatible with both courses and project groups, types are assigned to one of these categories.

The admin tool is simple but solves the problem of providing default task types to the students.

Part III

Implementation

10 General Moodle

When developing software for Moodle there are some general requirements that must be followed. These requirements will be explained in this chapter.

10.1 Custom Pages

Custom pages are pages in Moodle that follow the Moodle layout but has custom content. To create a custom page in Moodle certain actions must be taken to ensure the Moodle library files are loaded, and to ensure the page follows the Moodle layout. In Code snippet 10.1 the code for an empty custom page can be seen.

```
1  <?php
2  require_once ("../config.php");
3  $PAGE -> set_url('/test/test.php');
4  $PAGE -> set_context(get_system_context());
5  $PAGE -> set_pagelayout('standard');
6  $site = get_site();
7  $PAGE -> set_title("$site->shortname: " . "Title");
8  $PAGE -> set_heading("Big Heading");
9  echo $OUTPUT -> header();
10 echo $OUTPUT -> heading("Small Heading");
11 echo $OUTPUT -> footer();
```

Code snippet 10.1. PHP code for an empty custom page.

First the `config.php` file is included. This file contains various constants about the Moodle installation, and needs to be included on every page. The variable `$PAGE` is used to set variables concerning the page [30]. The function `set_url()` is then called, which takes an URL that Moodle uses when reloading the page, for example when block editing is turned on or off.

On line 4 the function `set_context()` is called, informing Moodle of the page's context. This can for instance be a specified course or project group. In Code snippet 10.1 the context is set to the system context.

The page layout, which is set on line 5, affects how the page looks, e.g. if it should have blocks on both sides, only the left side or not at all. In this example the standard layout is used, which allows blocks on both sides.

On line 6 and 7 the title of the page is set. The function `get_site()` is used, which returns a reference to Moodle's front page. This is used to get the "shortname" of the Moodle installation, i.e. in our Moodle installation "SW6MD". The heading of the page is set on line 8. On line 9-11 the `$OUTPUT` object is used to display the page's content.

The page generated by this code can be seen in Figure 10.1

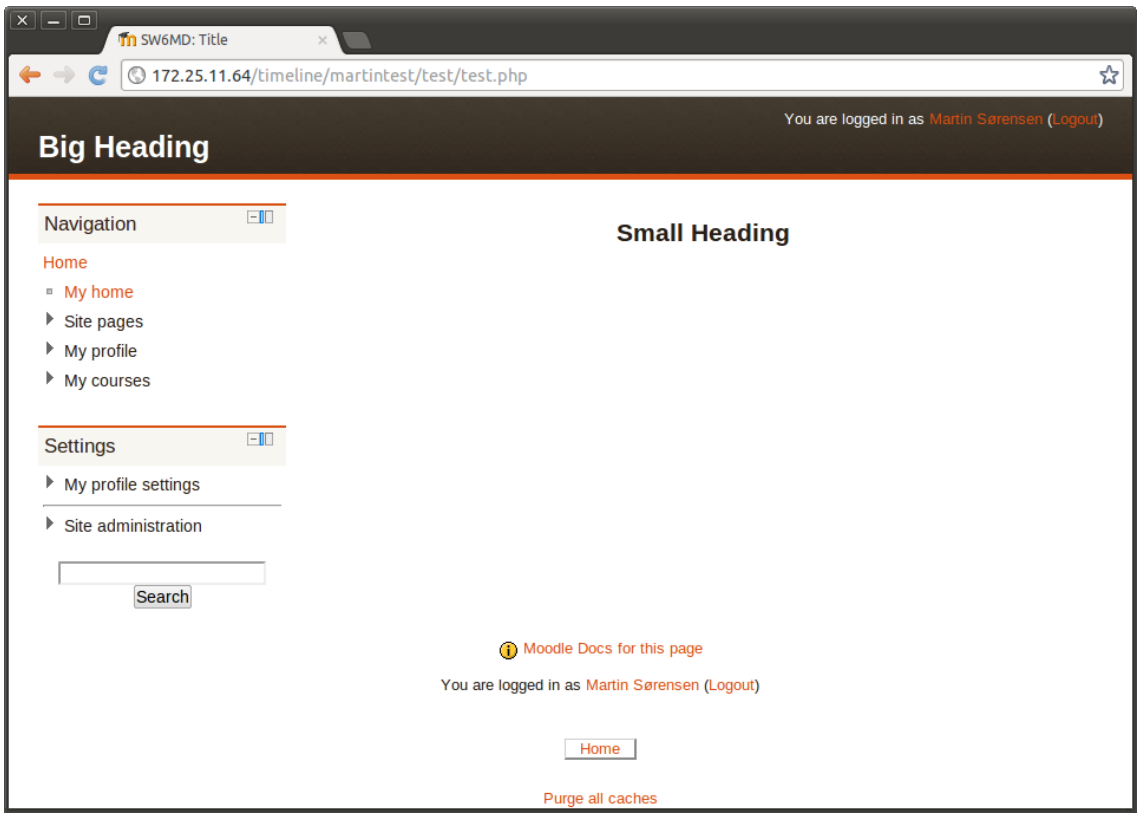

Figure 10.1. Example Moodle page.

10.2 Moodle Form

Moodle has a built-in class used to create web-forms, called Moodle form, which contains various functions for adding elements, setting layout, retrieving input and so on.

This paragraph on the `definition()` function is based on the Moodle Docs [28]. A sample Moodle form can be seen in Code snippet 10.2. Setting up the form itself is done in the function `definition()` beginning on line 6. This calls various functions on `$this->_form` to create the form. The function `addElement()` is used to add elements, such as check boxes. On line 10 in the code snippet a text field is added. The first parameter defines the type of element, the second the name used to reference the element, and the rest of the parameters depend of the type of element. The third element is often the text displayed with the element. The function `setDefault()` can be used to set the default value of elements like text fields, while the function `setSelected()` can be used on select elements to set the element which is selected by default. An example of `setDefault()` can be seen on line 11 where the textfield "email" is given the string "default value" as its default value. Buttons can be added with `addElement()`, except for the special

```
1  <?php
2  require_once($CFG->libdir.'/formslib.php');
3
4  class testform_form extends moodleform {
5
6      function definition(){
7          $mform =& $this->_form;
8          $mform->addElement('header', 'header', 'Form Header');
9
10         $mform->addElement('text', 'email', 'Email');
11         $mform->setDefault('email', 'Default Value');
12         $mform->addRule('email', 'Must be a valid email', 'email', null, '
               client');
13
14         $mform->addElement('textarea', 'desc_text', 'Description', 'wrap="
               virtual" rows="10" cols="20"');
15         $mform->setDefault('desc_text', 'Default Text');
16         $mform->addRule('desc_text', 'Description is required', 'required',
                null, 'client');
17
18         $this->add_action_buttons(true, 'Submit');
19     }
20 }
```

Code snippet 10.2. The PHP code for a Moodle form.

submit and cancel buttons, which are added using the function `add_action_buttons()`. `add_action_buttons()` is, unlike the other functions, called on `$this`, see line 18.

To show a form on a page the `display()` function is called on the form object. The `get_data()` function is used to check if anything has already been entered into the form, and retrieve it if this is the case. For `get_data()` not to return false the form must first be validated. Validation can be implemented in two ways, either the function `addRule()` is used to add rules to elements in the `definition()` function, or a `validation()` function is defined in the form class.

The `addRule()` function takes the name of the element to add the rule to, an error message to be displayed if the rule is broken, and the type of the rule. More arguments can be passed depending on the type of rule. Some of the rule types can be seen here:

- required — Value is not empty.
- maxlength — Value length must not exceed given number.
- regex — Value must match given regular expression.
- email — Value is a correctly formatted email.

If the `validation()` function is defined it must return an associative array on the form `$errors["elementtitle"] = "error"`. For the form to be validated this array must be empty. The explanation of validation in Moodle forms is based on the documentation for PEAR HTML_QuickForm [32].

An example of a Moodle form using `addRule()` validation, can be seen in Code snippet 10.2. The form made by this code can be seen in Figure 10.2.

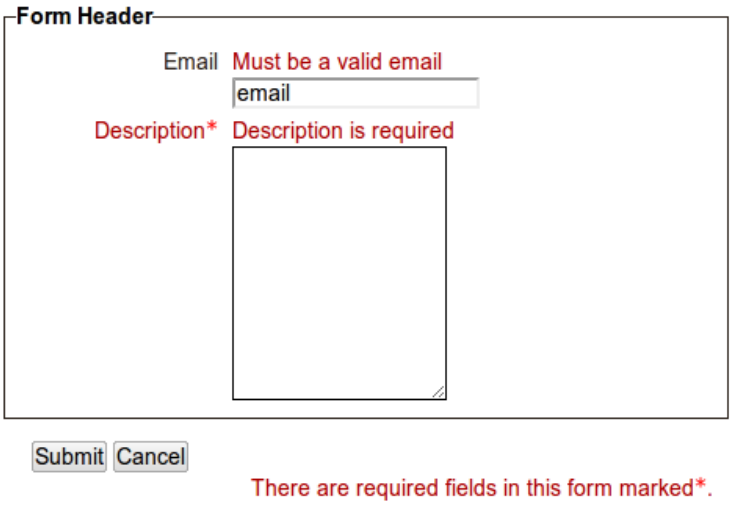

Figure 10.2. Example Moodle form.

10.3 Structure of a Block

A block is a type of plugin for Moodle, which purpose is to add a set of functionality to the user. There are some requirements for a block to be compatible with Moodle [26]:

- Must have a folder with its name in the blocks folder.
- The folder must contain two files.
  - `block_NEWNAME.php`, where `NEWNAME` is the block's name.
  - `version.php`, which contains the block's version number and the required Moodle version's number.
- Language files are put in the folder under `/lang/en/` for english languages. Additional languages can be added, and Moodle will automatically select the one preferred by the user.
- Forms for configuring a block are defined in the `edit_form.php` file.

The general structure of the `block_NEWNAME.php` file is as seen in Code snippet 10.3, slightly modified from the Moodle documentation [26]. Code snippet 10.3 shows a working example of a block that can be expanded. A working example of a `version.php` file can be seen in Code snippet 10.4. In this example `$plugin->version` is the version of the block. This should be updated whenever important changes are made.

To allow users to edit the block, Moodle requires an `edit_form.php` file to be present. A very simple version of this file can be seen in Code snippet 10.5. The code snippet allows the user to configure the text shown in the body of the block. `block_edit_form` is an extension of Moodle forms, thus it contains the same functionality as a Moodle form. As described in Section 10.2 the second element given as an argument to the `addElement()` function call on line 11 in Code snippet 10.5 is the name used to reference the element. This argument must have the prefix "config_" for elements in the `edit_form.php` file. The field is stored as everything following "config_", i.e. on line 11 a field called "text" is

```php
<?php
class block_NEWNAME extends block_base {
    public function init() {
        $this->title = get_string('NEWNAME', 'block_NEWNAME');
    }

    public function get_content() {
        if ($this->content !== null) {
            return $this->content;
        }

        $this->content         =  new stdClass;
        $this->content->text   = 'The content of our NEWNAME block!';
        $this->content->footer = 'Footer here...';

        return $this->content;
    }
}
```

Code snippet 10.3. General structure of the block_NEWNAME.php file.

```php
<?php
$plugin->version = 2012041300;  // YYYYMMDDHH (year, month, day, 24-hr time)
$plugin->requires = 2011120100; // YYYYMMDDHH (This is the release version for Moodle 2.0)
```

Code snippet 10.4. Structure of the version.php file.

added to the config object. The values of these configurable fields in the config object can be fetched from the block as shown in Code snippet 10.6 on line 2.

```php
<?php

class block_NEWNAME_edit_form extends block_edit_form {

    protected function specific_definition($mform) {

        // Section header title according to language file.
        $mform->addElement('header', 'configheader', get_string('blocksettings', 'block'));

        // A sample string variable with a default value.
        $mform->addElement('text', 'config_text', get_string('blockstring', 'block_NEWNAME'));
        $mform->setDefault('config_text', 'default value');
        $mform->setType('config_text', PARAM_MULTILANG);

    }
}
```

Code snippet 10.5. Simple example of the edit_form.php file [26].

In Code snippet 10.3 the function `get_string()` was used, this function is used to get strings from language files, either the block's language file or Moodle's core language files. Language files are a way to localize Moodle into different languages. Therefore there should be a language file for the language you implement a block in. An example of a

```
1 if (!empty($this->config->text)) {
2     $this->content->text = $this->config->text;
3 }
```

Code snippet 10.6. Fetching config settings in the block.

standard language file can be found in Code snippet 10.7. This file is placed in the folder `/lang/` under the appropriate name. Since this language file is for the English language it is placed in the folder `/lang/en/`. The file is named after the standard block file, thus: `block\_NEWNAME.php` will be the name.

```
1 <?php
2 $string['pluginname'] = 'NEWNAME the Block';
3 $string['NEWNAME'] = 'NEW NAME';
```

Code snippet 10.7. Simple example of a block_NEWNAME.php language file.

10.4 XMLDB

XMLDB is Moodle's built-in database abstraction layer. It supports making database tables in XML without manually editing the database, and it also has functions which allow access to the database.

install.xml

This section is based on the page XMLDB defining an XML structure [31] from the Moodle documentation. Every table that a given plugin needs are entered into a file called `install.xml`. The structure of the install.xml file is as follows:

XMLDB

TABLES

FIELDS

KEYS

INDEXES

These elements are defined as follows:

- XMLDB

  This element contains general information about the file, such as its location.
- TABLES

  This element contains a TABLE element for every table. All TABLE elements have a **name** attribute.
- FIELDS

  The container element for the FIELD elements. A FIELD element exists for each field in a table. Some common attributes are **name**, the name of the field, **type**, the type of the field such as int or char, and **length**, the size of values in the field.

- KEYS

  In KEYS any primary, unique, and foreign keys are defined. Each key must have the attributes **name**, the name of the key, **type**, the key can be either primary, unique, or foreign, and **fields**, denoting the fields that make up the key.
- INDEXES

  In INDEXES all the indexes are defined. Each index has the attributes **name**, **unique**, and **fields**. Moodle automatically creates indexes for all keys.

Moodle Data Manipulation API

Moodle has built-in functions for accessing the database without the use of SQL queries. These built-in functions are called on the global object `$DB`. A few of these functions will be explained here, based on [27].

record_exists() Returns true if a record exists for which a conditional is true.

Example:

`record_exists('example',array('id'=>5))` returns true if a record which **id** equals "5" exists in the table `example`.

get_records() Returns all records where a conditional is true.

Example:

`get_records('example',array('name'=>john))` returns all records from the table `example` where the field **name's** value equals "john".

insert_record() Inserts a record into a table, and returns the id of the newly created record.

Example:

`insert_record('example',(object)array('name'=>john))` inserts a record in the table `example` where the field **name's** value equals "john".

The API also allows SQL queries in case none of the other supplied functions are sufficient for a given request.

get_records_sql() Returns a number of records using an SQL statement.

Example:

`get_records_sql('SELECT * FROM {example}')` returns every field of every record in the table `example`.

10.5 Admin Tools

Admin tools are plugins available only to the admins on a Moodle installation. They are accessed through the site administration sidebar that can be seen in Figure 10.3. For an admin tool to function in Moodle, it must have a folder with its name in the `/admin/` folder, containing the following files:

- `version.php` - A file containing the version number for the plugin.
- `settings.php` - A file defining menu points for the plugin in the site administration sidebar.

Furthermore it should include a language file similar to that of a block described in Section 10.3. The `version.php` file is similar to that of a block. An example of a `seetings.php` can be seen in Code snippet 10.8

```
1 <?php
2 $ADMIN->add('root', new admin_category('tasktypes', 'Task Types'));
3 $ADMIN->add('tasktypes', new admin_externalpage('managetasktypes', 'Manage
    types for tasks', $CFG->wwwroot.'/'.$CFG->admin.'/tool/types/index.php'
    ));
```

Code snippet 10.8. Example of a settings.php file.

On line 2 a new root menu point is added to the site administration bar. The `admin_category` class is build into Moodle. Its first argument, in this case `tasktypes`, is its reference name, and its second argument is the text that will be displayed in the administration bar. On line 3 a child is added to the `tasktypes` menu point. This is done by adding an `admin_externalpage` object. The `admin_externalpage` class is given 3 arguments: `managetasktypes` is its reference name, `'Manage types for tasks'` is the string displayed in the menu point, and the last parameter is the path to the file that is to be loaded when the menu point is clicked.

It is possible to add as many menu points as needed to an admin plugin.

The functionality for admin plugins are either placed in a `lib.php` file, or in the files loaded by the `admin_externalpage` menu points. A sidebar with the menu points defined in Code snippet 10.8 can be seen in Figure 10.3.

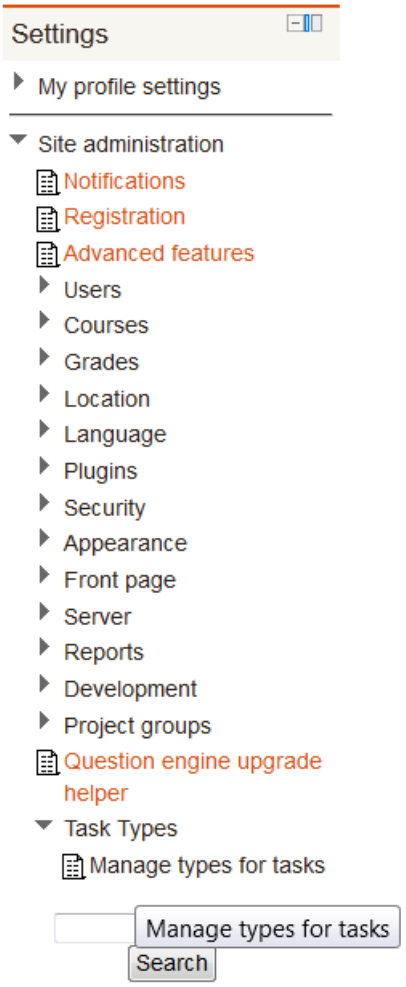

Figure 10.3. An Administration Sidebar.

Timeline 11

In this chapter the code for the Timeline block will be explained. The Timeline consists of two major parts: filtering and presentation.

11.1 Presentation of a Timeline

A timeline's appearance is as mentioned in Section 9.1.1 determined by a set of filters. Based on the configuration of a timeline, it is printed either vertically or horizontally. Both timeline formats have a black line representing the time. Furthermore, both of them have indicators separating the weeks from each other, and a red line indicating the current date.

The timeline is designed so that a displayed activity is always as wide as a day, e.g. if a timeline is 700 pixels wide and it shows five weeks, each week is 140 pixels and so each displayed activity is 20 pixels wide. To enhance readability each displayed activity is, however, only 16 pixels wide and have a 2 pixels margin on all sides. If more than one activity is to be displayed on a single day, these activities would be displayed next to each other.

Each activity on the timeline is designed to have a description, a timestamp representing a creation date, deadline, or similar, and a number of users attached. When hovering the mouse over an activity, the timeline displays a box with additional information on this activity. For a task, this information could be the title, deadline, description and persons responsible for this task. Each box will include a hyperlink to a page specified by each activity type, which will show more information about the specific activity. All the boxes are generated when the site is loaded and initially hidden.

The boxes' visibility is controlled by the JavaScript in Code snippet 11.1. Line 1 includes jQuery, which is a third party JavaScript library, that provide simplified HTML document traversing, event handling, etc. [10]. jQuery has built-in functions to detect when an HTML tag is hovered over with the mouse. These functions are used to show and hide the boxes generated when the site is loaded.

Every activity must be given a color. This color can be either activity type or activity specific, e.g. tasks can be of different types and each type can have its own color, while all meetings are of one color. This enables the user to distinguish between e.g. tasks and

```
<script type="text/javascript" src="//ajax.googleapis.com/ajax/libs/jquery
    /1.7.2/jquery.min.js"></script>
<script type="text/javascript">
    $(document).ready(function() {
            $(".tl_bullet").mouseover(function() {
            $(this).children().show();
        });

        $(".tl_bullet").mouseout(function() {
            $(this).children().hide();
        });
    });
</script>
```

Code snippet 11.1. Detect when the mouse overs above an element.

meetings.

When a timeline is to be displayed, its width (if horizontal) or height (if vertical) will always be known. This enables the timeline to calculate a relationship between the activities it must show and the space it has available. If the timeline is horizontal, only a set amount of pixels are available – 700 pixels. If the timeline is vertical, its height can expand as much as needed in order to show all activities on it. Based on the activities and the time span these cover the timeline will calculate where to display them. An example of a horizontal and vertical timeline can be seen in Figure 11.1 and Figure 11.2, respectively. Both examples show a five week time span with 10 activities, and shows the box for the orange activity in "Uge 21".

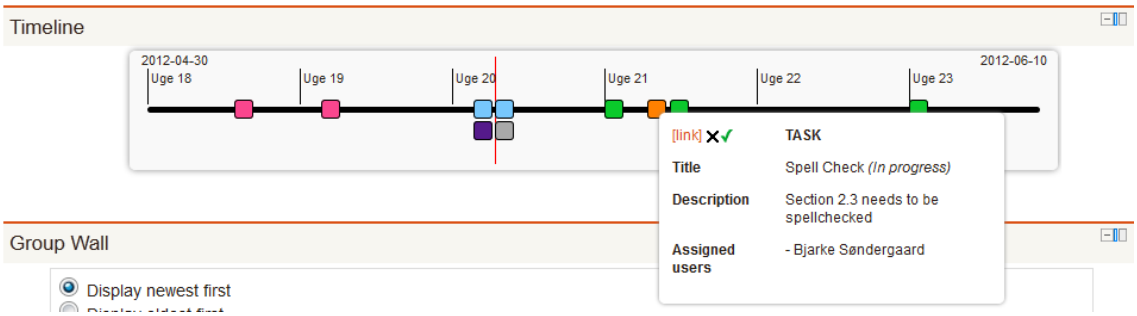

Figure 11.1. A horizontal timeline with 10 activities on it.

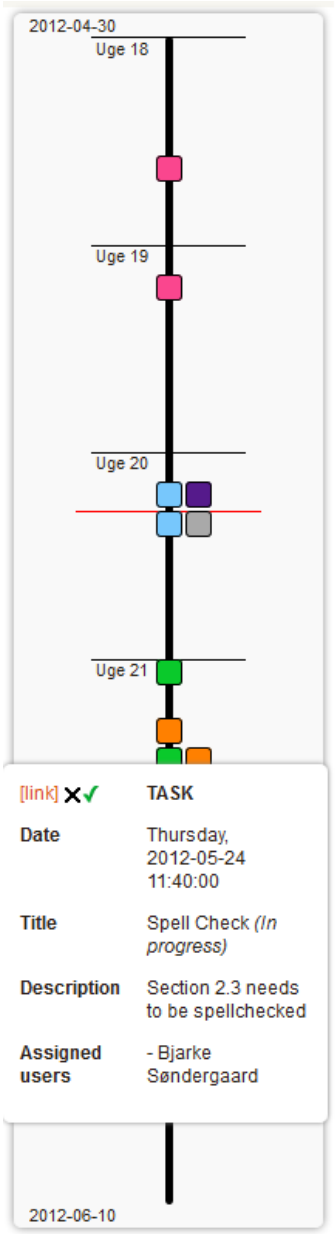

Figure 11.2. A vertical timeline with 10 activities on it.

11.2 Timeline Functionality

The original intention was to implement the timeline following the MVC model where the code is split up into different layers. But since the timeline is implemented as a block in Moodle, this is not possible. The functionality for the timeline is placed in the `block_timeline` class that extends the `block_base` class provided by Moodle. This class is located in the file `block_timeline.php`, and contains five primary functions.

controller_get_timeline

Code snippet 11.2 shows the function `controller_get_timeline()`, that fetches and prepares all data before it is printed. It returns an array of activities to display. The filters

```
1  protected function controller_get_timeline() { // Controller
2      $input = array();
3      $data = array();
4
5      if (isset($this->timeline_filter)) {
6          foreach ($this->timeline_filter AS $key=>$value) {
7              if ($value['enabled'] == 1) {
8                  $filter = array();
9                  foreach ($value AS $ikey=>$ivalue) {
10                     if ($ikey == 'enabled') {
11                         continue;
12                     }
13                     if ($ivalue['on'] == 1) {
14                         $filter[] = array(
15                             'ident' => $ikey,
16                             'type' => $ivalue['modifier'],
17                             'args' => $ivalue['values'],
18                         );
19                     }
20                 }
21                 $input[] = array(
22                     'func_name' => $key,
23                     'filter' => $filter,
24                 );
25             }
26         }
27
28         $data = $this->model_fetch_data($input);
29     }
30     else {
31         $data = array();
32     }
33     if (!is_array($data)) {
34         $data = array();
35     }
36
37     usort($data, "cmp_by_date_asc");
38
39     return $data;
40 }
```

Code snippet 11.2. controller_get_timeline method in block_timeline.

explained in Section 11.3 have been saved in a private variable `$this->timeline_filter`.

On line 6, a loop is initiated which runs through all filters. If a filter is enabled it will be added to the `$filter` array, which contains all enabled filters. This can be seen on line 7 to line 26.

The enabled filters are given as an argument to the `model_fetch_data()` function, which returns a single array `$data`, containing the requested activities. The `$data` array is then sorted by date on line 37 and returned.

model_fetch_data

The `model_fetch_data()` function, shown in Code snippet 11.3, finds activity types based on the passed filters. All activity types have a private function in the class named `fetch_{name}_data` where `{name}` is the name of the activity type. These functions calls

```
1  protected function model_fetch_data($input) { // Model
2      $output = array();
3
4      foreach($input AS $k=>$v) {
5          if(empty($v['func_name'])) {
6              continue;
7          }
8
9          $function_name = "fetch_" . $v['func_name'] . "_data";
10
11         if(!method_exists($this, $function_name)) {
12             return "function " . $function_name . " does not exist!!!";
13         }
14         $filter = array();
15         if (isset($v['filter'])) {
16             $filter = $v['filter'];
17         }
18         $temp = $this->$function_name($filter);
19         $output = array_merge($output, $temp);
20     }
21     return $output;
22 }
```

Code snippet 11.3. model_fetch_data method in block_timeline.

a function in PGLib that fetches activities based on the filters for that activity type. On line 18 these functions are called and their returned data is saved in the variable `$temp`. On line 19 the data in the variable `$temp` is merged with already fetched activities. On line 21 the array of requested activities is returned.

get_content

The `get_content()` function, shown in Code snippet 11.4, is called by Moodle when it is requested to display a block. In the Timeline block this function is used to find the format of a given timeline, set the filters, and call the `print_view()` function that prepares the display of the timeline. The functions assigns the HTML of the timeline to the variable `$this->content->text` and returns `$this->content`. Moodle prints what `get_content()` returns, thus the timeline's HTML is displayed.

```php
public function get_content() {
    global $SESSION;
    if ($this->content !== null) {
        return $this->content;
    }
    $filter_group = '';

    $parent_context = $this->context->get_parent_context();
    $parent_contextlevel = $parent_context->contextlevel;
    if ($parent_contextlevel == "50") {
        $filter_group = 'COURSE_ID';
        $SESSION->tasks_context = array(1, $parent_context->instanceid);
    }
    if ($parent_contextlevel == "55") {
        $filter_group = 'GROUP_ID';
        $SESSION->tasks_context = array(2, $parent_context->instanceid);
    }
    if (!isset($this->config->filter)) {
        ...
        // if no configuration exists for this concrete timeline, then
            create a default filter.
        ...
    } else {
        $this->timeline_filter = $this->config->filter;
    }

    if (isset($this->config->timeline_format_radio) && !empty($this->config
        ->timeline_format_radio)) {
        $format = $this->config->timeline_format_radio;
    } else {
        // else fetch the default format from the administration settings
            concerning the timeline.
        ...
    }

    $this->content = new stdClass;
    $this->content->text = $this->print_view($format);

    return $this->content;
}
```

Code snippet 11.4. get_content method in block_timeline.

print_view

Finally the `print_view()` function, shown in Code snippet 11.5, is the one that prepares the timeline. This function identifies whether to use the vertical or horizontal timeline. The activities that the two different layouts will display are the same, but since the implementation differs, they are implemented in each their private function. These are described in detail in Section 11.1.

11.3 The Filters

The content displayed by a timeline is defined by its filters. The filters are hardcoded into the `edit_form.php` file. All filters are implemented using the Moodle forms, see Section 10.2. An example could be filtering on the creation date of a task, which can be seen in Code snippet 11.6.

```
protected function print_view($format) {
    if ($format == '2') {
        return $this->print_vertical();
    }
    else {
        return $this->print_horizontal();
    }
}
```

Code snippet 11.5. print_view method in block_timeline.

```
$mform->addElement('advcheckbox', 'config_filter[assignments][DATE][on]',
    get_string('timeline_filter_on_setting', 'block_timeline'), get_string(
    'timeline_filter_on_setting_creationdate', 'block_timeline'));
$mform->addElement('select', 'config_filter[assignments][DATE][modifier]',
    get_string('timeline_filter_modifier_setting', 'block_timeline'), array
    (get_string('filter_modifier_between', 'block_timeline'), get_string('
    filter_modifier_is', 'block_timeline')));
$mform->addElement('text', 'config_filter[assignments][DATE][values]',
    get_string('timeline_filter_values_setting', 'block_timeline'));
$mform->setType('config_filter[assignments][DATE][values]', PARAM_TEXT);
$mform->addHelpButton('config_filter[assignments][DATE][values]', '
    datevalues', 'block_timeline');
$mform->addElement('html', '<hr />');
```

Code snippet 11.6. Filter on creation date of an Assignment.

Line 1 inserts a checkbox to enable or disable this activity type. Line 2 inserts a select-box enabling the user to pick between two values, either activities within a time span or on a specific date. Line 3 adds the text field where the date(s) for the filter can be entered. Line 5 adds the help icon, providing assistance in how to fill out the previously mentioned text field. Line 6 adds a horizontal line in the HTML to separate the activities.

Due to the filters following the same format, only the filter in Code snippet 11.6 is shown. Without filters for a given activity type it is not possible to display any activities of this type, as it must at least have the enable/disable filter. Moodle will handle when to show the form, e.g. when a user is editing a timeline.

Tasks 12

When implementing tasks there were several things that were not directly supported by Moodle, thus making the implementation more difficult. Tasks are an activity type, which is displayed on a timeline in Figure 11.1 and Figure 11.2

12.1 Creating Tasks

Tasks are created via the create form, which can be seen in Figure 12.1. The form has fields corresponding to the data which constitutes a task. Additionally the form supports creating new types, if none of the existing ones fit the nature of the task. The **Assigned Users** select box contains the names of all users who are assigned to the relevant course or project group.

When the create task page is displayed the Moodle form `create_task_form` is shown.

This form uses the variable `$SESSION->tasks_context`, to determine which users and types to display in the form. `$SESSION->tasks_context` is set whenever a course or project group page is visited, and contains an array of variables which are able to uniquely identify a course or project page. The **New type** and **Deadline** parts of the form are disabled per default, and become enabled when the corresponding checkboxes are checked. This is done using the `disabledIf()` function, which can disable form elements based on various criteria.

When the save task button is pressed, assuming that the form's content pass the validation rules, the user is sent back to the course or project page from whence they came. Here, the task is created by the tasks block. First, the block uses the function `get_data()` on the `create_task_form` to check if data from the form is present. If it is the block proceeds with creating the task. If a new type was created via the create task form it is necessary to save this new type in the database before saving the task. This is because a task is required to have a type, and a new type will not have an id before it has been created. Next, an associative array with all of the information entered in the form is created, and passed to the `create()` function defined in Section 9.2 to create the task.

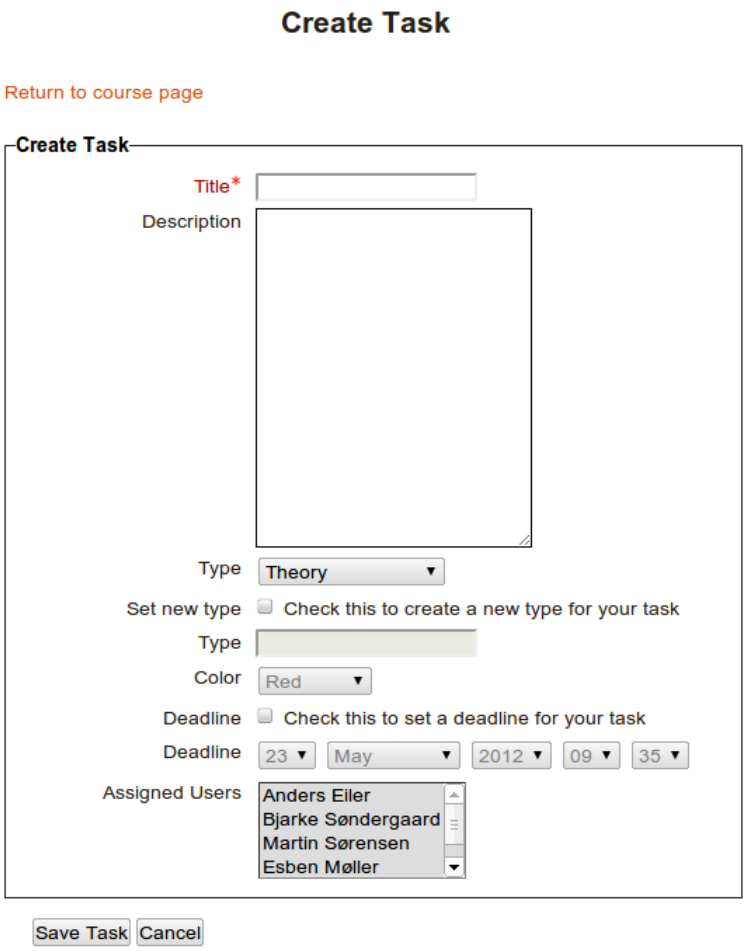

Figure 12.1. The task create form.

12.2 Editing Tasks

Task editing is done using two main parts, a select form and an edit form. The select form allows the user to select the task to be edited, and can be seen in Figure 12.2. The list of tasks shown in the drop-down list are the ones related to the course or project group the user accessed the select form from.

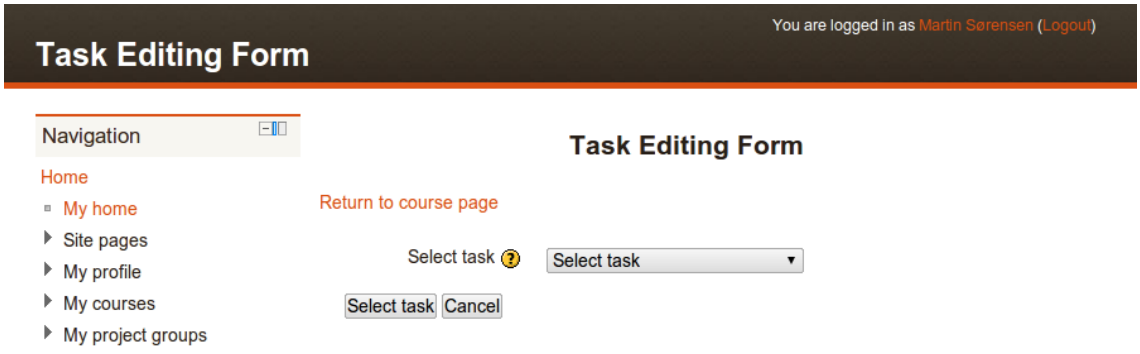

Figure 12.2. The select form used when editing tasks.

Once a task has been selected, the edit form, seen in Figure 12.3, is shown. The fields are loaded with the data from the task. The two fields with the label **Select Users** are used to add and remove users from the task. The right field contains the names of every user currently assigned to the task. The left field contains every user, who are related to the relevant course or project page, but not assigned to the task.

When editing a task the select form is first displayed. When a task is selected its id is saved in `$SESSION->edit_task_id`, and the edit form is displayed. With this form the user is presented with four buttons, each with different functionality. When pressing the add or remove buttons the task is fetched, and the list of assigned users is expanded or shrunk to reflect the user's selection.

The code used to remove users can be seen in Code snippet 12.1. First the selected task is fetched and then modified to fit the format expected by `edit()`. Now there are two arrays with users: `$fromform->assigned_users`, containing the users that are to be deleted, and `$fetched['assigned_users']`, which contains the users previously assigned to the task. Both of these arrays are of the form `array[x] = y` where y is the id of the user in question. Next, on line 7-14 these arrays are traversed, removing the users in `$fromform->assigned_users` from `$fetched['assigned_users']`. Whenever the values `$fv` and `$v` are the same on line 9 the user to be removed has been found, and it is unset in `$fetched['assigned_users']`. After this `edit()` is called, and the edit form is displayed again to allow the user to make further modifications.

```
1  <?php
2  $fetched = fetch('id', $SESSION->edit_task_id);
3  $fetched = $fetched[$SESSION -> edit_task_id];
4
5  unset($fetched['created_by']);
6
7  $fetched['id'] = $SESSION->edit_task_id;
8  $fetchedusers = $fetched['assigned_users'];
9
10 foreach ($fromform->assigned_users as $k => $v) {
11     foreach ($fetchedusers as $fk => $fv) {
12         if ($fv == $v) {
13             unset($fetchedusers[$fk]);
14             break;
15         }
16     }
17 }
18
19 $fetched['assigned_users'] = $fetchedusers;
20 edit($fetched);
21
22 $eform = &new edit_task_form('edit_tasks.php');
23 echo $eform -> display();
```

Code snippet 12.1. The PHP code used to remove users.

The code for adding users is similar, however since it is not necessary to find the specific user to be added a single `foreach()` is sufficient.

If the **Save Task** button is pressed the task is fetched and modified to fit `edit()` like it was when removing users. Next the task is modified with the information the user

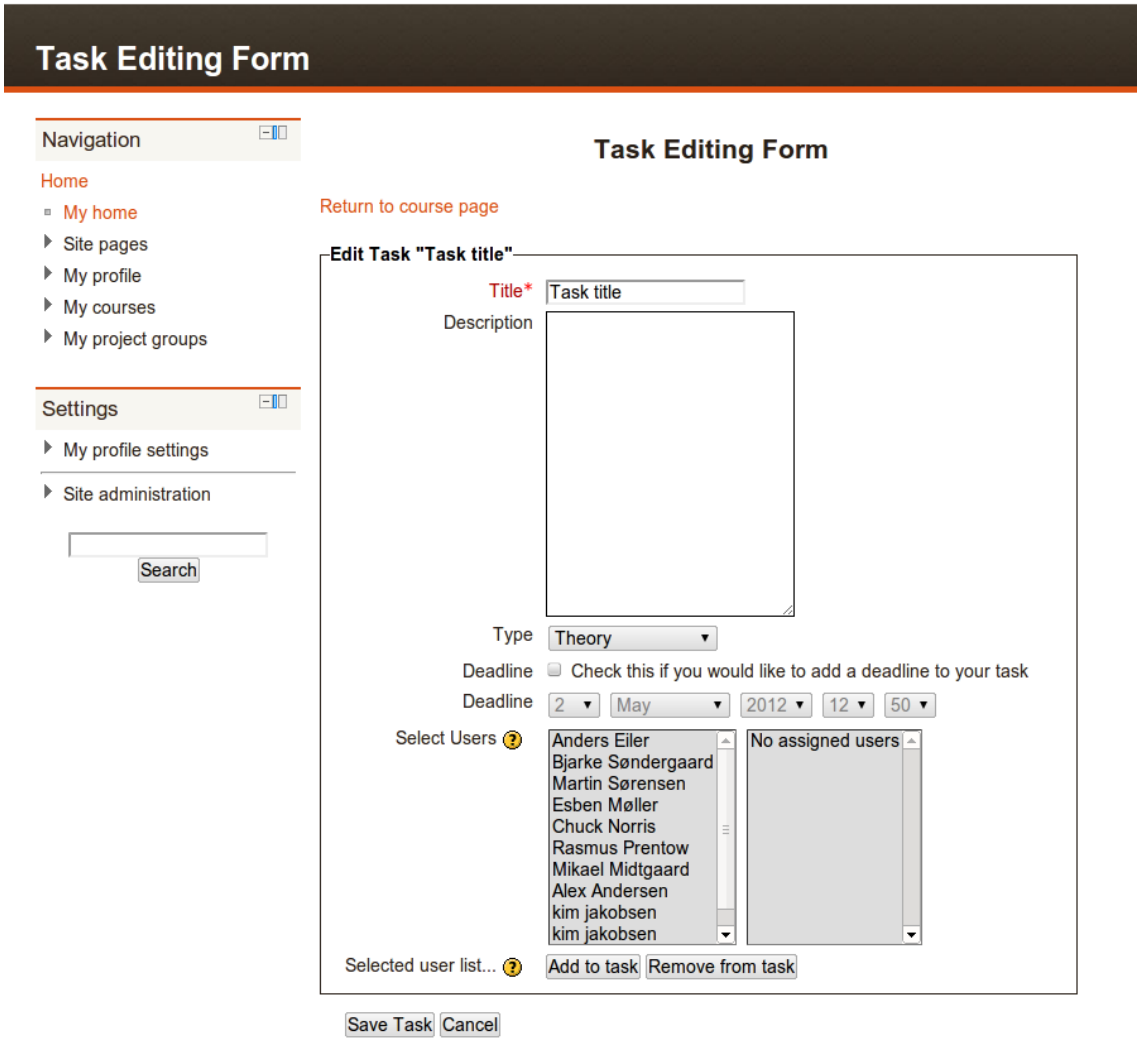

Figure 12.3. The task edit form.

entered in the form, and the modified task is passed as an argument to `edit()`, as defined in Section 9.2. Then `$SESSION->edit_task_id` is unset and the select form is displayed again, to allow the user to edit another task. The same thing happens when the cancel button is pressed, except the task is neither fetched nor edited.

Should the cancel button on the select form be pressed the user is redirected back to the course or project group page from whence they came. This is achieved by using the `$SESSION->tasks_context` variable, which is set whenever the tasks block is displayed. `$SESSION->tasks_context` is an array, the first value of which denotes whether the page is a course or project group page, while the second contains the id of the page.

The edit page can also be visited from the task list, a list showing every task relevant to a course or project group. When this is done the task to be edited has already been selected on the task list, so the select form should not be displayed. To achieve this a `$_GET` variable is set to the id of the selected task. When this `$_GET` variable is set another variable called `$SESSION->edit_task_id` is set as well, along with the variable `$SESSION->edit_origin`. `$SESSION->edit_origin` denotes that when the user is done editing or cancels she will be

returned to the task list. This is done by copying the save and cancel code to the top of the file, and only executing this when the `$SESSION->edit_origin` is set. In this version of the code instead of displaying the select form again the variables `$SESSION->edit_task_id` `$SESSION->edit_origin` are unset, and the user is redirected back to the task list.

12.3 Types Admin Tool

Management of predefined types is handled using the types admin tool. It consists of three files:

- `add_types_form.php` - contains the form for creating types. Will be referred to as add form.
- `delete_types_form.php` - contains the form for deleting predefined types. Will be referred to as delete form.
- `index.php` - generates the page where the add- and delete forms are displayed. Also contains the functionality that adds and deletes types from the database.

The add form is made as a Moodle form. The code for the add form can be seen in Code Snippet 12.2.

```
1  $mform = &$this -> _form;
2  $mform->addElement('header', 'add_types', get_string('add_types', '
       tool_types'));
3
4  $mform -> addElement('text', 'type_text', get_string('name', 'tool_types'))
       ;
5  $mform->addRule('type_text', get_string('missing_name', 'tool_types'), '
       required');
6
7  $mform -> addElement('select', 'colors', get_string('color', 'tool_types'),
        $this->get_colors());
8
9  $radioarray=array();
10 $radioarray[] = &MoodleQuickForm::createElement('radio', '
       courseprojectgroup', '', get_string('course_radio_button', 'tool_types'
       ), 1);
11 $radioarray[] = &MoodleQuickForm::createElement('radio', '
       courseprojectgroup', '', get_string('projectgroup_radio_button', '
       tool_types'), 2);
12 $mform->addGroup($radioarray, 'radioar', '', array(' '), false);
13 $mform->setDefault('courseprojectgroup', 1);
14 $mform->addElement('submit', 'add_button', get_string('add_button', '
       tool_types'));
15
16 function get_colors(){
17         return array("Red", "Blue", "Green", "Yellow", "Orange", "Grey", "
               Black", "Brown", "Magenta", "Purple");
18     }
```

Code snippet 12.2. The code for the add types form.

The form has a textfield, for entering the name of the type that is added on line 4. A rule that requires the field to be filled out is added on line 5. To assign colors to a type a select element is added. The elements for the list are returned by the function `get_colors()`

which returns a static array of colors. Radio buttons are added on line 8-13 for assigning a type to either a course or a project group. Lastly a button is added which must be pressed for the type to be created.

The delete form consists of a select element which displays the name of all the existing predefined types and allows the user to select them. Additionally it has a button identical to the one in the add form shown on line 14 in Code Snippet 12.2.

Existing types are gathered using two methods, which can be seen in Code Snippet 12.3.

The method `get_types()` sends an SQL query to the database and retrieves an array containing all predefined types. On line 12-18 this array is reformatted and returned. It is reformatted to get an object containing both the id and name of the type.

The method `get_types_string()` is used to generate an array of strings containing the names of the predefined types. This is used in the delete form to display the name of all existing predefined types. The function had to be created as the select element displaying the names of the types requires that the names are passed as an array of strings.

```
1  function get_types() {
2      global $DB, $SESSION;
3      if(!isset($SESSION -> tasks_context)){
4          return;
5      }
6      //Get the types associated with the course or project from the DB
7      $typedata = $DB -> get_records_sql("SELECT id, name
8                                          FROM {tml_task_types}
9                                          WHERE predefined = 1");
10     $counter = 0;
11     $types = array();
12     foreach ($typedata AS $k => $object) {
13         $type = new stdClass();
14         $type -> id = $k;
15         $type -> name = $object -> name;
16         $types[$counter] = $type;
17         $counter++;
18     }
19     return $types;
20 }
21
22 function get_types_string() {
23     $data = $this -> get_types();
24     $output = array();
25     $counter = 0;
26     foreach ($data AS $k => $object) {
27         $output[$counter] = $object -> name;
28         $counter++;
29     }
30
31     return $output;
32 }
```

Code snippet 12.3. Methods for gathering predefined types.

The `index.php` file contains the content that is displayed in Moodle. Additionally it contains the functionality that adds and deletes types from the database. A selection of its code can be seen in Code Snippet 12.4. Line 3-22 shows the code for adding a

predefined type. On line 4-8 the `$type` object is set up. The `$type` object is the one that is added to the database. `name` and `ref_type` is extracted from the Moodle form through the `$data` object. The color is only given as a string, so a `switch` is used to add the right HEX Code to the `$type` object. On line 21 the object is inserted into the database. On line 22 the `$data` object is unset to clear the form.

Types are deleted using the code on line 25-32. First it is checked if any types have been selected, if this is the case it is further checked that the type still exists in the database. There is a small chance that the type has been deleted in the meantime. If it exists it is deleted.

```
1  $mform = new add_types_form();
2  $colors = $mform -> get_colors();
3  if ($data = $mform -> get_data()) {
4      $type = new stdClass();
5      $type -> name = $data -> type_text;
6      $type -> ref_id = -1;
7      $type -> ref_type = $data -> courseprojectgroup;
8      $type -> predefined = 1;
9      $color = $colors[$data -> colors];
10     switch ($color) {
11         case "Red" :
12             $type -> color = "#FF0000";
13             break;
14                 ...
15
16                 ...
17         case "Purple" :
18             $type -> color = "#551A8B";
19             break;
20     }
21     $type = $DB -> insert_record('tml_task_types', $type, TRUE);
22     unset($data);
23 }
24
25 $mform = new delete_types_form();
26 $types = $mform -> get_types();
27 if ($data = $mform -> get_data()) {
28     if (isset($data -> delete_types_list)) {
29         foreach ($data->delete_types_list AS $k => $value) {
30             $exists = $DB -> record_exists('tml_task_types', array('id' =>
                   $types[$value] -> id));
31             if ($exists) {
32                 $DB -> delete_records_select("tml_task_types", "id = " .
                       $types[$value] -> id);
33             }
34         }
35     }
36 }
```

Code snippet 12.4. Code for adding and deleting types.

My Moodle 13

This chapter describes how the plugins described in the two previous chapters were implemented in MyMoodle, and hereby integrated with the other sub-systems.

13.1 PGLib

The shared functionality and the components used across the sub-systems are stored in PGLib located in `/local/projectgroup/`. The actual project group concept was developed and implemented by another group, so the details of this sub-system will not be covered in this report. However, some of its functionality is important to the parts that were described in the two previous chapters.

The `lib.php` is the standard library file in the PGLib. Each sub-system has its own lib file in PGLib that contains the shared functionality from each sub-system. The functionality shared among the sub-systems is implemented in the `/local/projectgroup/lib_{name}.php` file. Where `{name}` refers to the individual names of the plugins that make up the sub-systems. All `lib_{name}.php` files are included in the standard `lib.php` file in the Project Group plugin.

Code snippet 13.1 shows a minimal version of the `fetch_tasks_data()` function. This function takes a filter as an argument and fetches the appropriate tasks in the database. It proceeds to format them into an array and returns it. By looping through the filters on line 8, the SQL query is constructed from pieces. On line 37 the SQL query is executed, with the variables `$join` and `$where`, which contain join statements and where clauses respectively.

13.2 System Integration

Both the timeline and tasks are implemented as blocks in Moodle that can be used in several different contexts depending on the user's wishes. The overall purpose of this project, however, was to create a system that integrates the Aalborg PBL model in Moodle.

Figure 13.1 shows the final product where all sub-systems of MyMoodle have been integrated into Moodle. This report will not cover the details of the project group home page as it is not developed by us, but since it is a part of the final solution, it is hereby shown. The purpose of this page is to collect everything with relevance to PBL in one

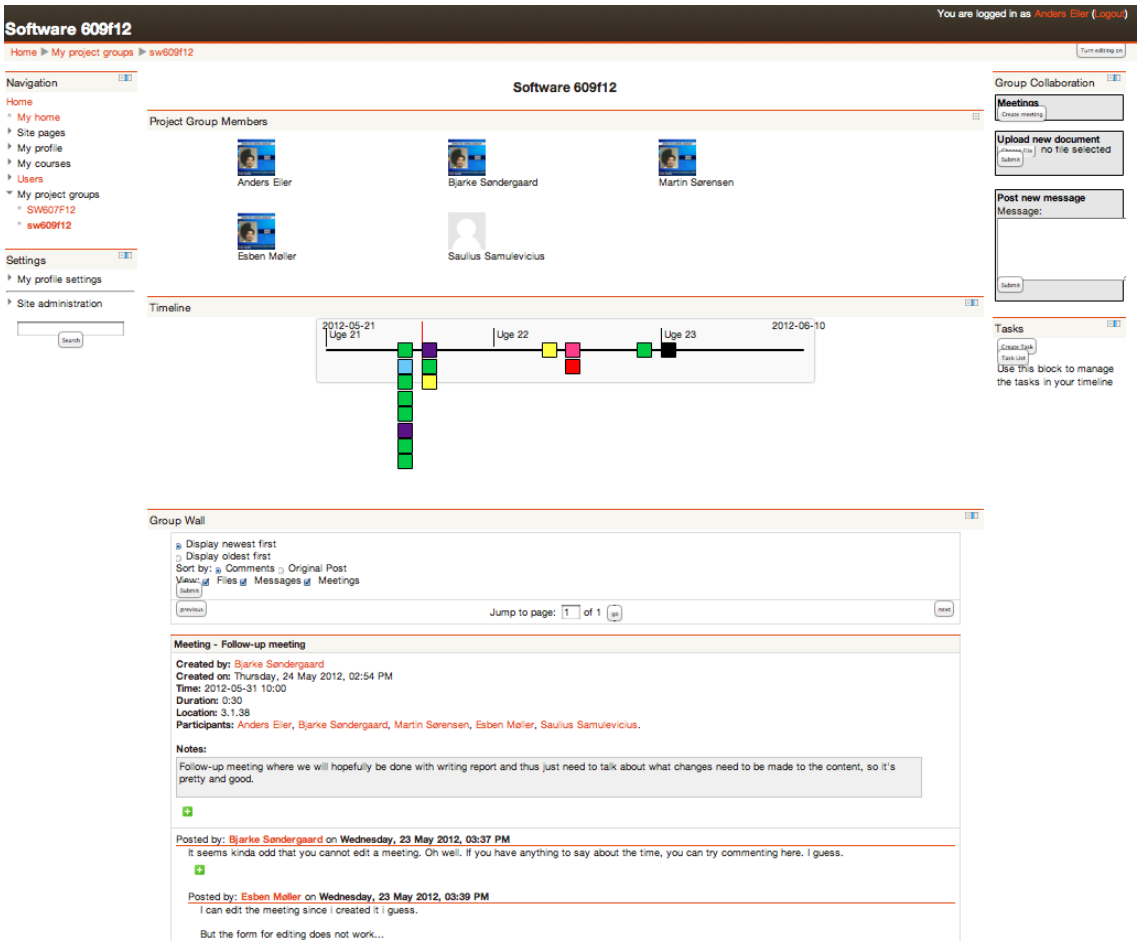

Figure 13.1. MyMoodle

place – the timeline, tasks, blackboards, communication, supervisor communication and interaction with the administrational personal.

```php
function fetch_tasks_data($filter=array()) {
    global $DB, $CFG, $USER;

    $where = "WHERE 1=1 ";
    $join = "";
    $group_or_course = false;
    $group_flag = "";
    if(count($filter) > 0) {
        foreach($filter AS $k=>$v) {
            switch($v['ident']) {
                case 'DATE':
                    if($v['type'] == '0') {
                        $values = str_replace(' ', '', $v['args']);
                        $values = explode(",", $values);
                        $where .= "AND t.timestamp > '" . strtotime($values
                            [0]) . "' AND t.timestamp < '" . strtotime(
                            $values[1]) . "' ";
                    }
                    elseif($v['type'] == '1') {
                        $where .= "AND DATE(FROM_UNIXTIME(t.timestamp)) = '
                            " . $v['args'] . "' ";
                    }
                    break;


                // ... more filters here
            }
        }
    }

    if ($group_or_course) {
        $group_flag = "";
    }
    else {
        $join .= " JOIN {tml_task_users} AS u ON u.task_id=t.id ";
        $group_flag = " AND u.user_id='".$USER->id."'";
    }

    $d = array();
    $t = $DB->get_records_sql(" SELECT t.id, t.deadline, t.status, t.title,
         t.description, t.type, ty.id as type_id, a.ref_id, a.ref_type, ty.
        name, ty.color
                                FROM {tml_tasks} AS t JOIN {tml_task_types}
                                     AS ty ON t.type=ty.id
                                JOIN {activities} AS a ON a.id=t.
                                    activity_id $join $where AND t.deadline
                                     IS NOT NULL $group_flag
                                ORDER BY t.deadline");

    if (is_array($t)) {
        foreach($t AS $k=>$v) {
            // ... prepare data to be returned here

            $d[$k] = array( ... );
        }
    }
    return $d;
}
```

Code snippet 13.1. fetch_tasks_data function from the tasks_lib.php.

Part IV

Testing

Testing Approach 14

In this chapter dynamic and static black- and white-box testing will be described. Based on this description a testing approach will be chosen for the project and tests conducted to ensure that the implemented software works. This chapter is based on [51].

14.1 Black-box Testing

Black-box testing is a term used to describe testing where the system is viewed as a black box, i.e. the inner workings of the system is unknown. The static form of black-box testing involves examining the specification, whereas the dynamic form involves giving an input and verifying that it matches the output specified in the specification.

14.1.1 Static Black-box Testing

Static black-box testing is concerned with examining the specification. This can be done in two steps. Firstly a high-level review is conducted, where the overall concept and content of the specification is examined. Secondly a low-level specification is conducted that examines what is written, e.g. is the specification unambiguous or does it contain unspecific terms and words.

A high-level examination may involve many facets. It is important to ensure the system described in the specification matches the expectations of the end-user. Similarly it must be checked that the system conforms to any standard it meant to follow. As an example, systems delivered to the military must follow certain standards, and this should be reflected in the specification. Lastly, existing similar systems should be examined to get an idea of the functionality the system should contain and ensure this is reflected in the specification.

Low-level examination involves examining the specification according to eight attributes:

- **Complete** - Is the specification thorough or does it lack anything e.g. the customers' desires or a standard specification it should contain.
- **Accurate** - Is the solution that the specification presents correct or does it contain any errors.
- **Precise, Unambiguous, and Clear** - Is the specification exact and without ambiguity, and is it easy to read and understand.

- **Consistent** - Is the specification of a feature written so it does not conflict with itself or other features.
- **Relevant** - Is there any unnecessary information in the specification.
- **Feasible** - Can the defined system be implemented within the budget and time frame with the available personnel, tools, and resources.
- **Code-free** - Does the specification describe details concerning actual code.
- **Testable** - Can the system be tested, and if yes does the specification contain enough information to write proper tests.

Lastly it should be tested that the specification does not contain unspecific words such as *good*, *etc.*, *fast*, etc.

14.1.2 Dynamic Black-box Testing

Dynamic black-box testing is defined as software testing with no knowledge of the inner workings of the software. To perform dynamic black-box testing a specification of the system is required. Based on this specification it should be possible to deduct what the result would be of a given action.

Using the specification concrete test cases are created for the software. Specific test cases for a function that increments an integer could be: `input 1 should yield output 2`, and `input 1.3 should yield an error message`. These two test cases serve to demonstrate two important aspects of writing test cases, that is test-to-pass and test-to-fail.

Test-to-pass refers to test cases written to pass, e.g. the first test case in the previous paragraph. If any test that is written to pass fails, it means that the software does not live up to its specification.

Test-to-fail refers to test cases which are written to fail. In the example above a function that has an integer as parameter is given a floating-point number instead, which should result in an error. This demonstrates the purpose of test-to-fail, namely to try and break the software by deliberately given it incorrect input, and hereby discover hidden bugs that test-to-pass would not have found.

Since it is not feasible to write a specific test case for every possible input, test cases should be written with equivalence partitioning in mind. Equivalence partitioning is the reduction of all possible test cases to a set of cases that do not necessarily yield the same result, but are so similar that performing one of the test cases gives the same assurance as performing all of the test cases within the set. For the example of the increment function all positive integers might be equivalent, and thus testing any one positive integer would do. Other partitions for this function could be zero and all negative integers. In this case they test how the function handles an input of the correct data type which is within the boundaries of that data type.

A proper set of test cases should not contain equivalent test cases.

14.2 White-box Testing

Unlike black-box testing the code is available during white-box testing. The static element is formal reviews ranging from peer-reviews to code inspections. The dynamic element is concerned with writing test cases based on the concrete code. Similarly it is possible to measure how much a system has been tested, e.g. with code coverage.

14.2.1 Static White-box Testing

Static white-box testing refers to the process of reviewing the design, architecture, or code. The purpose is to discover bugs without executing the code. Static white-box testing is done through formal reviews. The term, formal reviews, is a loose term ranging from two people going through code to an entire team examining the design in detail.

One of the least formal methods of white-box testing is peer-review. During peer-review two or more programmers exchange code and look through each others work for bugs or deviations from the specification. An expanded method of peer-review is walkthroughs. During walkthroughs the programmer presents his code to a group of programmers and testers. A walkthrough is more thorough as more people are reviewing the code and they are given the code in advance so they can prepare questions and notes. The trade-off is, however, that it takes more time.

The most formal type of white-box testing is inspections. Due to the formality of inspections, they require some form of training for all participants. In inspections the code is given to a group of people, which does not include the programmer. Every member of the group is given a specific role while reviewing the code. One is given the role of the programmer, and has to present the code to the other group members. The others are given roles such as tester, user, etc. An inspection meeting is then held similar to how a meeting is held in walkthroughs.

Static white-box testing aims to discover conceptual flaws in the code as well as concrete errors such as:

- **Reference errors** - Are all referenced variables instantiated, of the correct type, etc.
- **Declaration errors** - Are there duplicate variable declarations, are there unused variables, etc.
- **Computation errors** - Are there errors in calculations, e.g. formulas that result in an incorrect result.
- **Comparison errors** - Are the comparison operators used correctly, are the boolean expressions correct, etc.
- **Control flow errors** - Is there a possibility for infinite loops, are all control structures exited at the correct time, etc.
- **Parameter errors** - Are the given parameters of correct format and/or type, if constants are passed as parameter to a function are they changed in its body, etc.

14.2.2 Dynamic White-box Testing

Dynamic white-box testing refers to dynamic testing where the tester has access to the code, and using this information determines what to test and how to do it. It can be divided into the following categories:

- Testing up against an API. This means to test low-level functions, libraries, etc.
- Testing at the top level. This involves testing the complete program, but using the knowledge of the underlying code to create and adjust test cases.
- Gaining access to variables and the programs current state to determine if a test is working as intended. This also allows the tester to force the software to do something normal testing would not allow for.
- Measuring which parts of the code actually gets executed during testing, and using this information to remove redundant test cases, and add more to execute previously inactive code.

The access to the code in dynamic white-box testing creates new testing criteria as well. With access to the code it is possible to measure how much of the code has been tested. This can be measured on three levels.

- **Code coverage** - How many lines of the program have been executed during the tests.
- **Branch coverage** - How many of the different branches (paths through the code) in the program have been executed during the tests.
- **Condition coverage** - How extensively have the conditions in the program been tested. A condition might contain an and clause giving it two conditions which must be tested.

It should be noted however that obtaining complete code, branch, or condition coverage is often not realistic, so it should be decided what an acceptable amount of coverage is.

14.3 Testing Approach

The aspects to consider when choosing a testing approach is primarily the amount of resources available. Ideally a system should be tested with a branch- and condition coverage of 100%, however as that would often require an unreasonable amount of test cases to be written it is not feasible. As MyMoodle is divided into four sub-systems, which are further divided into sub-components, the main approaches will be unit testing and dynamic black-box testing. Unit testing is testing specific units of code, such as functions. Unit testing will be performed on every developed component to reduce the chance that they carry bugs that might propagate into the final system. The dynamic black-box testing will be performed on the completed components, to ensure they work correctly when given input through the GUI instead of through test cases. The unit testing will be done via SimpleTest which was described in Section 5.2, and the dynamic black-box testing will be done through the user interfaces. We will further perform integration and end-user testing.

Test Plan 15

In this chapter the test plan will be presented along with sample test cases covering both unit-testing and dynamic black-box testing. Furthermore, the tools used for testing will be presented.

15.1 Test Phases

The testing is divided into three phases. Unit testing will be performed continuously through the implementation phase. At the end of the implementation additional test cases will be written to ensure a sufficient amount of the code has been tested. After unit testing is completed, dynamic black-box testing will be performed to ensure data is transferred correctly from the GUI to the underlying functions. Lastly integration testing will be performed on MyMoodle.

15.1.1 Unit Testing Phase

The unit testing phase is the initial test phase. It will run from the beginning of the implementation until it is completed, and is to be interpreted as a formal variation of the code-and-fix model. The traditional code-and-fix model is a simple model where the programmers find and fix bugs as they program, and continue this until the program works as intended. In this phase code-and-fix is employed in its traditional manner, but it is additionally a requirement that unit tests are written for all public functions. The intention is that code cannot be considered complete when it seems to function as intended. Instead all code must be exposed to a form of formal testing. The phase is implemented in the development process by adding unit testing as an item to every sprint backlog. All group members are required to write unit tests. The phase will be considered complete when the implementation is finished and there exist at least one test case for every public function where it is feasible.

SimpleTest

The unit testing will be conducted using the SimpleTest framework described in Section 5.2. SimpleTest is built into Moodle and tests can be run through Moodle's GUI. It works by looking through all folders named `/simpletest/` for PHP files with the prefix `test_`. The files containing the tests should follow the template shown in Code snippet 15.1 [18].

```php
<?php

if (!defined('MOODLE_INTERNAL')) {
    die('Direct access to this script is forbidden.'); //  It must be
        included from a Moodle page
}

// Make sure the code being tested is accessible.
require_once($CFG->dirroot . '/mod/quiz/editlib.php'); // Include the code
    to test

/** This class contains the test cases for the functions in editlib.php. */
class some_plugin_test extends UnitTestCase {
    function test_something() {
        // Do the test here.
    }

    // ... more test methods.
}
?>
```

Code snippet 15.1. Test class template for SimpleTest in Moodle.

They should contain a class which extends `UnitTestCase`, along with a method for each test case.

When performing tests on functions that read and write from the database a so-called Mock Database *(Mock DB)* should be used. A Mock DB is a database that imitates a real database. This is to keep the real database free of test data, and to keep it safe in case of bugs. A Mock DB has to be set up to expect certain operations to be performed on the database. An example can be seen in Code snippet 15.2. On line 3 a Mock DB uses the function call `expectAt()` to set up an expectation for a call to the function `insert_record()` with the given array as argument. The `0` indicates that it is the first time `insert_record()` is used on the database. An expect call to a Mock DB will return an error if the function is not called as expected. There exist a variety of expects such as: `expectNever()`, `expectOnce()` and more which can be used to create more specific test cases. The function `setReturnValueAt()` is used to set what database functions return, as, per default, all functions on the Mock DB return null. The code on line 11 means that on the first call to `insert_record()` it will return 1.

Tests are run through a user interface in Moodle. It can be limited to only execute the tests associated with a particular plugin. An example of a set of test cases being run can be seen in Figure 15.1

These functions will be used to conduct unit tests on the system.

```
1   ...
2
3     $DB->expectAt(0, 'insert_record', array('tml_tasks', (object) array(
4         'timestamp'     => time(),
5         'title'         => $title,
6         'description'   => $desc,
7         'type'          => $type,
8         'activity_id'   => 1,
9         'created_by'    => 1,
10        'status'        => 1), true));
11    $DB->setReturnValueAt(0, 'insert_record', 1);
12    $DB->setReturnValueAt(1, 'insert_record', 2);
13
14  ...
```

Code snippet 15.2. Mock DB example.

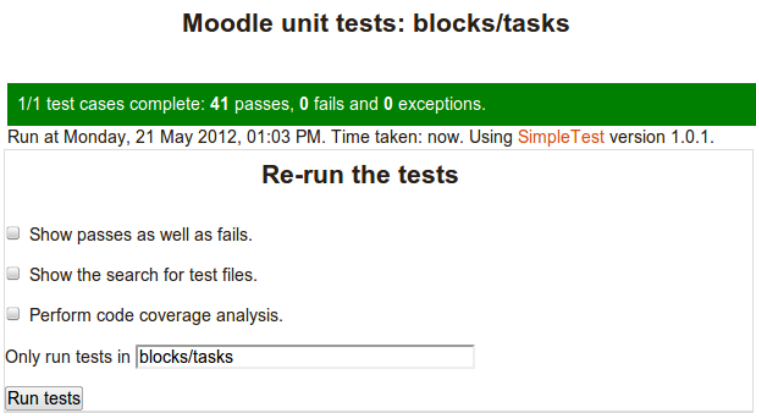

Figure 15.1. Unit tests being run through Moodle's interface.

15.1.2 Dynamic Black-Box testing phase

The dynamic black-box testing will be conducted through the created GUI for the different components. The aim is to uncover any bugs that might originate from the input being given through a GUI. Potential issues are: the input being wrongly formatted or the function extracting the data from the GUI returning it incorrectly.

Test cases for dynamic black-box testing follow the same format as those for unit testing. The difference is that the test cases are evaluated by the actions taken by the system. An example is when input of the wrong format is given, the system should inform the user of this. Therefore every test case should contain a pass/fail criteria specifying when a test is considered successful and unsuccessful.

A sample test case can be seen in Figure 15.2. The test case is for the `create_task` form, and tests the various correct and incorrect input that can be entered. The other test cases can be seen in Appendix B.

ID: Test case #1 Create task. Creating a task consists of a number of fields being filled out. This test should make sure that no rogue values are entered into any of the fields, and that the software handles the entered informations correctly in order to create the wanted task.
The input consists of:

- Title (string)
- Description (string)
- Deadline (date-time)

The output of the function will consist of either an error message or a success message, that the task has either been saved correctly or that an error occurred during processing of the entered data. If it was saved correctly we also expect the new task to show up on the list of tasks.
In order to create a new task, only the creator, data about the new task, and a PC is needed.
Input specification

1. Title: Null - Description: Null - Deadline: Null.
2. Title: 256 characters - Description: 256 characters - Deadline: date-time.
3. Title: Not UTF-8 - Description: Not UTF-8 - Deadline: date-time.
4. Title: UTF-8 content - Description: UTF-8 content - Deadline: Not date-time format.
5. Title: UTF-8 content - Description: UTF-8 content - Deadline: date-time format.

Output specification

1. If any of the input fields is null, reject.
2. If the title or description is 256 characters, reject.
3. If the title or description is not UTF-8, reject.
4. If the deadline is not a date-time format, reject.
5. If the title and description are both UTF-8 content and the deadline is of date-time format, accept.

Environmental needs A single tester working with the system on a PC. The system is installed on a server.
Intercase dependencies This case has no dependencies.

Figure 15.2. Test Case #1.

Code Coverage

The goal for the tests are that all public functions are run at least once, and that at least 55% of lines of the code are executed. This guarantees that all functions which can get input from the user have been tested, and that over half of the code has been run during tests. The satisfying amount of code coverage can vary depending on the nature of the system. Given the time constraints of the project, a code coverage of 55% was chosen.

15.1.3 Integration Testing Phase

At the completion of MyMoodle, integration testing will be conducted. The integration testing will consist of two separate tests: one internal and one with potential end-users. Internal integration testing will be conducted in a workshop manner with at least one participant from every group present with the purpose of finding bugs. Both the setup and use of MyMoodle will be tested, and it will be done in a dynamic black-box manner. The setup consists of a project group being created from scratch and setup to suit the needs of a fictional project group. Following the setup the different plugins will be tested from the perspective of a user, as well as testing that aims to break the system by performing unintended actions. Unintended actions might be editing a task and then deleting it before saving the changes. The result of the internal integration test can be seen in Section 16.3.

The test with potential end-users will aim to discover any conceptual errors in MyMoodle. Similarly possible usability errors should be uncovered by having external people using the system. The end-users involved in the test are students and administrative staff. The results of this test can be seen in Section 16.4.

Test Report 16

In this chapter the results of the performed tests will be presented. As the unit tests were conducted continuously throughout the implementation and the bugs fixed as they were discovered the results from these tests will not be presented. The code coverage obtained by the unit tests will be presented, however, to give an evaluation of how well the system has been tested. The results presented are those from the dynamic black-box tests, integration, and the end-user test.

16.1 Code Coverage Results

Dynamic white-box testing was conducted on the functions in the task and timeline blocks. The code coverage report from the initial test on the task functions revealed that one function was not covered at all. This was a function not initially planned and thus added halfway through the development process, without corresponding tests being written. After writing tests for this function the code coverage for the tasks library file was *84.45%*.

Before presenting the code coverage for the timeline tests it should be noted that most of the timeline functions were related to printing the actual timeline. Since it is impossible to test what code prints with SimpleTest there were no tests written for these functions. Functions were tested where possible, which resulted in a code coverage of *4.72%*.

A total of 75 unit tests were written, which all passed.

16.2 Dynamic Black-box Test Results

The dynamic black-box tests conducted can be seen in Appendix B. In this section only the test cases which uncovered bugs or showed interesting results will be presented.

The first test case was concerned with testing the form handling the creation of tasks. The results can be seen in Table 16.1.

The first input in Test Case #1 revealed an issue. When a task is created without the required field title, the user is redirected to the page she came from, but not informed that a task has not been created. This is not directly classifiable as a bug, since no task without a title is created. It is however still an issue as it might confuse users, and without

knowledge of the issue they might wrongly assume the system is unable to create tasks. This bug was fixed so the user is not redirected unless a task is successfully created.

The second input in Test Case #1 revealed a bug. No checks are done on the length of the task's title, and since the title field in the database is of the type `varchar`, there is a limit on the length of strings it can contain. This is not checked before an insert query is made to the database, and consequently it is possible to crash the system with a title longer than 255 characters. This bug was fixed by adding a rule to the form that required the title length to be less than 255 characters, as 255 is the maximum input size for that field in the database.

The other inputs in Test Case #1 did not reveal any issues. It should be noted though that due to the date-time selector being used it is not possible to give a wrongly formatted date as input. The date-time selector is part of Moodle forms.

Index	Input Specification	Result	Status
1	**Title:** Null **Description:** Null **Deadline:** Null.	No task created - redirects to project page. No error on empty title	Fixed so the user is no longer redirected. User is presented with "Title is required" instead
2	**Title:** Too long **Description:** Too long **Deadline:** Date-time	Title too long results in a database error	Fixed so the user is presented with "Title length must be shorter than 255 characters" instead of database issues
3	**Title:** Not UTF-8 **Description:** Not UTF-8 **Deadline:** Date-time	Task created without issues	No fix needed
4	**Title:** UTF-8 content **Description:** UTF-8 content **Deadline:** non date-time format	Irrelevant due to a date-time selector being used to set deadline	No fix needed
5	**Title:** UTF-8 content **Description:** utf 8 content **Deadline:** date-time format.	Task created without issues	No fix needed

Figure 16.1. Test results for Test Case #1.

Test Case #2 was concerned with assigning users to a task. The first input did not reveal any issues, as the page reloaded without changing any data. The second input yielded the same result, i.e. no actions were taken except reloading the page. The last input successfully assigned users to the task. This test case reveals that the function which handles assignment of users to a task properly checks that the input given is valid before performing any operations. Consequently no issues or bugs were uncovered in this test.

Index	Input Specification	Result
1	**Assignment:** Null **User:** Null	Performs no actions
2	**Assignment:** Invalid ID **User:** Invalid ID	Performs no actions
3	**Assignment:** Valid ID **User:** Valid ID	Success - a user is added to the task

Figure 16.2. Test results for Test Case #2.

The other tests had similar results to those found in Test Case #2 and can be seen in Appendix C. The only bugs or issues which caused the system to cease functioning were the issues found in Test Case #1.

Overall the system conformed with the expectations specified in Appendix B. There are however some cosmetic problems which will be explained in Section 16.4.

16.3 Integration Test

At the end of the implementation phase an integration test was planned. The setup was described in Section 15.1.3. This section will cover the internal testing, while the external with potential end-users is covered in Section 16.4.

In this section the focus will be on bugs and issues uncovered that relates to the timeline and tasks. The full results of the internal test can be found in Appendix D. It should be noted that the list below contains notes and thus context will be vague.

1. Timeline - The objects become weirdly formatted when they are large. Their spacing is probably relative to the size, or it does it incorrectly compared to vertical/horizontal.
2. Timeline - Create links for meetings to an anchor point, so the Timeline links down to the group wall instead of the calendar.
3. Timeline - Have a look at the boxes that pop up when the mouse is hovered over an object at the bottom of the page.
4. Timeline - When there's a link to the edit page you can arrive from both project page and course page.
5. Timeline - Delete task page does not render.
6. Timeline - The task list page says there are no tasks.
7. Timeline - A year in the future on the timeline causes the box to go outside of the timeline.
8. Timeline - When the year changes you start in week 0.
9. Timeline - Edit task sets the deadline to today.
10. Timeline - Hide complete task on completed tasks on the task list. It should be replaced by a minus and an uncompleted text.
11. Timeline - Consider the same form for edit and create tasks.
12. Timeline - Find the difference between course and project group.

All the bugs and issues on the list were fixed before receiving user feedback.

16.4 User Feedback

The people who were interviewed prior to the development of this project were also interviewed after the implementation was done. These interviews were conducted to get feedback from the same users that told us their expectations, in order to see if they were met. These interviews were meant to cover the test with potential end-users as explained in Section 15.1.3. The interviews were conducted as informal interviews where one person introduced the interviewee to the system and gave a brief walkthrough. The purpose of these interviews was to find out whether or not the system met requirements of potential end-users.

16.4.1 Presented Setup

A setup was prepared for these interviews. It consisted of MyMoodle, where a project group was created. This project group was set up to look as any project group could look like in production. It contained six members, a supervisor, a timeline, some meetings and tasks, and two blackboards.

This setup was used to gain a realistic scenario.

16.4.2 Interview Results

The interviews all covered the same topics. The results of these and the users' feedback will be ordered by topic, instead of by user. Only the parts of the interview that are relevant to this project will be presented here. The interview notes can be found in Appendix D.

Tasks

The general idea of a todo list was well received with the test users. They liked the idea of having a place to organize all the tasks. There were, however, a couple of things they would like to have changed. Most of them were related to usability and not the functionality:

- The calendar does not close when a date is selected.
- The default title does not disappear when the field is selected.
- A color should be uniquely attached to a task-type.
- A task should be able to contain sub-tasks (a tree-like structure).

Timeline

The timeline was also welcomed and an appreciated feature. They were happy with the overview it gave of the current period and the work or tasks that had to be done. Also here, a few notes were made:

- It would be nice if it was possible to show scheduled lectures on the timeline.
- If many months are shown in a horizontal timeline, it may get a bit chaotic and easy to lose the overview.

General Impressions

The general opinion of all the information being collected on one page in Moodle, i.e. MyMoodle, was positive. Both test users meant that some work could be done with the looks of the system, e.g. the Group Wall could be optimized to use space efficiently while also looking better. These comments were, however, of cosmetic character.

Part V

Epilogue

17 Discussion

This project differed from previous projects: in previous projects we were given an assignment or a topic, and based on this we would formulate a problem statement to work on within our project group. This semester, however, we worked as part of a multi-project where four groups of three to four people collaborated on solving a problem. Consequently a common system analysis and product were constructed.

This meant that we had a lot of new challenges compared to previous semesters. A lot of effort was put into deciding the development method that accommodated the work form of this semester the best. We chose the agile development method Scrum, more specifically the modified version Scrum of Scrums. The approach proved successful in organizing the multi-project and handling the issues that inevitably occurred.

While the environment in which the project was conducted worked great and enabled us to solve the co-operational challenges without major issues, the implementation of the system proved to be more challenging. The ideas we had and wanted to implement – project group pages, timeline, blackboard, supervisor contact, etc. – covered simple yet important aspects of problem based work, but proved troublesome to implement. Our lack of knowledge concerning Moodle's inner workings and how to extend it proved to be the greatest challenge. It was an important lesson to experience that something with a simple concept could be challenging to implement in an existing and unknown system. A lot of time was spent trying to understand how Moodle works and how plugins should be constructed. While it is a open source system, the documentation is of varying quality. This was a problem, especially in the beginning of the project, when we were still new to the system. Our lack of knowledge concerning Moodle caused additional problems during our planning poker sessions, since estimating tasks proved difficult. It should, however, be noted that our estimations improved as the project progressed, since we gained more knowledge of Moodle.

Our sub-system was concerned with managing tasks and project planning. Both of these were implemented as a type of Moodle plugins known as blocks. The tasks block was implemented as a todo list, while the timeline block became a view of the tasks as well as activities from the other sub-systems of MyMoodle. We had to change our approach towards the implementation of our plugins twice, as our knowledge of Moodle increased and we discovered more efficient ways to implement the same functionality.

A number of tests were conducted to ensure that the systems worked and did not have any errors. For this purpose both dynamic black- and white-box testing were used combined with unit testing and interviews with different test persons. These tests, based on our test plan, were conducted repeatedly during the testing phase to get rid of any bugs that might exist within the system.

Any errors that are still present in the system are of cosmetically character.

Conclusion 18

The problem definition was:

Moodle does not fully support the work method used at AAU. Moodle is built up strictly around courses, and does not contain the concept of project groups. To accommodate for this students must use different tools for project group work. This project deals with how support for the Aalborg PBL model can be implemented in Moodle. This involves researching other systems than Moodle for information on how they accommodate project groups, and conducting interviews with students and administrative personnel of different faculties to gather requirements for such an extension to Moodle.

The problem definition consists of three parts:

1. Research other systems similar to Moodle for how they accommodate project groups.
2. Conduct interviews with both members of the faculty and students to gather requirements for such an extension to Moodle.
3. Design and implement extensions for Moodle that enables it to handle the Aalborg PBL model.

A number of LMSs were researched in order to determine whether or not they could support the Aalborg PBL model. None of these supported the model natively. While none of them offered what we needed, they provided us with inspiration on how we could implement the system ourselves.

Each project group conducted a number of interviews with different people who were relevant to their sub-system. We conducted interviews with three different students before designing and after implementing. The purpose of the first interview was to get their requirements for such a system. The last interview was conducted to see if the results from the implementation phase met their expectations. The potential end-users found MyMoodle to be a potential solution to their problems with project management.

The design of the overall system was done in co-operation with three other project groups. PGLib contains all the functionality that is shared amongst the different sub-systems. Each sub-system's implementation structure was designed in accordance with the rest of MyMoodle.

MyMoodle works as a whole, and the four sub-systems come together to create functionality for Moodle that supports the Aalborg PBL model.

Future Work 19

The future work for the project is based on the last interviews and the scope of the project.

19.1 From Proof of Concept to Production

MyMoodle is, in its current state, not meant for production. It is a proof of concept showing the possibility of implementing the Aalborg PBL model in Moodle. Therefore a lot of future work lies in transforming the current proof of concept into a system that can be used in production. Amongst other things, this includes expanding functionality, redesigning the GUI to make it more user friendly and performing usability tests.

19.2 Implementation of Suggested Changes

Interviews have been conducted after the implementation was completed in order to get the interviewees' feedback on what could be improved. They found the concept and functionality good. However, several of them felt that we needed to put more effort into the styling and visual presentation of the product. Therefore some of the future work can be put into making the system more user friendly based on usability tests.

19.3 Expanding Functionality

The four project groups have implemented tools to integrate the Aalborg PBL Model in Moodle. Based on the interviews, more can be added in the future. This could be functionality such as course integration with the timeline, integration with Google Docs, version control systems, etc. to make MyMoodle a complete package containing all aspects of the project related work.

Bibliography

[1] Adobe connect. URL http://www.adobe.com/products/adobeconnect.html. Last viewed: 22/5-2012.

[2] Definition af e-lÃ¦ring, . URL http://www.itst.dk/ferdigheder/e-lering/om-e-lering/definition-af-e-lering/definition-af-e-lering. Last viewed: 18/4-2012.

[3] Definition of e-learning, . URL http://www.webopedia.com/term/e/e_learning.html. Last viewed: 2/5-2012.

[4] Dropbox. URL https://www.dropbox.com/. Last viewed: 22/5-2012.

[5] E-lÃ¦ringssamarbedet ved aalborg universitet. URL http://www.elsa.aau.dk/. Last viewed: 19/4-2012.

[6] Git. URL http://git-scm.com/. Last viewed: 2012-05-31.

[7] Google docs. URL https://docs.google.com/. Last viewed: 22/5-2012.

[8] Gnu general public license. URL http://www.gnu.org/copyleft/gpl.html. Last viewed: 19/3-2012.

[9] Mercurial. URL http://mercurial.selenic.com/. Last viewed: 2012-05-31.

[10] jquery. URL http://www.jquery.com/. Last viewed: 1/6-2012.

[11] Litmos. URL http://www.litmos.com. Last viewed: 9/5-2012.

[12] Mahara. URL https://mahara.org. Last viewed: 9/5-2012.

[13] Moodle, . URL http://moodle.org/. Last viewed: 19/3-2012.

[14] Unit test api, . URL http://docs.moodle.org/dev/Unit_test_API. Last viewed: 7/3-2012.

[15] Master in problem based learning in engineering and science. URL http://www.mpbl.aau.dk/. Last viewed: 19/4-2012.

[16] Sharepointlms. URL http://www.sharepointlms.com. Last viewed: 9/5-2012.

[17] Simpletest: Unit testing for php, . URL http://www.simpletest.org/. Last viewed: 7/3-2012.

[18] Unit test api, . URL http://docs.moodle.org/dev/Unit_test_API. Last viewed: 2/5-2012.

[19] Skype. URL http://www.skype.com/. Last viewed: 22/5-2012.

[20] Apache subversion. URL http://subversion.apache.org/. Last viewed: 6/3-2012.

[21] Scrum of scrums, 2007. URL http://www.scrumalliance.org/articles/46-advice-on-conducting-the-scrum-of-scrums-meeting/. Last viewed: 2/5-2012.

[22] Analysis of git and mercurial, 2008. URL http://code.google.com/p/support/wiki/DVCSAnalysis. Last viewed: 14/3-2012.

[23] Studieordningen for bacheloruddannelsen i software, 2009. URL http://www.sict.aau.dk/digitalAssets/3/3331_softwbach_sept2009.pdf. Last viewed: 22/5-2012.

[24] Eventum issue / bug tracking system, 2012. URL http://dev.mysql.com/downloads/other/eventum/. Last viewed: 13/3-2012.

[25] Eventum issue / bug tracking system, 2012. URL http://dev.mysql.com/downloads/other/eventum/features.html. Last viewed: 12/3-2012.

[26] Moodle - blocks tutorial, 2012. URL http://docs.moodle.org/dev/Blocks. Last viewed: 10/5-2012.

[27] Data manipulation api, 2012. URL http://docs.moodle.org/dev/Data_manipulation_API. Last viewed: 28/5-2012.

[28] Moodle - form api, 2012. URL http://docs.moodle.org/dev/lib/formslib.php_Form_Definition. Last viewed: 26/4-2012.

[29] Moodle - local plugin, 2012. URL http://docs.moodle.org/dev/Local_plugins. Last viewed: 9/5-2012.

[30] Moodle - page api, 2012. URL http://docs.moodle.org/dev/Page_API. Last viewed: 26/4-2012.

[31] Xmldb defining an xml structure, 2012. URL http://docs.moodle.org/dev/XMLDB_defining_an_XML_structure. Last viewed: 25/5-2012.

[32] Pear html_quickform - validation documentation, 2012. URL http://pear.php.net/manual/en/package.html.html-quickform.intro-validation.php. Last viewed: 9/5-2012.

[33] Morten Mathiasen Andersen, 2012. URL http://personprofil.aau.dk/121493?lang=en. Title: Assistant.

[34] Mads Peter Bach, 2012. URL http://personprofil.aau.dk/102508?lang=en. Title: Systems Administrator.

[35] Scott Barge. Principles of problem and project based learing - the aalborg pbl model, Sep 2010. Last viewed: 18/4-2012, Folder describing the Aalborg PBL Model.

[36] Alex MoesgÃ¥rd Bek, Dianna Hjorth Kristensen, Rasmus OxenbÃ¸ll Lund, Nikolaj Dam Larsen, and Kenneth Eberhardt Jensen. E-learning mangement system : Implementing customisable schedules, 2011. URL http://projekter.aau.dk/projekter/files/52575492/Report.pdf.

[37] Barry Boehm and Richard Turner. Observations on balancing discipline and agility. In *Agile Development Conference, 2003. ADC 2003. Proceedings of the*, pages 32 – 39, june 2003. doi: 10.1109/ADC.2003.1231450.

[38] Mikkel Boysen, Christian de Haas, Kasper Mullesgaard, and Jens Laurits Pedersen. E-lms - authentication, database, and educator, 2011. URL http://projekter.aau.dk/projekter/files/52596547/document.pdf.

[39] Individual bugzilla.org contributors. Bugzilla: Features, April 2011. URL http://www.bugzilla.org/features/. Last viewed: 12/3-2012.

[40] Lillian Buus, 2012. URL http://personprofil.aau.dk/profil/103311?lang=en. Title: Staff Member with University Degree.

[41] Mark Drechsler. Moodle structural overview, 2012. URL http://www.slideshare.net/mark.drechsler/moodle-structural-overview. Last viewed: 9/5-2012.

[42] Ryann K. Ellis. Field guide to learning management systems, 2009. URL http://conference.unctlt.org/proposals/presentations/conf4/900_LMSfieldguide1.pdf. Last viewed: 19/4-2012.

[43] Marie Glasemann, 2012. URL http://personprofil.aau.dk/116492?lang=en. Title: Head of Section.

[44] Boris Gloger. Your scrum checklist. Webpage, March 2011. URL http://people.cs.aau.dk/~jeremy/SOE2012/resources/Scrum%20CheckList%202011.pdf. Last viewed: 2012-05-17.

[45] The PHP Group. Package information: Phpdocumentor, 2012. URL http://pear.php.net/package/PhpDocumentor/. Last viewed: 6/3-2012.

[46] Jette Egelund Holgaard, 2012. URL http://personprofil.aau.dk/103630?lang=en. Title: Associate Professor.

[47] Steven Kerschenbaum and Barbara Wisniewski Biehn. Lms selection: Best practices. URL http://www.trainingindustry.com/media/2068137/lmsselection_full.pdf. Last viewed: 26/4-2012.

[48] Anette Kolmos, Flemming K. Fink, and Lone Krogh. *The Aalborg PBL model*. Aalborg University Model, 2004. ISBN 87-7307-700-3.

[49] Craig Larman. *Agile & Iterative Development*. ADDISON WESLEY, first edition, 2004. ISBN 0-13-111155-8.

[50] Nilesh Parekh. The waterfall model explained, September 2011. URL `http://www.buzzle.com/editorials/1-5-2005-63768.asp`. Last viewed: 19/4-2012.

[51] Ron Patton. *Software Testing*. Sams Publishing, 2nd, edition, 2006. ISBN 0-672-32798-8.

[52] Mary Poppendieck. A rational design process â€“ it's time to stop faking it, April 2000. URL `http://www.leanessays.com/2010/11/rational-design-process-its-time-to.html`. Last viewed: 6/3-2012.

[53] Mikkel Todberg, Peter Hjorth Dalsgaard, and Jeppe Lund Andersen. E-lms - administration, calendar, model, educations, courses, 2011. URL `http://projekter.aau.dk/projekter/files/52553822/E_LMS.pdf`.

[54] Gareth Watts. Phpxref, December 2010. URL `http://phpxref.sourceforge.net/`. Last viewed: 6/3-2012.

[55] Moodle Wiki. License, August 2011. URL `http://docs.moodle.org/dev/License`. Last viewed: 29/5-2012.

Appendix

Demo Meeting A

A.1 Lea Gustavson

Date: 07/05/12 Participants: Lea Gustavson, Anders Eiler, Kim Jakobsen, Henrik Koch, Jesper Dalgas Interviewer: Anders Eiler

Lea is given a complete walkthrough of our system. A casual conversation is conducted about the different elements. The following is a summary of the conversation between Lea and Eiler:

A.1.1 Meetings

- A pop-up appears when creation of a meeting is complete, it is disruptive.
- When a user enters the page for a meeting, no back button is visible.

A.1.2 Timeline

- It would be nice if it was possible to show scheduled lectures on the timeline.

A.1.3 Tasks

- When creating a task, the calendar does not close when a date is selected.

A.1.4 Wall

- If several comments exist on an item, it can disrupt the clarity of the wall. It could be nice if older comments collapsed and only new comments are displayed in full.
- The current filter options were not useful, Lea suggest more filtering options.
- It is not all comments that are relevant for the supervisor, and some comments are for the group only. An option to send the message to the supervisor would resolve this problem.
- The structure is very clear and gives a good overview.

A.1.5 Blackboard

- It is currently not possible to assign a name to a blackboard.

A.1.6 Other

- The overall layout seems simple and easy to use.
- The meeting feature is very attractive and Leas thinks that her group will use it often.
- They use SVN for writing the report, it could be integrated with moodle in some way.
- Lea explains that group rooms should be private for the group.

A.1.7 Comments from the observers

- Meetings: It would be nice if it was possible to see what other activities that is arranged on a given date. This is relevant when creating a new meeting.(Henrik Koch)
- General: The name of the buttons are not consistent, some are labeled " submit" and others "save". (Kim AhlstrÃ¸m)
- If the wall only contains a few elements, it displays the next page button even though only one page is available. (Kim AhlstrÃ¸m)
- Lea was not able to edit group settings because she was not admin. It is a problem that admin rights are required to edit a group. (Kim AhlstrÃ¸m)
- It would be nice with a small calendar view in one of the sidebars (Anders Eiler)

A.2 Mathilde Gammelgaard

Date: 09-05-12 Participants: Mathilde Gammelgaard , Anders Eiler, Kim Jakobsen Interviewer: Anders Eiler

Mathilde is given a complete walkthrough of our system. A casual conversation is conducted about the different elements. The following is a summary of the conversation between Mathilde and Eiler:

A.2.1 General

- Not very user friendly.

A.2.2 Members

- Considering that the group members know each other, it takes up a lot of space.

A.2.3 Meetings

- Data is not danish standard.
- Standard meeting should not include supervisor.

A.2.4 Timeline

- It is a good thing that it is dynamic.
- Four months makes the horizontal timeline chaotic.
- Good for overview.

A.2.5 Tasks

- When entering the name of the title one must first delete the "Task title" text.
- A color should be uniquely attached to a type.
- If a huge amount of tasks exit, a view with tasks of the day is wanted.
- A task should contain subtasks.
- In task overview there need to be added a "assigned to" column.

A.2.6 Wall

- It is important that the supervisor is not able to see all messages and files
- It should be an active task to post something to the supervisor
- There is to much emphasizes on filters and item information on the wall. It is disturbing.
- It is a good thing that it is similar to Facebook, it makes it easy to learn.
- Less spacing between text
- When handling multiple documents, an explorer like appearance would be appreciated.

A.2.7 Blackbaard

NOTE: Did not work, but we talked about it.

- Not very usable.
- Add arrow functionality.
- Add equation options.

A.2.8 Comments from the observers

- Task: It is possible to assign task to the supervisor, seems odd. –Kjakob09 09:19, 9 May 2012 (UTC)
- Task: Assign task to members form is not consistent with the rest of the system. (Eiler)
- Task: Overdue tasks should be marked with an ! (Eiler)

Dynamic Black-box Test Cases

B

ID: Test case #1 Create task

Creating a task consists of a number of fields being filled out. This test should make sure that no rogue values are entered into any of the fields, and that the software handles the entered informations correctly in order to create the wanted task.

The input consists of:

- Topic (string)
- Description (string)
- Deadline (date-time)

The output of the function will consist of either an error message or a success message, that the task has either been saved correctly or that an error occurred during processing of the entered data. If it was saved correctly we also expect the new task to show up on the list of tasks.

In order to create a new task, only the creator, data about the new task, and a PC is needed.

Input specification

1. Title: Null - Description: Null - Deadline: Null.
2. Title: 256 characters - Description: 256 characters - Deadline: date-time.
3. Title: Not UTF-8 - Description: Not UTF-8 - Deadline: date-time.
4. Title: UTF-8 content - Description: UTF-8 content - Deadline: Not date-time format.
5. Title: UTF-8 content - Description: UTF-8 content - Deadline: date-time format.

Output specification

1. If any of the input fields is null, reject.
2. If the topic or description is 256 characters, reject.

3. If the topic or description is not UTF-8, reject.
4. If the deadline is not a date-time format, reject.
5. If the topic and description are both UTF-8 content and the deadline is of date-time format, accept.

Environmental needs A single tester working with the system on a PC. The system is installed on a server.

Intercase dependencies This case has no dependencies.

ID: Test case #2 Assign task

Assigning a task to a user consists of combining a user and a task to make the user responsible for this task. All users will be listed in a select-box, so there is no way to enter any rogue data. The only type of error one can make, is to choose either the wrong task or the wrong user.

The input consists of:

- Choosing a task (shows name, returns id).
- Choosing a user (shows name, returns id).

As both are listed and the user does not need to enter any data manually, it is expected that the input data is always correct. It should still, however, be tested to make sure.

Input specification

1. Assignment: Null - User: Null
2. Assignment: Invalid id - User: Invalid ID
3. Assignment: Valid ID - User: Valid ID

Output specification

1. If either the task id or user id is null, reject.
2. If either the task id or user id is invalid, reject.
3. If both the task id and user id are valid, accept.

Environmental needs A single tester working with the system on a PC. The system is installed on a server.

Intercase dependencies This case depends on the create task case.

ID: Test case #3 Remove task

Removing a task can consist of either deleting it or marking it as “done”. This is only doable by clicking a specific button.

The input consists of:

- Action (delete or mark done).
- ID of the task in question.

Since the user will not be entering any data manually, there is no possibility for errors on that part. Still, the action and the id should be checked to make sure they are correct.

Input specification

1. Action: Null - ID: Null.
2. Action: Invalid action - ID: Invalid ID.
3. Action: Delete or Mark Done - ID: Valid ID.

Output specification

1. If either the action or ID is null, reject.
2. If either the action or ID is invalid, reject.
3. If both the task id and user id are valid, accept.

Environmental needs A single tester working with the system on a PC. The system is installed on a server.

Intercase dependencies This case depends on the create task case.

ID: Test Case #4: Change deadline for task

Test Item A task should be created in the system with a correct deadline. The test item is the edit page for the task concerning its deadline. The function tested is the one concerning editing tasks.

Input specification The date for the deadline is changed to:

1. A date earlier than the current date
2. A date before the beginning of the timeline
3. A date after the last date of the timeline
4. A date later than the current date but earlier than the last date of the timeline
5. A wrongly formatted date
    a) A date containing characters
    b) A date with special characters
    c) A date which is too long
    d) A date which is too short
6. No new date

Output specification For the different inputs specified the expected output of the system is:

1. The system should inform the user the date is out of scope

2. The system should inform the user the date is out of scope
3. The system should inform the user the date is out of scope
4. The deadline for the task should be updated and display the entered deadline
5. In all cases the system should inform the user the date is invalid
6. The system should inform the user a date should be entered

The responses given by the system should be correctly formatted and fit within the specified boundaries in the GUI

Environmental needs
A web server running Moodle needs to be available. A tester is needed, and if possible a user to perform the actions.

Special procedural requirements
None

Intercase dependencies
The changes performed to a deadline should be reflected in the timeline. This is tested in Test Case #6 and will therefore not be covered by this test.

ID: Test Case #5: Check the check for if a deadline has been missed works correctly

Test Item A cron job runs at least once everyday checking if any deadlines have been missed. The message system is checked indirectly as the system should inform the user of missed deadlines.

Input specification A task is created and its deadline set to an arbitrary date within the scope of the timeline, but after the current date. For the deadline of this task these tests are performed:

1. The systems date is changed to one after the deadline without uploading a file.
2. A file is uploaded to the task and the systems date is changed to one after the deadline.

Output specification For the different inputs specified the expected output of the system is:

1. A message is send to the person assigned to the task that the deadline is met
2. The task is marked as completed

Environmental needs
A web server running Moodle needs to be available. A tester is needed to observe the

correct actions are taken by the cron job.

Special procedural requirements
None

Intercase dependencies
The completion of a task and missing of a deadline should be reflected in the timeline. This is tested in test case #6 and will therefore not be covered by this test.

ID: Test case #6 Proper representation of deadlines

The deadline of a task should be shown in the timeline, based on the whether the task is completed or not completed, and whether the deadline is expired or not.

Test Item Testing that various kinds of deadlines are displayed properly.

Input specification

1. A task is created with a deadline later the same day.
2. A task is created with a deadline a week from the current time.
3. A task is created with a deadline, and the server clock is then advanced such that the deadline is expired, and on the current day.
4. A task is created with a deadline later the same day, the task is then marked as complete.
5. A task with a deadline is created, and the task is marked as complete. The server clock is then advanced such that the deadline is earlier the same day.

Output specification

1. The deadline should be shown in the timeline as a normal, non-expired deadline.
2. The deadline should be shown in the timeline as a normal, non-expired deadline
3. The deadline should be shown in the timeline as an expired deadline, on the current day.
4. The deadline should be shown in the timeline later the current day, and the deadline should be shown as a completed deadline.
5. The deadline should be shown in the timeline earlier the current day, and the deadline should be shown as a completed deadline.

Environmental needs
A single tester working with the system on a PC. The system is installed on a server. The server clock will have to be changed for this test, and changed back after it.

Intercase dependencies
This case depends on the task creation process.

ID: Test case #7 Format of task on the task list

The system has a list containing all tasks. This list needs to be properly formatted and not look obscure if given strange input.

Test Item Testing the formatting of tasks on the task list.

Input specification

1. A task is created with a title of 2 characters, a description of 2 characters, and no deadline.
2. A task is created with a title of 2 characters, a description of 2 characters, and a deadline.
3. A task is created with a title of 254 characters, a description of 2 characters, and no deadline.
4. A task is created with a title of 254 characters, a description of 2 characters, and a deadline.
5. A task is created with a one worded title of 254 characters, a description of 2 characters, and no deadline.
6. A task is created with a one worded title of 254 characters, a description of 2 characters, and a deadline.
7. A task is created with a title of 2 characters, a description of 254 characters, and no deadline.
8. A task is created with a title of 2 characters, a description of 254 characters, and a deadline.
9. A task is created with a title of 2 characters, a one worded description of 254 characters, and no deadline.
10. A task is created with a title of 2 characters, a one worded description of 254 characters, and a deadline.
11. A task is created with a title of 254 characters, a description of 254 characters, and no deadline.
12. A task is created with a title of 254 characters, a description of 254 characters, and a deadline.
13. A task is created with a one worded title of 254 characters, a one worded description of 254 characters, and no deadline.
14. A task is created with a one worded title of 254 characters, a one worded description of 254 characters, and a deadline.

Output specification

1. The task list should show the task without issues. Both the title and the description must fit within the borders. It should not show anything in the deadline field.
2. The task list should show the task without issues. Both the title and the description must fit within the borders. It should also show the correct date in the deadline field.

3. The task list should show the task without issues. The title should wrap when approaching borders, and the description should still be within the borders. It should not show anything in the deadline field.
4. The task list should show the task without issues. The title should wrap when approaching borders, and the description should still be within the borders. It should also show the correct date in the deadline field.
5. The task list should show the task without issues. It should hyphen the one worded title so it wraps, while keeping the description the same. It should not show anything in the deadline field.
6. The task list should show the task without issues. It should hyphen the one worded title so it wraps, while keeping the description the same. It should also show the correct date in the deadline field.
7. The task list should show the task without issues. The description should wrap around the borders, and the title should fit within the borders. It should not show anything in the deadline field.
8. The task list should show the task without issues. The description should wrap around the borders, and the title should fit within the borders. It should also show the correct date in the deadline field.
9. The task list should show the task without issues. The description should be hyphened to enable wrapping words at borders, the title should also fit within the borders. It should not show anything in the deadline field.
10. The task list should show the task without issues. The description should be hyphened to enable wrapping words at borders, the title should also fit within the borders. It should also show the correct date in the deadline field.
11. The task list should show the task without issues. Both the title and the description should wrap at borders, and they should not make the fields so wide that there is not room for both deadline and status. It should not show anything in the deadline field.
12. The task list should show the task without issues. Both the title and the description should wrap at borders, and they should not make the fields so wide that there is not room for both deadline and status. It should also show the correct date in the deadline field.
13. The task list should show the task without issues. Both the title and the description should hyphen the word so it wraps properly at borders, and they should not make the fields so wide that there is not room for both deadline and status. It should not show anything in the deadline field.
14. The task list should show the task without issues. Both the title and the description should hyphen the word so it wraps properly at borders, and they should not make the fields so wide that there is not room for both deadline and status. It should also show the correct date in the deadline field.

Environmental needs

A single tester working with the system on a PC. The system is installed on a server.

Intercase dependencies

This case depends on the task creation process.

ID: Test case #8 Complete task

Test Item Completing a task through the task list

Input specification

1. An ID that corresponds to an existing task in the database.
2. An ID that does not correspond to an existing task in the database.

Output specification

1. The task with the given ID is marked as complete.
2. An error saying that an incorrect ID was inputted.

Environmental needs
A database with valid tasks and a server running Moodle with the timeline plugin installed.

Special procedural requirements
None.

Intercase dependencies None.

ID: Test Case#9 Uncomplete task

Test Item Uncompleting a task through the task list.

Input specification

1. An ID that corresponds to an existing task in the database.
    a) On a task which deadline is not missed and that has people assigned to it
    b) On a task which deadline has passed
    c) On a task which deadline is not missed and has no people assigned to it
2. An ID that does not correspond to an existing task in the database.

Output specification

1. The task with the given ID is given the correct status
    a) The task is marked as in progress if people are assigned to it and its deadline is not missed
    b) The task is marked as overdue if the deadline is missed
    c) The task is marked as on hold if no people are assigned to it and its deadline is not missed
2. An error saying that an incorrect ID was inputted.

Environmental needs
A database with valid tasks and a server running Moodle with the timeline plugin installed.

Special procedural requirements
None.

Intercase dependencies
None.

Dynamic Black-box Testing Results

C

Index	Input Specification	Result
1	**Topic:** Null **Description:** Null **Deadline:** Null.	No task created - redirects to project page. No error on empty title
2	**Topic:** Too long **Description:** Too long **Deadline:** Date-time	Title too long results in a database error
3	**Topic:** Not UTF-8 **Description:** Not UTF-8 **Deadline:** Date-time	Task created without issues
4	**Topic:** UTF-8 content **Description:** UTF-8 content **Deadline:** non date-time format	Irrelevant due to a date-time selector being used to set deadline
5	**Topic:** UTF-8 content **Description:** UTF-8 content **Deadline:** date-time format.	Task created without issues

Figure C.1. Test results for Test Case #1

Index	Input Specification	Result
1	**Assignment:** Null **User:** Null	Performs no actions
2	**Assignment:** Invalid ID **User:** Invalid ID	Performs no actions
3	**Assignment:** Valid ID **User:** Valid ID	Success - a user is added to the task

Figure C.2. Test results for Test Case #2

Index	Input Specification	Result
1	**Action:** Null **ID:** Null	Performs no actions
2	**Action:** Invalid action **ID:** Invalid action	Performs no actions
3	**Action:** Delete or Mark Done **ID:** Valid ID	Task is successfully deleted

Figure C.3. Test results for Test Case #3

Index	Input Specification	Result
1	A date earlier than the current date	Possible, but the task is now overdue
2	A date before the beginning of the timeline	Is not shown on timeline but change is done successfully
3	A date after the last date of the timeline	Is not shown on timeline but change is done successfully
4	A date later than the current date but earlier than the last date of the timeline	Date changed successfully - Timeline changes it view to accommodate for the new deadline if needed
5	A wrongly formatted date	Not performed as the date-time selector makes this impossible
6	No new date	Deadline removed, task has no deadline

Figure C.4. Test results for Test Case #4

Index	Input Specification	Result
1	The systems date is changed to one after the deadline without uploading a file.	A task is marked as overdue - system does not require an uploaded file for a task to be completed
2	A file is uploaded to the assignment and the systems date is changed to one after the deadline.	The task is marked as complete - system does not require an uploaded file for a task to be completed

Figure C.5. Test results for Test Case #5

Index	Input Specification	Result
1	An assignment is created with a deadline later the same day.	Successful
2	An assignment is created with a deadline a week from the current time.	Successful
3	An assignment is created with a deadline, and the server clock is then advanced such that the deadline is expired, and on the current day.	Successful - task is created but marked as overdue
4	An assignment is created with a deadline later the same day, the assignment is then marked as complete.	Successful - marked as complete
5	An assignment with a deadline is created, and the assignment is marked as complete. The server clock is then advanced such that the deadline is earlier the same day.	Successful

Figure C.6. Test results for Test Case #6

Index	Input Specification	Result
1	An ID that corresponds to an existing task in the database	Task is marked as completed
2	An ID that does not correspond to an existing task in the database	Only possible through the URL and no actions are taken with invalid IDs

Figure C.8. Test results for Test Case #8

Index	Input Specification	Result
1	**Title:** 2 characters **Description:** 2 characters **Deadline:** Null.	Success. All columns are stretched enough to keep the content in one row.
2	**Title:** 2 characters **Description:** 2 characters **Deadline:** Date-time	Success. See previous.
3	**Title:** 254 characters **Description:** 2 characters **Deadline:** Null.	Success. The title column has expanded and wraps around on separation of words.
4	**Title:** 254 characters **Description:** 2 characters **Deadline:** Date-time	Success. See previous.
5	**Title:** 254 characters long one word **Description:** 2 characters **Deadline:** Null.	Success. The title column hyphens the word so it wraps around.
6	**Title:** 254 characters long one word **Description:** 2 characters **Deadline:** Date-time	Success. See previous.
7	**Title:** 2 characters **Description:** 254 characters **Deadline:** Null.	Success. The description has expanded, but other columns are still at a width keeping them on one line.
8	**Title:** 2 characters **Description:** 254 characters **Deadline:** Date-time	Success. See previous.
9	**Title:** 2 characters **Description:** 254 characters long one word **Deadline:** Null.	Success. The description is hyphened when the word approaches the border, so the word continues on the next line.
10	**Title:** 2 characters **Description:** 254 characters long one word **Deadline:** Date-time	Success. See previous.
11	**Title:** 254 characters **Description:** 254 characters **Deadline:** Null.	Success. The title and description fields are expanded forcing the status field to have its text on one to two lines.
12	**Title:** 254 characters **Description:** 254 characters **Deadline:** Date-time	Success. See previous.
13	**Title:** 254 characters long one word **Description:** 254 characters long one word **Deadline:** Null.	Success. Both the title and description are hyphened so the word wraps. The status field is again forced to be on one to two lines.
14	**Title:** 254 characters long one word **Description:** 254 characters long one word **Deadline:** Date-time	Success. See previous.

Figure C.7. Test results for Test Case #7

Index	Input Specification	Result
1.a	An ID that corresponds to an existing task in the database - On a task which deadline is not missed and that has people assigned to it	Task is marked as In progress
1.b	An ID that corresponds to an existing task in the database - On a task which deadline has passed	Task is marked as Overdue
1.c	An ID that corresponds to an existing task in the database - On a task which deadline is not missed and has no people assigned to it	Task is marked as on hold
2	An ID that does not correspond to an existing task in the database	Only possible through the URL and no actions are taken with invalid IDs

Figure C.9. Test results for Test Case #9

Integration Test Results D

1. Admin - Advisor should be changed to supervisor everywhere.
2. Admin - The type variable has not been set.
3. Admin - Fix tooltips on the list of project groups.
4. Admin - Consider which role a user should receive after being removed as supervisor.
5. Blackboard - When you resize figures you can move them unintentionally.
6. Blackboard - Creating a textarea results in a server crash.
7. Blackboard - Line and Penline cannot be moved.
8. Generally - Margins on buttons.
9. Supervisor - Comment Textbox needs to be toggle-able with jQuery.
10. Supervisor - Turn editing on removes the id from the URL, PAGE should be used to get id and context.
11. Supervisor - Turn editing on means that meetings cannot find the id.
12. Supervisor - Meeting module needs to be better with representing what it does.
13. Supervisor - Anchor points for meetings so the timeline can link to them.
14. Supervisor - You can only edit meetings. Remove the functionality or implement it the other places where "Edit" is available.
15. Supervisor - Spaces are needed under Group Wall.
16. Supervisor Wall - Shows all groups that you are a member of, and not only those you are supervisor of.
17. Supervisor Wall - Provide link to the group walls.
18. Supervisor Wall - Provide links to everything, e.g. profiles.
19. Supervisor Wall - Not actually deleting when it should.
20. Timeline - The objects become weirdly formatted when they are large. Their spacing is probably relative to the size, or it does it incorrectly compared to vertical/horizontal.
21. Timeline - Create links for meetings to an anchor point, so the Timeline links down to the group wall instead of the calendar.
22. Timeline - Have a look at the boxes that pop up when the mouse is hovered over an object at the bottom of the page.
23. Timeline - When there's a link to the edit page you can arrive from both project page and course page.
24. Timeline - Delete task page does not render.
25. Timeline - The task list page says there are no tasks.
26. Timeline - A year in the future on the timeline causes the box to go outside of the timeline.

27. Timeline - When the year changes you start in week 0.
28. Timeline - Edit task sets the deadline to today.
29. Timeline - Hide complete task on completed tasks on the task list. It should be replaced by a minus and uncompleted text.
30. Timeline - Consider the same form for edit and create tasks.
31. Timeline - Find the difference between course and project group.

Summary E

Aalborg University is famous for its' use of PBL *(Problem Based Learning)*, however the e-learning tool used by the university staff and students, Moodle, does not have support for PBL. During interviews with students we found that they had to use several different tools in addition to Moodle to support their project work at AAU. This lead us to the problem of extending Moodle to make it better suited for PBL. This is a daunting task as Moodle is a big system, so it was meant to be spread both over several groups, and several semesters. In this semester four groups worked on this problem, with our group being tasked with creating a system for project organization and time planning. As all 4 groups had to work together to deliver one coherent system, a development method which supported this kind of group work had to be chosen. To this end we chose the agile development method Scrum of Scrums, which was modified to suit out needs. To complete our part of the system we implemented the concept of tasks in Moodle. Tasks can be given a deadline, and multiple users can be assigned to each task. In addition to tasks we implemented a timeline in Moodle, which gives a overview of past and future events, such as upcoming task deadlines. This timeline was implemented to be easily extensible to show other types of events, such as lectures. The timelines functionality is implemented in PHP, and its GUI is created using HTML, Javascript, JQuery, Cascading Style Sheets *CSS* and Moodle Forms. The finished timeline was able to show tasks, in addition to meetings and blackboards, which were implemented by the other groups. We invited the interviewed users back after finishing the implementation, and they were happy with the result, though they did have suggestions for improvements. So while we did improve Moodles support for PBL, there is still work to be done when the project is continued next year.