Aalborg University
Department of Computer Science

Title:
ProjectTracker

Topic:
Programming in the large

Project Period:
2/9 2008 - 19/12 2008

Project Group: s303a
Kenneth Madsen
Jens-Kristian Nielsen
Frederik Højlund
Søren Magnus Olesen
Heine Pedersen
Peter L. Hansen

Supervisor:
Nadeem Iftikhar

Number of appendixes: 4

Total number of pages: 93

Number of pages in report: 72

Number of reports printed: 8

Abstract:

The purpose of this project was to develop a web-based project support tool based on predefined requirements. We analyzed the requirements in the design phase, determining the design of the product. Based on our design, we developed our *ProjectTracker*, which we have used in this semester and will use it in following semesters. Different tools and methods are described and evaluated. Furthermore we document the process and suggest ideas for further development. The experiences gained from developing this program are also described in the report. Finally we evaluate and conclude our project.

Preface

This project was made by group s303a during the third semester, in fall 2008, of Software Engineering at the Department of Computer Science at Aalborg University. The theme of this semester was "Programming in the large". In this project the focus was not on analyzing a problem, but more minded on solving a predefined problem. It covers the creation of a Project Manager.

This report requires a basic understanding of programming - especially C# and ASP.NET. On the attached CD-ROM you will find the source code for the system along with a SQL dump of the database scheme and test data. You will also find documentation of the program. A live demonstration of *ProjectTracker* can be found at http://www.cs.aau.dk/~kenmad and screenshots of all pages in appendix D on page 87.

Kenneth Madsen

Jens-Kristian Nielsen

Frederik Højlund

Søren Magnus Olesen

Heine Pedersen

Peter L. Hansen

Contents

Chapter

1

Introduction

1.1 About this report

This report documents the following topics; project analysis, problem analysis, design, solving and evaluation of a *web-based project support tool* for use in a university project group. The system is to be used as an aid tool when developing software projects, to keep track of project status, tasks, time and human resources.
The requirements are based on the project specification in appendix A on page 76, which we were handed at the beginning of the project period.
The system must be developed in an object oriented language, according to the study programme, furthermore the system must be fully tested, including tests from automated testing. This will insure the quality of the software and that the project meets the specified requirements.

All the decisions regarding the system development must be thoroughly documented and substantiated in this report. The experiences acquired in the process, are to be documented in the report, and whether the support tool itself could be used in the project. The system is considered useful if it can be used in this and later projects.
As a part of the project, we will choose a development method, and document our use of it.

Chapter

2

Project Analysis

2.1 Problem Formulation

When developing nontrivial software in a development group, a wide array of tasks and problems often arise. Especially in relation to human resource management, fixing bugs and implementing new functionalities. Each task needs to be assigned to a developer or offered to a group of developers, its status can vary, depending if it has been submitted, assigned or implemented/fixed. Tasks may also have different priority levels depending on how vital they are to the system. They may have a deadline and they may have an implementation priority. As the amount of tasks becomes larger, managing all this information can easily become very complex and time consuming.

2.2 Initiating Problem

How can we develop a software system that eases management of otherwise unorganized development processes, development tasks and resources?

Chapter 3

Requirements

3.1 Introduction

This chapter covers the product requirements and describes the program and its functionalities. The chapter will elaborate on different users of the system, sorting by user roles and privileges. It will explain which users are allowed to use which functionalities. The chapter continues with functional and non-functional requirements and will provide an overview of all functionalities of *ProjectTracker*. The quality goal section will cover expectations to the importance of different quality aspects of the program, usability for instance.

3.2 Procedural Requirements

This section covers the overall functionality of the system and describes the individual functionality in details. For getting an overview of the functionalities of the system, a use case diagram is displayed on figure 3.1.

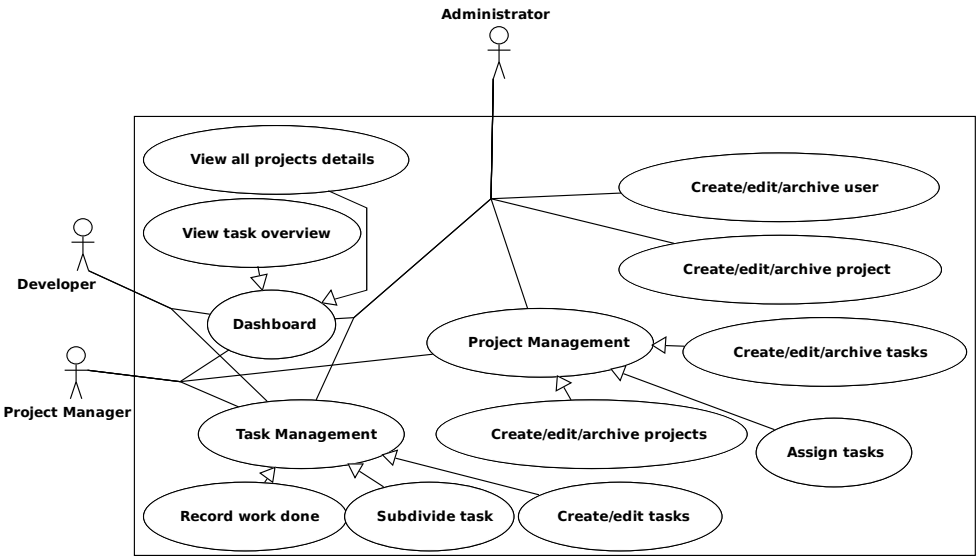


Figure 3.1: Use case diagram.

Please note that the administrator also have the functionalities of the developer and the project manager. As described in the requirements, the users of the system has one (or more) of three roles: Developer, Project Manager or Administrator. Some functionalities are available for all users.

All users

All users has access to a dashboard. The purpose of the dashboard is to give a quick overview of the ongoing tasks. The dashboard has the following main functionalities:

- View all projects details
  This functionality gives the user insight into the status of the project: "Is it delayed?", "how many hours remains?" and so on.

- View task overview
  This functionality gives the user the possibility to see the status of all the tasks and makes it easy for the user to determine which coworker has time to help.

Developer

The developer has further access to task management. The functionalities related to task management is:

- Create/edit tasks
  The developer can add tasks.

- Subdivide a task
  The developer can also divide a task into several subtasks, if the developer finds that the main-task is too complex or overwhelming.

- Record Work Done
  Here the developer records the hours spent on the task. This enables the project manager to see how many hours that has been spent on the task and the project in total.

Project Manager

The project manager has further access to project management. This page contains all functionalities concerning project management. These functionalities are:

- Create/edit/archive projects
  A project manager can create a new project and edit existing ones. This involves to change the deadline of a project. If a project is canceled while in development the project manager can archive the project.

- Create/edit/archive tasks
  In the project the project manager can create tasks so the project is divided into smaller problems. These tasks can be edited later if the

requirements or priorities change. Archiving a task can also be done if e.g. the customer does not want that functionality in the final system.

- Assign tasks
  The last functionality in project management is to hand out tasks to a developer or a group.

Administrator

The administrator has all the above mentioned access and functionality. Furthermore the administrator has access to the following functionalities:

- Create/editing/archiving a user
  The administrator has access to adding, archiving and editing users of the system.
- Create/editing/archiving a project
  It is the administrators responsibility to add a new project or to edit one. Furthermore the administrator can also archive a project.

3.3 Functional and Non-Functional Requirements

System requirements can be divided into two categories: Functional and non-functional requirements. Functional requirements are requirements of the systems functionality or behavior. For instance if it is required that the system should encrypt tasks, that would be a functional requirement. Non-functional requirements are also called qualities, constraints, quality attributes, quality goals etc. They represent how the operation of the system has to be[18]. For instance how well encrypted the tasks has to be, is a non-functional requirement. Non-functional requirement are security, usability, testability, maintainability, extensibility and scalability.

Our Functional Requirements

Most of the requirements we have written in the requirement specification are functional requirements. This may be because our system will be a first release and getting the system up and running is the first priority. All our core requirements are functional requirements. Descriptions of the requirements can be viewed in the requirements appendix A on page 76.

- Group members
- User Roles
- Interface
- Projects

- Main Activities
- Task
- Schedule
- Monitoring
- Estimation* (both functional and non-functional)
- Source Control Integration
- Test and Review
- Grouping of tasks
- Documentation
- Bug Reports

Our Non-Functional Requirements

- Estimation*
- Quality Goals**
  - Usability
  - Integrity
  - Effectiveness
  - Correctness
  - Reliability
  - Maintenance
  - Testability
  - Flexibility
  - Reusability
  - Portability
  - Interoperability

*Estimation is both a functional and non-functional requirement, because the estimation requirement is not only a requirement of the user being able to view and estimate, but also a requirement of a certain quality/precision.
**Quality goals are explained in section 3.4 on the following page.

3.4 Quality Goals

Below is a rated list of the different quality goals. The following rating scale will be used:

1. Not important
2. Less important
3. Important
4. Very important

Besides the rating, each quality goal has a small description explaining the reason of its rating. We will also explain how the quality of the goals that we rate *important* and *very important* will be measured.

Assessment

Usability - **Less Important**

Effort required to learn, operate, prepare input, and interpret output of a program

According to the project requirements no big effort should be put in to make a system with a high degree of usability, but when the evaluation criteria is whether we should use the system in the next semester we have justified a rating of less importance.

Integrity - **Less Important**

Extent to which access to software or data by unauthorized persons can be controlled

The integrity is rated less important because the goal of this project is not to create a secure project manager. The data within the system should be isolated from intruders. Therefore it should be an access controlled area, and should have some degree of importance.

Effectiveness - **Less Important**

The amount of computing resources and code required by a program to perform a function

With each iteration of the development process the code will be improved and/or optimized but it is not a high priority for this project to have a fast

code execution.

Correctness - **Very Important**

Extent to which a program satisfies its specifications and fulfills the user's mission objectives

It is very important that the product fulfills the requirements, that is why we have rated correctness very important.

Reliability - **Important**

Extent to which a program can be expected to perform its intended function with required precision

As the software is not mission-critical the reliability is not very important, but we want the software to have a good degree of reliability, and thus rating it as important.

Maintenance - **Important**

Effort required to locate and correct an error in an operational program

Using our own project manager in the development process is one of the success criteria for this semester. For that reason maintenance is rated important in order for us to correct errors later.

Testability - **Important**

Effort required to test a program to ensure it performs as intended

Testability is rated important. It is a requirement that we test the system for its correctness and prove that the system fulfills the requirements.

Flexibility - **Important**

Effort required to modify an operational program

Since we will start using the system in this semester, and hoping to be using it next semester, we want to make it easy to modify, should new needs arise.

Reusability - **Not Important**

Extent to which a program can be used in other applications related to the packaging and scope of the functions that the program performs

Reusability is a core idiom of object oriented programming, but is not a part of our requirements.

Portability - **Important**

Effort required to transfer a program from one hardware configuration and/or environment to another

We want the system to be able to run on most operating systems and it would not be forced to undergo another development phase in order to use it on a different operating system.

Interoperability - **Not Important**

Effort required to couple one system with another

Based on the requirements interoperability has been rated not important. The goal of the project is to create a complete system and not be dependent on third party applications.

Table 3.1 is a summary of how we rated the quality goals.

Name	**Very Imp.**	**Imp.**	**Less Imp.**	**Not Imp.**
Usability			✗	
Integrity			✗	
Effectiveness			✗	
Correctness	✗			
Reliability		✗		
Maintenance		✗		
Testability		✗		
Flexibility		✗		
Reusability				✗
Portability		✗		
Interoperability				✗

Table 3.1: Summary of quality goals. Imp. is short for important.

Measurement

In this section we will describe the important/very important quality goals and how we will insure that we reach our goals. We will do this by assigning a method of measuring the goal and give minimum requirements for success.

Correctness

We will measure the correctness of the system by how many of the requirements it fulfills. The minimum requirement for success is a system that fulfills all the core requirements.

Reliability

We will measure the reliability of the system by using unit and user tests. The minimum requirement for success is to have a system with no known crashes or deficiencies.

Maintenance

We will measure the maintainability of the system by the time an error is reported to it is fixed. The minimum requirement for success is one day from discovery until the error is fixed.

Testability

We will measure the testability of the system by how much of the code we can test using unit tests. The minimum requirement for success is that most of the testable code is tested.

Flexibility

We will measure the flexibility of the system by how easy it is to implement new features. The minimum requirement for success is that the system design should not change when we implement a new feature.

Portability

We will measure the portability of the system by the amount of operating systems the system works on. The minimum requirement for success is to have the system to run on at least two operating system.

Table 3.2 on the following page is a summary of how we measure the quality goals and what the minimum requirement for success is. “Measured How”

is how we are going to measure the given goal and "Limit value" is the minimum requirement for success.

Quality goal	**Measured how?**	**Limit value**
Correctness	How many requirements it fulfills	All core requirements are in the system
Reliability	Testing	No known crashes or deficiencies
Maintenance	The time from report until the error is fixed	One day
Testability	How much of the testing we can automate	most of the testable code is tested
Flexibility	How easy we can implement new features	The system design is not changed when we implement a new feature
Portability	Number of operating systems	Minimum two

Table 3.2: Measurement schema.

Chapter

4

Problem Analysis

4.1 Introduction

As we stated in our problem formulation, we need to develop a software system that eases management of the tasks that occur in the development process. The requirement specifications in appendix A on page 76 defines what needs to be implemented. In this chapter we analyze these requirements and design our system accordingly.

4.2 Overall Design

As specified in the core requirement specification, the system should be accessible through an Internet browser. We will develop a web based system, that can present the relevant data to the user from a common database. The database contains all data about projects, tasks and different types of users.

As specified in the requirements the users will access the system via an Internet browser, so we need to build a client/server system, where the user uses a web browser. A data flow diagram for the initial design can be seen on figure 4.1.

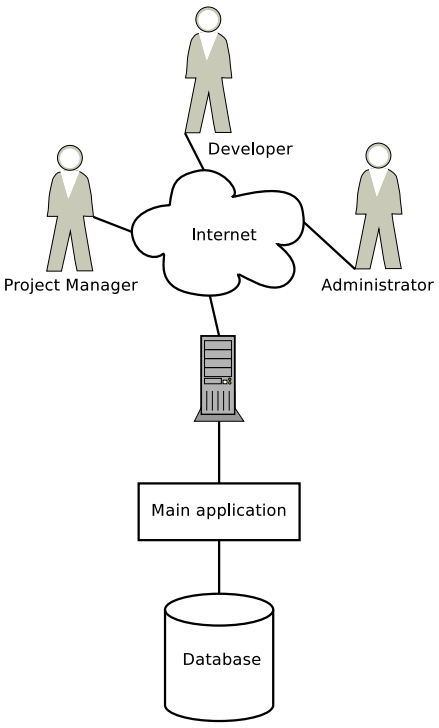


Figure 4.1: Overall Design.

Client

The client side will be a simple web interface viewed in a browser. The main focus of this project is functionality and to a certain extent security. We will not be focusing on usability in the client application.
Since the system contains non-public information, it will be protected by only allowing users with accounts to log in. The account functionality will make it possible to present relevant data to the user.

Main/Server application

The main application is the actual program in our system. It is what generates the HTML, that the client browser interprets. The server application takes input from the client browser and queries the database for the right data. The application also controls users, their permissions and finally it generates the output to the user.

Database

The database will manage all the persistent data in our system. The database will be modeled in a rational database, and the schematics are covered in section 4.6 on page 24.

4.3 Design Patterns

A design pattern describes a problem that occurs multiple times, and is not solely intended for software construction. It also describes a solution to this problem, that has been proved to work well.

According to *Gamma et al.* [1] a pattern has 4 elements:

- A Name
- Problem
- Solution
- Consequences

Names gives people common ground for talking about specific problems and helps discuss them. The name (and the use of a pattern itself) helps people that are reading the code to understand it better.
The Problem describes the issue and where to apply the design pattern.
The Solution describes the elements that make up the solution and how they interact with each other.

The Consequences describes the trade-offs of using a particular design pattern.
Design Patterns are not bound to any particular programming language. Often the problem patterns arise in every language, and the design pattern gives a well working approach to solving a certain design challenge no matter the language [1].

Singleton

The singleton pattern is a common design pattern in software engineering. A singleton is a class in an OOP language, that only allows one instance object of itself to be constructed. Whenever an instance object of the singleton class is needed, rather than creating a new object, the first object created is reused. It rarely allows parameters at creation, because a second request of that instance with different parameters may conflict. A singleton can be "lazily" created, which means that it is not created before its first use.

The example shown in source code 4.1 is a simple implementation of a lazy singleton pattern in C# [20].

```
1  public sealed class Singleton
2  {
3      static readonly Singleton instance = new Singleton();
4
5      static Singleton()
6      {
7      }
8
9      Singleton()
10     {
11     }
12
13     public static Singleton Instance
14     {
15         get
16         {
17             return instance;
18         }
19     }
20 }
```

Source Code 4.1: C# singleton pattern.

Consequences when using a Singleton pattern are [1]:

1. Controlled access to sole instance.
   Because the Singleton class encapsulates its sole instance, it can have strict control over how and when clients access it.

2. Reduced name space.
   The Singleton pattern is an improvement over global variables. It avoids polluting the name space with global variables that store sole instances.

3. Permits refinement of operations and representation.
   The Singleton class may be a superclass, and it is easy to configure an application with an instance of this extended class.

Model View Controller

A Model View Controller (MVC) is a design pattern designed to divide the code in three parts: Model, View and Controller. These parts are so specific that they do not contain anything from the other parts. This means that the data classes do not contain any information about the user interface nor do they have specific presentation layers. The user interface classes only deals with the user interface and has no data modeling.

Figure 4.2 represents the structure of a MVC Design Pattern, the solid lines indicate method invocations, and the dashed lines indicate events [9].

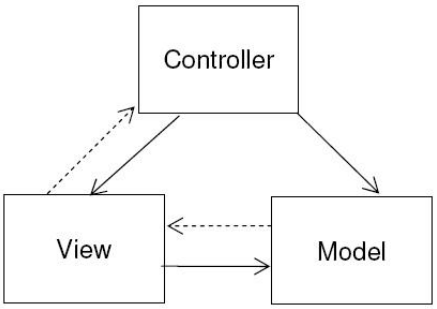


Figure 4.2: MVC.

Model

The Model is a representation of data or information. The classes here contains only information and no functionality. For storing the data the Model often communicates with a database and in most cases the database is considered to be a part of the model.

View

The View is the interface the user interacts with. Here the design pattern separates the interface from the rest of the code. In most cases it can not be done without some functional code, but it gives a greater overview.

Controller

The Controller is where all functionality is located. This is where the operations on the Model-classes is performed. The View triggers a method in the Controller part and then the Controller operates on the Model-classes and generates the next view [21].

4.4 Model Classes

This section will describe the classes we have designed for our object oriented system. The class diagram in figure 4.3, shows the designed structure of the system.

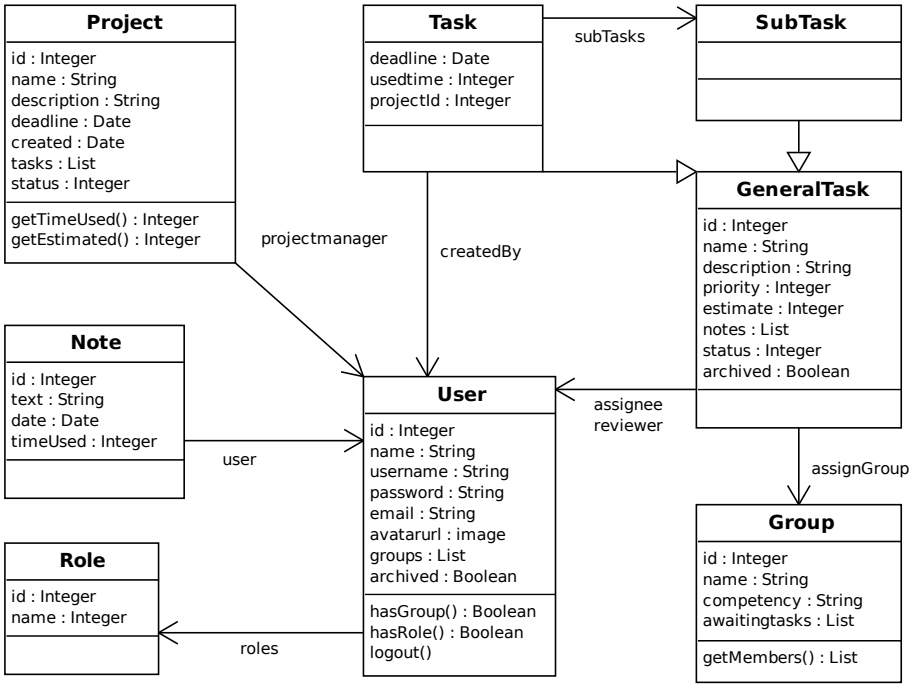


Figure 4.3: Class Design.

User

The User class is used to store information about users of the system. In addition to storing login details, this class holds information about user privileges and competencies. Functionalities are based on the privileges given in this class. Furthermore it can tell if the user has a specific role or is a member of a specific group. A user can be archived if the developer

is not needed for a while. This also results in a login error that barres the user from logging in and use the system. This class also controls the logout routine for the user.

Group

The group class is used to organize users into categories. Every user will belong to one or more groups according to their competencies. This helps assigning the right tasks to the right people. Tasks can be offered to competency groups. Members of relevant groups can assign these tasks to themselves. The members of those groups can then themselves sort our who gets the tasks.

Role

The role class is used to give the users different access levels. For instance, there will be a role called “Administrator” which gives access to more features, like user and project administration.

Project

Project is the class for handling a development project. It is used to organize current tasks. This class also keep track of information about how well the project is keeping within the time frame. It keeps track of work hours spent on the project and the project deadline. The Project class will also have manager of the project and a small description.

GeneralTask, Task and SubTask

The task class will store information about the task. Mainly who it is assigned to (group or user), it’s deadline and priority level. Unless the task is awaiting a developer in the group list, a task will have a developer and maybe a reviewer assigned to it. These person(s) will carry out the work described in the task description. We will have tasks and subtasks so it will be possible to split a task into smaller subtasks, so the developer can get a better overview. The use of subtasks is optional.

The Task and SubTask will inherit from the GeneralTask and the difference between Task and SubTask is that SubTask does not have a deadline. Bugs can be created as sub tasks, to imply that they are related to the parent task. This may also improve "time used" accuracy for the parent task.

Note

The note class is used by developers to inform coworkers about how the task is progressing and how much time spent on solving the task.

4.5 Controller Classes

Controllers are static classes that are accessible throughout the system. They contain methods that are used to modify the data models. They enforce a caching mechanism, that avoids fetching identical data more than once.

UserController

Figure 4.4: UserController Class Design.

The UserController class, as shown in figure 4.4, controls the users of the system. It uses the User class described in section 4.4 on page 20. The class contains methods to create new users, archive users and change the users role or group memberships etc.

GroupController

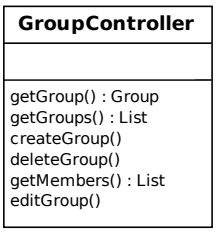


Figure 4.5: GroupController Class Design.

The GroupController class, as shown in figure 4.5 on the preceding page, handles all groups. It is used to add, delete and edit groups and is also able to get all the members of a specific group. This class uses the Group data class.

RoleController

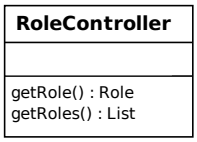


Figure 4.6: RoleController Class Design.

The RoleController class, as shown in figure 4.6, controls roles. RoleController uses the Role class to retrieve informations about a role. It is possible to get a specific role or all the roles defined in the system.

ProjectController

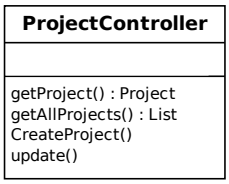


Figure 4.7: ProjectController Class Design.

The ProjectController class, as shown in figure 4.7, handles the creation and updating of a project. It is used to retrieve either a single project or all the projects in the system.

TaskController

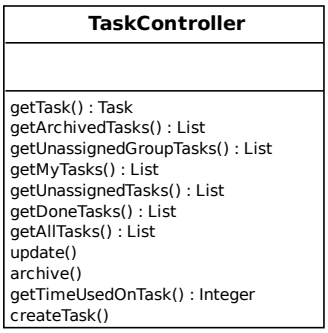


Figure 4.8: TaskController Class Design.

The TaskController class, as shown in figure 4.8, retrieves information about a single task or a collection of tasks. It contains methods for subsets of tasks based on different filtering, for instance archived, unassigned or done. This class also controls how the task is updated, created and archived.

NoteController

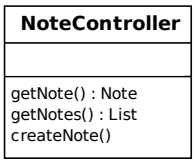


Figure 4.9: NoteController Class Design.

The NoteController class, as shown in figure 4.9, retrieves all the notes for a specific task. It also controls the creation of the note.

4.6 Database Design

This section is divided into subsections. Each subsection corresponds to a database table. The Entity Relationship (ER) model is available in appendix B on page 83.

User table

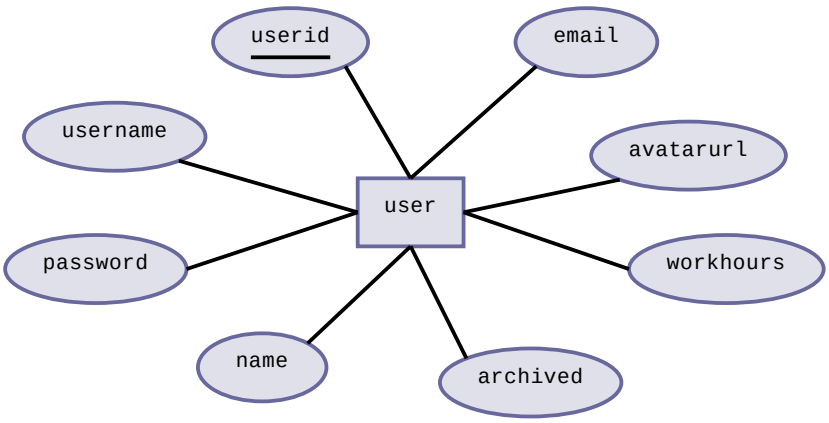


Figure 4.10: User ER model.

As shown in figure 4.10, the *user* table contains a unique userid, a username, a password, a name, whether it is archived or not, the amount of work hours available, an avatar URL and a email address.

Group table

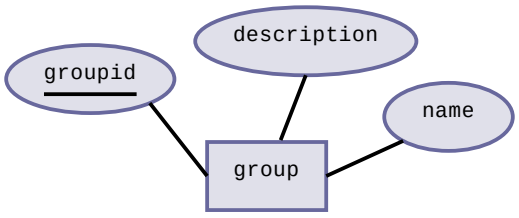


Figure 4.11: Group ER model.

As shown in figure 4.11, the *group* table contains a unique groupid, a description and the name of the role.

Role table

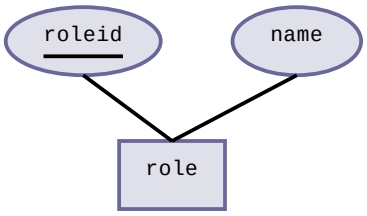


Figure 4.12: Role ER model.

As shown in figure 4.12, the *role* table contains roles with a unique roleid and the title of the role.

Project table

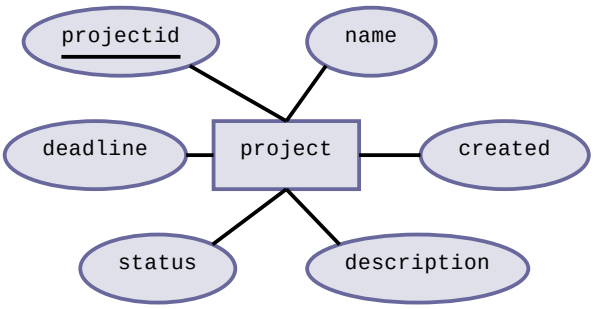


Figure 4.13: Project ER model.

As shown in figure 4.13 the *project* table contains a unique projectid, the deadline of the project, the current status, a small description, the creation date and the name of the project.

Task table

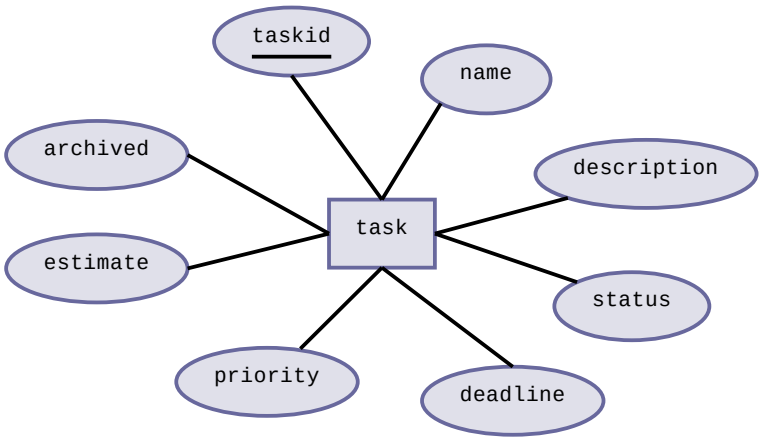


Figure 4.14: Task ER model.

As shown in figure 4.14, the *task* table contains a unique taskid, whether the task is archived or not, the estimated time, the priority, the deadline, current status a small description and the name of the task.

Note table

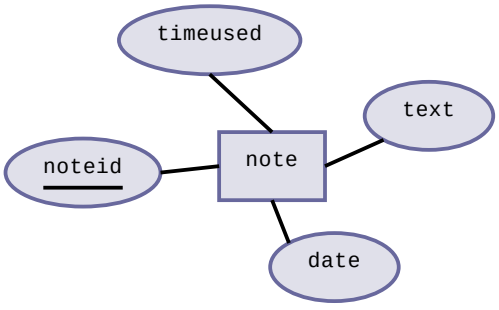


Figure 4.15: Note ER model.

As shown in figure 4.15, the *note* table contains a unique noteid, the date of creation, the text of the note and the amount of time used fixing the task so far.

Table relations

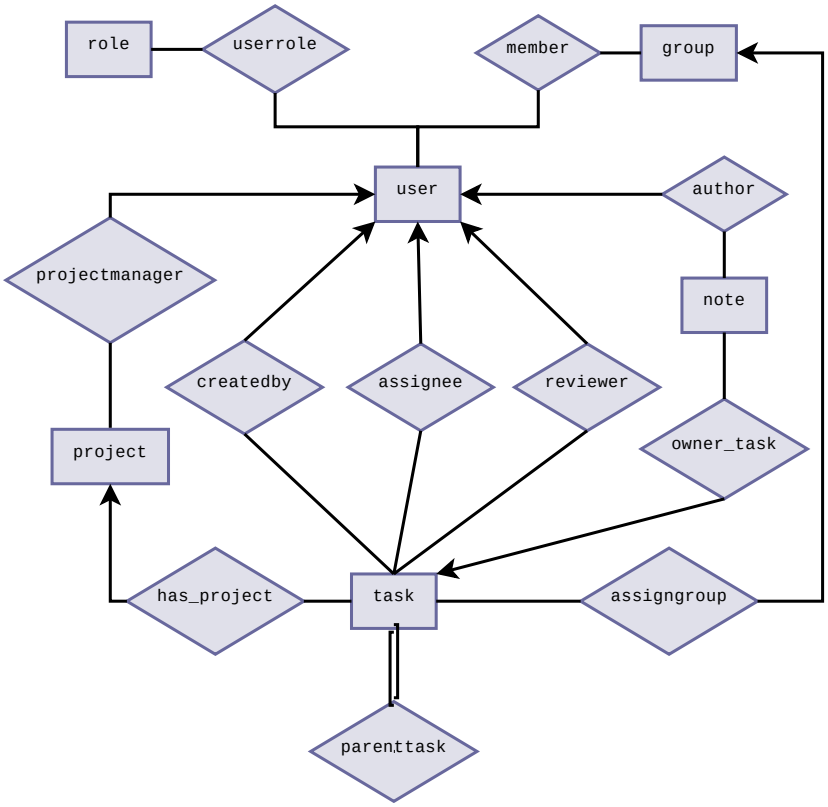


Figure 4.16: ER model.

The table relations is shown in figure 4.16. The *role* table has a many-to-many relation to the user table through the *userrole* table. The same thing is done with the *group* table through the member table. The *project* table has a relation to the *user* table which contains the manager of the project. Further it has a relation to the *task* table to identify which projects is containing the task. The *note* table has a relation to the *user* table, which author of the note and has a relation to the *task* table to the task the note is written for. The *task* table has three relations to the *user* table which contains the user who created the task, the user who is assigned the task and the reviewer. It also has a relation to the *group* table which contains the group the task is assigned to (if the task has assigned it to a group and not a specific user). Further it has a relation to itself. This contains a parent task if it is a subtask.

4.7 Product Description

Access to the product will be provided by a web server and the users will connect to it via the Internet. The user is redirected to the dashboard after logging in. On the left side of the page there will be a navigation menu and a text field with a button for jumping directly to a task using the id. At the top of the page there will be a project selection drop down list and the logout button. All pages will have these elements.

Dashboard

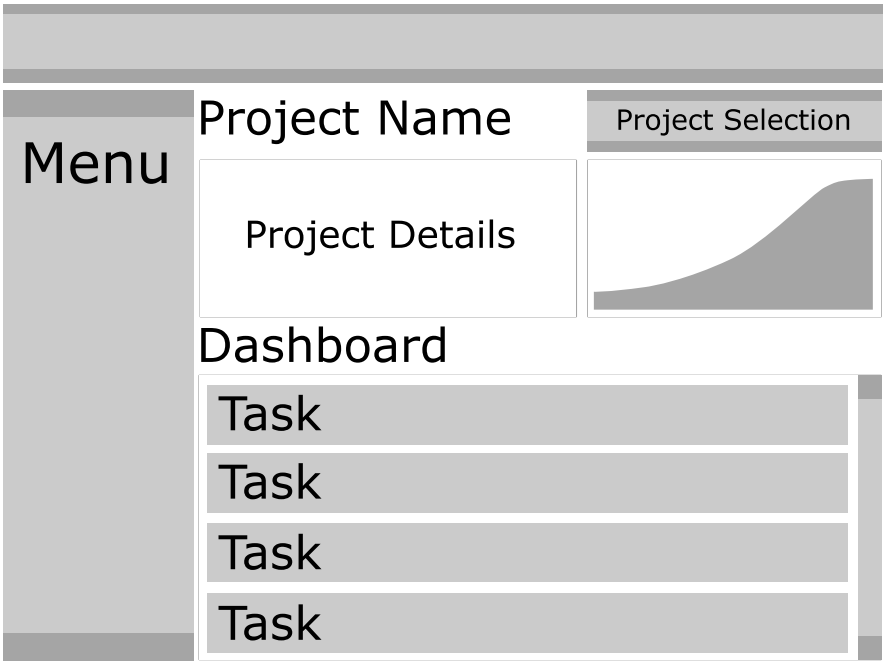


Figure 4.17: Dashboard draft.

This page will be available to all users. It is shown in figure 4.17. The dashboard will show information about the selected project, a graph indicating progress and the tasks of the project. The user can use a drop down list to filter what tasks are shown, for instance the user can filter by the task status. The user can click "view" on a task to see details about it. The task view page will show information about a task, mainly a description and notes. On this page the user can add notes to a task. Notes are used to enter work hours used working on the task. It will also be possible to archive tasks, which will not delete the task, but hide them from the dashboard.

Coloring of tasks

The overview will colorize the task according to its status. The colors will be white, light yellow, yellow, pink, red and green:

- White
  A white background color on a task shows that it is unassigned.

- Light Yellow
  A light yellow background color on a task shows that it is assigned.

- Yellow
  A yellow background color on a task shows that the developer has begun working on it.

- Pink
  A pink background color on a task shows that it is close to overdue.

- Red
  A red background color on a task shows that it is overdue.

- Green
  A green background color on a task shows that it is done.

User Profiles

Available to all users. User Profiles links to a profile page for users. It will show information about users, including roles and competences. It will be used to identify other developers and keep track of contact information, for instance email addresses.

History

Available to all users. The History page will be used to view the system log. The user can see entries from the log file and filter them after their level. This can be used to see who did what and when, and is also useful for debugging.

Calendar

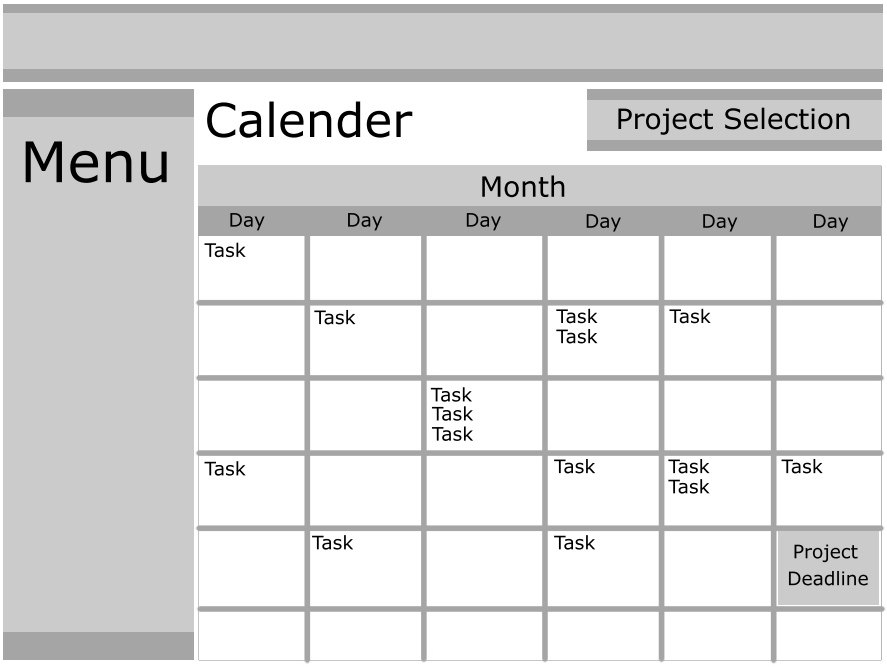


Figure 4.18: Calendar draft.

The calendar will be available to all users. It is shown in figure 4.18. This page will show an interactive calendar, where tasks show up on the day of their deadlines. Users can easily get an overview of the distribution of tasks among the days of the week. This will make scheduling of work and planning future development easier for project managers. The calendar can be filtered the same way as the dashboard. Tasks will show up in different colors according to their status as described above in section 4.7 on the facing page. The calendar will also show project deadlines and these days will be emphasized on the calendar. This way it will be easy to see if tasks are overdue and if it is required to take action.

Task Creation

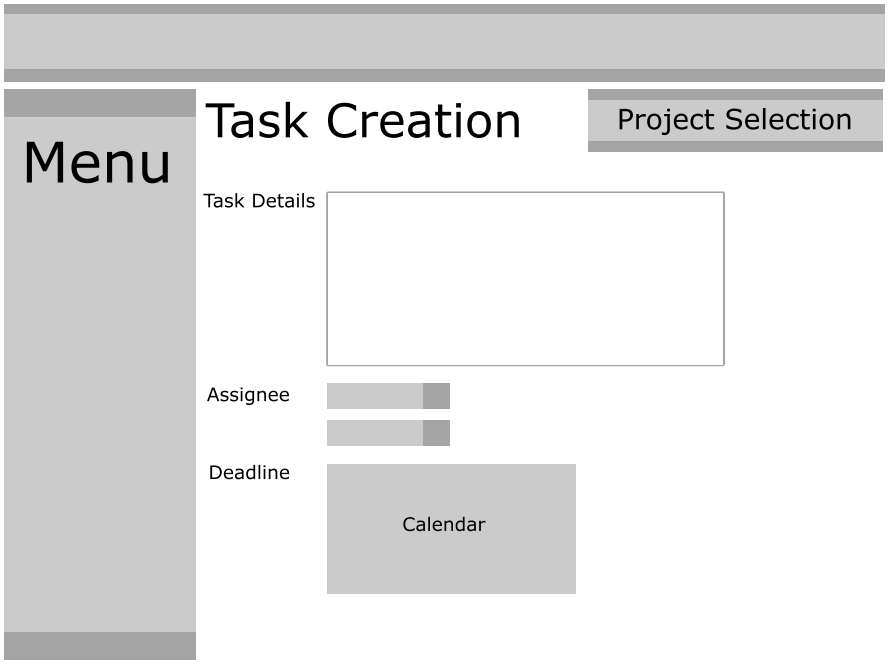


Figure 4.19: Task Creation draft.

Available to all users. It is shown in figure 4.19. The task creation page will be used to create new tasks. Tasks will be created in the project selected in the drop down list at the top of the page. Here the user will enter details for the new task and select who it will be assigned to. The user can also choose to offer it to a competency group so the developers can assign it themselves later. The estimate entered will be a qualified guess.

Task Editing Page

The following will be available to project managers and administrators. When viewing a task, users can click the "edit" link. The edit task page allows the user to edit everything about a task, for instance assign it to another developer or change the description to be more sufficient.

Group Administration

This will only be available to administrators. Here it will be possible to create new user groups and edit existing ones. Administrators will use this page to create a better overview of who has which competencies.

User Administration

The user administration is available to administrators and allows editing and creating users. Here, the administrators can change a users roles and select what groups the users belongs to. The administrator can also archive users here. When users are achieved, they can no longer log in to the system.

Project Administration

The project administration page allows administrators to see all current projects and create new ones. When creating a new project, a user with the project manager role can be assigned to the project as project manager.

Source Control

All users can view the source control page. Monitoring of a source control system will be incorporated in this page. The integration will provide features such as:

- Actions in commit message
- Note in commit message
- Record files associated
- Record the user

It should be possible to work on any number of files, with a commit message like "Worked on task #1234 and there still seems to be problems". This will then add a note on task #1234 with the commit text and a listing of all the files affected. A link could be provided, to a revision difference view. The note should be added with the correct user (mapped from the source control username). It should also be possible to close a task from the commit message.

Documentation

The documentation page is available to all users. A documentation page will allow developers to view documentation for different parts of a project. This can be documentation about many different things, for example on the project manager itself or some of the projects currently being developed. It will be possible to show both auto generated documentation like doxygen, and other documents. There will be a drop down list where it is possible to select projects. Users can add new documentation and edit existing entries using different forms. The page will help keep all documentation centralized and make sure everyone has access to it at all times.

Estimation Algorithm

An algorithm to calculate estimated work hours for a task will be developed. The calculation will be based on some of the same factors a human mind would use. When a task is created it will be assigned to a certain group. Members of a competency group may have different levels of skill in that competency, which will be one factor in the calculation. Another factor is the size of the task, which will be represented by a number. Finally the algorithm will check the precision of earlier estimates to optimize new estimates.

Test and Review

All users will have access to this page which will show results from unit testing. The written unit tests will run automatically with a certain frequency and if the tests passes, this will be shown on the page. If all or some tests fail, warnings will show up on this page. Furthermore emails with the warnings will be sent out to designated receivers. The list of people who will receive the warning emails can be edited on this page. Rules that allows sending of only a certain test-result to a user can also be setup here.

Bug Reports

When creating a new tasks there will be an option to flag the task as a bug report. Instead of new work that needs to be done, it will describe a bug in the system that needs to be fixed. When the bug flag is checked it will be possible to select severity level of the bug.

Chapter 5

Scientific Methods

5.1 Development Methods

During the initial phases of the project we discussed different models for software development. Sources for this section are the slides from SWP (Software Processes) course with Peter Axel Nielsen [13].

Waterfall model

In this model the developers work on one phase at the time. The phases are: Requirements specification, Design, Implementation, Verification, Maintenance. When a phase is done the developers continues on to the next phase and regard the previous phase as done. The waterfall model is an advantage in projects with fixed deadlines and a non-changing specification.

Reuse model

The reuse model is based upon the use of existing components. It is required to have access to a base of components. These components will be combined and integrated with the new system.

Evolution model

This model allows changes in system design during the development, unlike the waterfall method. The model has an iterational. For each iteration, developers return to the certain phases, where they can choose to alter content. The only thing that does not change is the requirements.

Incremental model

The incremental model has many properties in common with the evolution model, in the sense that it has some iterative aspects. This model uses the same requirements throughout the process. The requirements are divided and prioritized. Each incremental phase starts with a design phase and then the increment is programmed. After the programming the increment is tested. If it passes the tests the increment is implemented and the process starts over again with a new increment.

Our choice

Seeing that we do not have any components we could reuse, the reuse model is not suitable for our project. In theory we could recycle existing commercial and open source components and we would learn something doing so. We feel that the knowledge we would gain from writing the components ourselves is more important at our current educational state.

An important thing to consider is that we have static requirements in the project, but even though these requirements have been distributed to us, we do not feel that the waterfall method is suitable for this project. The requirement, that have been assigned to us, are not detailed enough for us to completely avoid redoing parts of our architecture. We cannot rewrite the requirements during the development, but maybe we can interpret them differently as we go along, if it is appropriate.
For these reasons we consider the evolution model to be the most convenient model in our case.

5.2 Coding Standard

A coding standard is a way for developers to insure that we develop code that is easy to read and understand.
The code standard is used to make it consistence and thereby making it easier to read. Based on our knowledge and experience we made the following rules when creating code for the system.

1. All variable-, method- and class names must be as logical as possible.
2. Methods that are complex should have one-line comments at the complex parts of the code to explain it.
3. When making changes to the system all references must be checked so system crashes can be avoided.
4. Classes should be structured such that fields and properties are declared first then constructors afterwards and at last all member methods.

We will follow these simple rules as we develop our program.

5.3 Review And Feedback

To insure quality drafts of code and the report we have a choice between two methods of checking. Pair Programming and Peer Review Programming in the following section we will present the two methods and explain how we made our choice.

Pair Programming

As the name states, this programming technique dictates two developers working together at writing code. The two developers switches between two roles frequently. One developer writes the code while the other is free to

think further and set things in perspective. This observer can associate the code with the overall design while the code is being written.

All in all this technique favors quality over quantity. It takes longer to develop code but the quality of the code will evidently be of a higher quality. The amount of bugs, for example, is reduced because if one developer misses something, then there is a chance the other will notice it.

Studies show that pair programming finds 40-60% defects in code, before testing starts[5].
The developers can use more time on decision making and discussions than a single developer could, and in many cases make more intelligent and well-founded decisions.

Peer Review Programming

Peer Review Programming is similar to Pair Programming, but instead of observing the code while it is being written, the code is examined later. The observer examines the code with a new point of view. The observer might discover new bugs and comes up with suggestions for improving the code for example optimizing of code.

There are two forms of review: Formal and informal. Formal reviews are scheduled meetings where people meet prepared and review the code in question and discuss it. Informal reviews are not scheduled, the developer mails or talks with a coworker to review the code for each other.
Studies have shown, that formal code review finds 45-70% of defects in the code, and informal review finds 20-40%.
It would seem that using these techniques would extend development time, but the opposite is actually true, studies at IBM has shown that each hour used in formal review saved 100 hours of work afterwards used on testing and debugging[5].

The advantages of these methods is like Pair Programming where quality is enhanced and bugs reduced. Review is more flexible than Pair Programming because the two developers is not required to work at the same task at the same time, and informal even more, because there are no scheduling involved.

Our Review choice

As Steve McConnel writes: “As pair programming and formal inspections produce similar results for quality, cost, and schedule, the choice between

them becomes a matter of personal style rather than one of technical substance"[5].

Based on this, and the fact that we already used pair programming in earlier semesters, is an ideal situation to try the review concepts.

Chapter

6

Choice of Tools

6.1 Introduction

In this chapter we will describe the different tools we could use for developing this program. Each tool is produced by a number of suppliers so we pick a few of them out and describe them. At the end of each section we will discuss and conclude which kind of tool we will use.

6.2 Programming Language

When starting a project like this, one must carefully choose a programming language. The choices that we looked at, were the following, based on that we have learned Java as a part of the education last semester, and we have a C# course this semester [15].

Java

The Java programming language was developed in 1995 by Sun Microsystems, as a fully object orientated language. The language has C++ like syntax, but has fewer low-level facilities. Java compiles into byte code, which enables it to be executed on large range of operating systems by the so-called Java Virtual Machines (JVM). The Java philosophy is "Write once, run anywhere". In 2007 Sun made their Java technologies open source under the GNU General Public License [14].

C#

C# is developed by Microsoft as part of the .NET initiative and later approved as a standard by ECMA (ECMA-334) and ISO (ISO/IEC 23270). C# is an object oriented language, and is highly influenced by the Java programming language. C# is a new language, its first release were in 2000. C# compiles into byte code too, so this enables it to run on other platforms than Microsoft Windows, if someone writes an interpreter for that platform. Currently an open source project exists, called Mono, to run C# programs and byte code on other platforms than Microsoft Windows [15].

PHP

PHP: Hypertext Preprocessor is a scripting language, originally designed for producing dynamic web pages. It has evolved to include a command line interface capability and can be used in standalone graphical applications. It generally runs on a web server, taking PHP code as its input and creating web pages as output. It can be deployed on most web servers and on almost every operating system and platform [19].

Summary

It is a factor that we have had Java and C# as semester courses. The choice between Java and C# are close, and functionality are almost equal, so the decision would have been influenced by other factors than the language itself, for instance a company's existing developers knowledge, current platform, servers and licenses and the like, more that the language itself. Because we have already used Java in earlier semesters, we would gain more by using C#.

6.3 .NET

When the choice has been made to write the software in the C# language, next one has to choose which .NET compiler and CLR (Common Language Runtime) to use. Currently there are several choices; Microsoft .NET, Mono (by Novell), DotGNU and portable.NET. We will only discuss the first two, as they are the two most significant.

Microsoft .NET

Microsoft .NET [6] is available for several Microsoft Windows operating systems. It includes a virtual machine to execute the programs written in the supported languages. Microsofts CLR shields the developer from making decisions regarding the CPU architecture that the software will run on, the CLR will handle this.

Mono

Mono is a set of .NET compatible tools, including C# compiler and CLR, the project is led by Novell. In contrast to Microsoft .NET, Mono can run on several platforms, such as Linux, BSD, OS X, Solaris, Windows and others. In the current release of Mono, it supports C# 3.0, LINQ and Winforms. Furthermore it supports Gtk# 2.0, which is a graphical library for GTK, like Winforms is for Windows [16].

Summary

We have chosen to use Mono for this project, as we would like the system to be platform independent. Mono supports many new features of the .NET framework, and it comes with a ASP.NET compatible webserver. MonoDevelop has features to compile and start the project directly without further configuration.

6.4 Database

To store our persistent data, we have chosen to use a rational database. One reason for this is that we have a course this semester that covers the subject of rational databases. We will discuss some of the candidates here:

Microsoft SQL Server

SQL Server is a RDBMS (Relational Database Management System) produced by Microsoft, originally based on Sybase SQL Server. SQL Server's primary query language is Transact-SQL (T-SQL), a proprietary extension to the ANSI SQL. SQL Server is a good choice, by the reason that it is directly and well supported in C# as Microsoft is the company behind both products. A drawback is, that SQL Server only runs on Microsoft Windows. SQL Server will be a good choice together with Microsoft Visual Studio, Microsoft .NET and Microsoft Windows, as they all work well together in Visual Studio [7].

MySQL

MySQL is one of the leading open source and free RDBMS. MySQL is actively developed by a large community, and not least the Swedish MySQL AB company. MySQL utilizes the ANSI SQL query language, with some MySQL specific additions. C# can communicate with MySQL through the ODBC connector. MySQL runs on a many platforms, including Microsoft Windows, Linux, Apple OS X and many others. MySQL will be a good choice together with a Linux operating system and Mono .NET runtime. However it can be used together with Microsoft .NET and Visual Studio too [17].

Summary

The choice is between SQL server and MySQL, the difference between the two, is that MySQL runs on most operating systems unlike SQL server that only runs on Microsoft Windows. After a discussion we chose MySQL. This choice enables us to run our system on most operating systems.

6.5 Nunit

The source of the following is [3].

Nunit is a unit test framework for testing the .Net-platform. Unit testing is used to check if the code works as intended. Unit testing ensures that later changes to code will preserve the intended functionality. Each class in

the software has its own test class usually named as "classname"+Tests. A test class tests every method in that class. In this way one can easily point out which method has an error if the test fail. Unit testing runs a setup before performing a test. After the test, regardless of the result, it runs a tear down method. This makes the test environment exactly the same for each test in a test class. Every test class is located in a test suite, that keeps track of which test classes that should be executed at each test run.

Unit test is a main part of the Test-Driven-Development (TDD) method. In TDD one key rule exists: "Test twice, code once" [3]. The rule covers the three steps in TDD:

- Start with writing a test
- Write the simplest code
- Watch the test pass

You start by writing a test for the code you are planning to write. When your test is done, you write the simplest code that satisfies your task. When the unit test succeeds, your work is done and you can move on to the next task.

Mockobject

When coding unit test in TDD, all tests are written before the classes. This means that when writing the test you use mock objects to simulate other classes that is not written yet. Mock objects uses an interface to simulate the class. By using an interface one can then write what the class returns from the different methods.

6.6 ASP.NET

The web programming platform ASP.NET (Active Server Pages with .NET-technology) is developed by Microsoft[8] as the successor of ASP (Active Server Pages, also called Classic ASP). The first release of the .NET framework was released in January 2002 with the version 1.0. ASP.NET uses the CLR which allows the developer to write ASP.NET in any .NET language, such as Visual Basic, C# or Java. ASP.NET has the ability to separate the web application from the presentation layer of the web page, by using a model called code-behind. ASP.NET has a concept of MasterPages to keep layout of pages and common elements.

Chapter

7

Product Development

7.1 Introduction

At this point we have done sufficient research and analysis to begin working on the product. In these following sections we will elaborate on the different challenges we faced in our development phase. We will further document the system in all its details.

7.2 Classes

This sections will describe solutions to some of the problems we faced in the development phase. We describe the solutions from design to implementation and substantiate our choices during development. We will not cover the implementation of all the classes in the project, but we will only describe a few that we see as the most interesting and of great importance.

Database

In this section we will describe our design and the implementation of our database abstraction layer.

Why do we need an abstraction layer? The key feature here, is that we are able to change the underlaying database connector, or the whole database vendor for that matter. We have made our database abstraction using the Singleton design pattern (see section 4.3 on page 18) to keep identical resources to a single instance. There is no need for more than one connection to the database, it is much faster to reuse the connection rather than opening a new connection for each statement needed throughout our system [12].

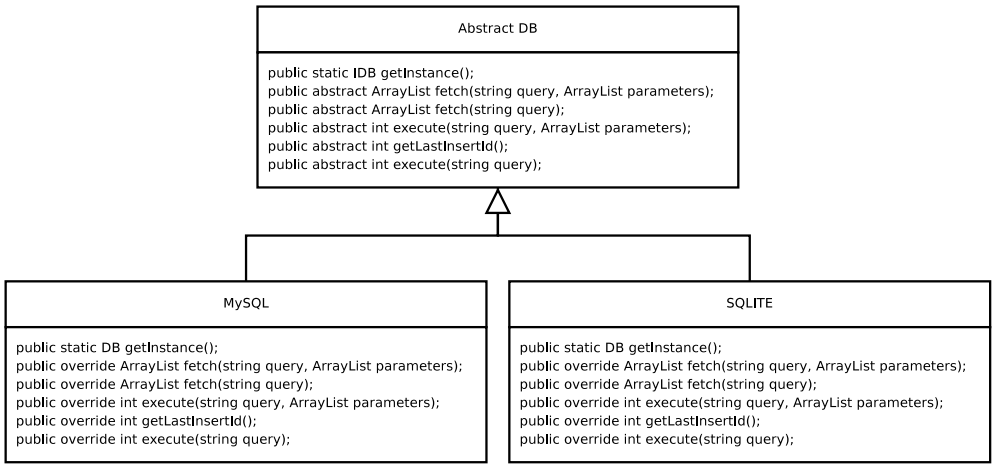


Figure 7.1: Database API Diagram.

Using this hierarchy (figure 7.1), we can instantiate a instance of DB and

get the relevant underlaying type MySQL or SQLite depending on how the system is running, MySQL for production and quality testing, and then In-Memory SQLite for unit testing. This design gives us full control over the connections to the database, only a single instance at all times and the capability to switch between database vendors. With this design we can add other database vendors without editing the application, as with the case when we added support for In-Memory SQLite database for unit testing.

This scheme may not be sufficient for scaling the system up to a high number of concurrent users, if the system will be used in such situation, we can add support for Connection Pooling using the connection pooling capabilities of the database layer in .NET [12].

The Logger Class

As the name implies the logger class is used to log system events in a text file. It is the same principle as the black box of an aircraft. The system log has multiple purposes. It is used to provide a trail of information about the systems activity so that it can be analyzed. We mainly use the system log for diagnosing the system when it fails, but we also use it for tracking project management related events, such as task and project creations, alterations, etc.
As the system runs and the log increases in size, analyzing the system log may become confusing. To categorize our log we have chosen to introduce log levels as seen in table 7.1. Our log levels are inspired by the apache web servers logging. We use four levels in our system.

Level	**Name**	**Use**
1	ERROR	For critical errors, for instance errors that crash the system.
2	WARN	For warnings and non critical errors.
3	INFO	For information statements, for instance if a task is altered
4	DEBUG	For anything that the programmer may need to know when debugging, for instance SQL statements.

Table 7.1: Our Log Levels.

The system current log level is specified in the web.config file. Here the level can be reduced and increased. If the system log level for instance were specified to DEBUG, all log statement would be written to the log, however if the log level were reduced to WARN, only WARN and ERROR statements would be written to the log. It is designed this way, because once the system is running in production environment, a certain amount of the logging

information may no longer be interesting and for performance reasons.

The logger class uses the singleton design pattern to ensure that different system events are never logged at exactly at the same time and to avoid conflicts when working with files.

A log typical log statement may look as source code 7.1:

```
1 Logger.Instance.log(Logger.Levels.INFO,
2         username.Text + " has logged in");
```

Source Code 7.1: Log statement.

7.3 Performance Issues

At some point in the development phase, we encountered a critical performance issue, where the system performed so slow that it were nearly unusable. We have set up an investigation group to find the issue.

By using the logger it became clear that there were a problem with our database connection, and that we in fact did heavily stress the database with useless query statements.

We queried the database for alot of equal statements, in fact we made more than 690 SQL queries to the database just to show the dashboard.

The reason that we had all these statements where actually caused by object creation. To illustrate the problem, we made a graph (see figure 7.2 on the facing page).

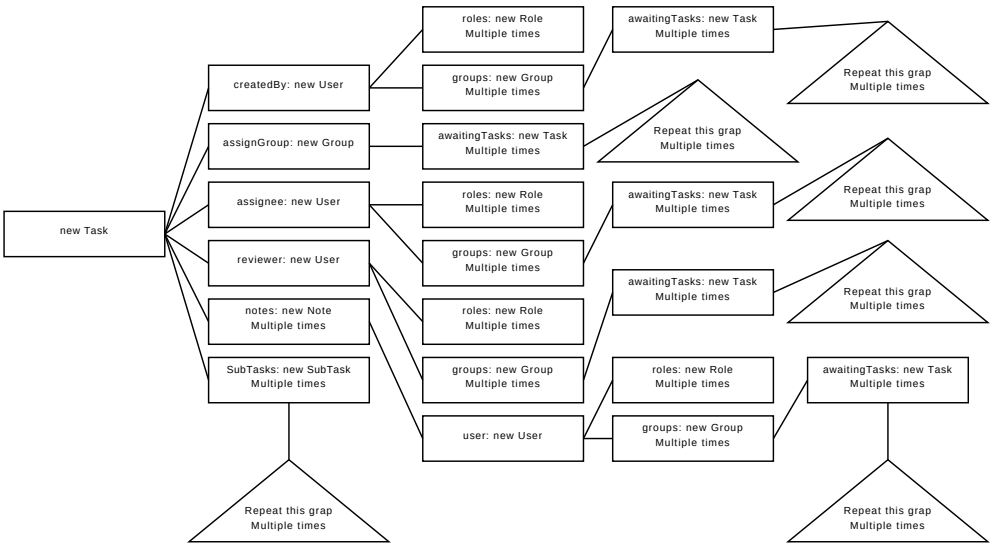


Figure 7.2: Graph of object creation.

The graph is presented in full size in appendix C on page 85
When loading the dashboard, far too many objects are created, as indicated by the graph on figure 7.2. This causes two major problems: First, there are many SQL queries to the database that fetches the same data multiple times, and secondly, the vast consumption of memory, to store identical objects.

When we figured that this was the source of our performance problem, the solution was straight forward. We moved object creation to our controller classes, such that instead of writing:

```
1 Class c = new Class(id);
```

Source Code 7.2: Caching Example 1.

We would use the controller class, like this:

```
1 Class c = ClassController.get(id);
```

Source Code 7.3: Caching Example 2.

The implementation of the simple caching solution in the static controller class consists of a simple `Dictionary` collection, where we can check for the ID, and return a reference to the already created object. If the object is not found, we can create it, save it in the cache and return a reference.

7.4 Software Testing

The following section is about how we used the test framework Nunit, Mock object and user tests. It will explain why we wrote our code before we wrote our tests and how we tested our system.

Our Use of NUnit

In this project we have used nunit for testing our code. We originally planned to write unit tests before we wrote the code, but we ran into some problems. The first problem we faced was concerning the database. Running unit tests on a production database would cause consistency problems, because we need to separate our test data from the live data. To overcome this problem we used abstraction by making a general database class and then have two classes that inherits from the general class. This did overcome the problem concerning the production database, but how do we switch between the databases? We solved this by making a static class that was accessible from any file or class. In this class we had a boolean value that decided if the program was running in production mode or not. This way we had the possibility to switch between databases. The next problem we discovered was, which database we should use as the test database. We wanted a local database so that the unit test did not stress the production database server. We had experience using a SQLite database in a previous project and combined with the possibility to use it in-memory, we chose to use SQLite again. This was very time consuming to implement, because of some SQLite specific SQL-statements. SQLite could not interpret our SQL-dump from the production database directly. We rewrote the SQL-dump so SQLite could interpret it and we were then able to create a SQLite database. The next problem occurred, when we began to insert data into the database. The auto increment did not work and the fault was that instead of:

```
1 taskid int(11) NOT NULL PRIMARY KEY AUTOINCREMENT
```

SQL Statement 7.4: NUnit Example 1.

we should use:

```
1 taskid INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT
```

SQL Statement 7.5: NUnit Example 2.

As it clearly states above, SQLite did not act as intended. That aside another problem occurred when reading from the database. This time the error was related to the DateTime fields. We found that SQLite could not understand the MySQL function `NOW()`, so to overcome this problem we did the following in the `SQLite` class:

```
1 dbcmd.CommandText = dbcmd.CommandText.Replace("NOW()",
2         "DATETIME('NOW')");
```

Source Code 7.6: NUnit Example 3.

SQLite uses the `DATETIME` function and by replacing `NOW()` with `DATETIME('NOW')` SQLite now finally worked as we wanted it to. Dealing with these problems while developing caused the unit tests to be written after the programming phase. Nonetheless the tests revealed some minor errors in the code, so it was not completely useless to write the unit tests.

User Tests

During the development of the product, we used user tests to see if newly written code was working. The developer who tested was always the same as the one who wrote the functionality that was being tested. In this way the developer could easily find a bug and correct it quickly. When the developer had successfully completed a task and tested it, he committed it to subversion.

7.5 Implementing MVC

The way we have chosen to solve a handful of problems is with the use of the MVC design pattern. In the following sections we will use handling of projects as an example of implementing MVC in our system.

Model

Our data models are separated in two parts. One part is the persistent data contained in the database and the other is the in memory data contained in the data classes when the system is running. Our data classes are the following: `GeneralTask`(abstract), `Group`, `Note`, `Project`, `Role`, `Subtask`, `Task`, `User`. Each of these classes has a number of Properties to set and get the private or protected variables within them. The data classes also possess constructors which we use to instantiate data objects with the persistent data from the database as seen in source code 7.7 on the next page.

```
public Project(int _id)
{
        this.id = _id;
        ArrayList param = new ArrayList();
        param.Add(new SqlParameter("id", this.id));
        string query = "SELECT * FROM projects WHERE projectid = @id";
        IDB db = IDB.getInstance();
        ArrayList data = db.fetch(query, param);
        Dictionary<string, object> d =
                (Dictionary<string, object>)data[0];
        this.name = d["name"].ToString();
        this.projectmanager = UserController.getUser(
                Convert.ToInt32(d["projectmanager"]));
        this.description = d["description"].ToString();
        this.deadline = Convert.ToDateTime(d["deadline"]);
        this.created = Convert.ToDateTime(d["created"]);
        this.status = Convert.ToInt32(d["status"]);
}
```

Source Code 7.7: Project Constructor.

Controller

Each data class has a controller class associated with it, so for instance Project is controlled by ProjectController. The controller classes contain a number of methods for handling data from the data classes. Since the controller classes only contain these methods, they are all static. This method is used to retrieve a list of projects contained in the database as seen in source code 7.8.

```
public static ArrayList getAllProjects()
{
        ArrayList projects = new ArrayList();
        string query =
                "SELECT projectid FROM projects ORDER BY deadline";
        IDB db = IDB.getInstance();
        ArrayList data = db.fetch(query);
        foreach (Dictionary<string, object> d in data)
        {
                int id = Convert.ToInt32(d["projectid"]);
                projects.Add(ProjectController.getProject(id));
        }
        return projects;
}
```

Source Code 7.8: ProjectController.getAllTasks().

Note that `getProject` returns a Project object with a given id.

View

The way data is viewed and altered in our system is through the web interface. We use ASP.NET's object data source to exchange data between the database and the web interface. In the example in source code 7.9 we have made an object data source that uses the `getAllproject` as its select method. It is located in the ProjectController class.

```
1 <asp:ObjectDataSource ID="ODSP" runat="server" SelectMethod="
      getAllProjects" TypeName="ProjectController" />
```

Source Code 7.9: Project Object Data Source.

We can then extract the projects from the the object data source to a drop down list. When this drop down list is clicked, the user will be able to select between the project names from the database. Once the user selects a project name, the `OnSelectedIndexChanged` method is called, which in the example in source code 7.10 is `onChange`.

```
1 <asp:DropDownList id="project" runat="server" OnSelectedIndexChanged="
      onChange" AutoPostBack="True" DataSourceID="ODSP" DataTextField="
      Name" DataValueField="Id" />
```

Source Code 7.10: DropDownList of projects.

The `onChange` method is located in the asp.net file's C# code behind file as seen in source code 7.11. Once the `onChange` method is called upon, the session project is changed to the project selected from the drop down list.

```
1 public void onChange(object sender, EventArgs e)
2 {
3         Session["project"] = project.SelectedValue;
4 }
```

Source Code 7.11: onChange method.

Figure 7.3 on the following page shows a sequence diagram of how the project drop down list works through the system.

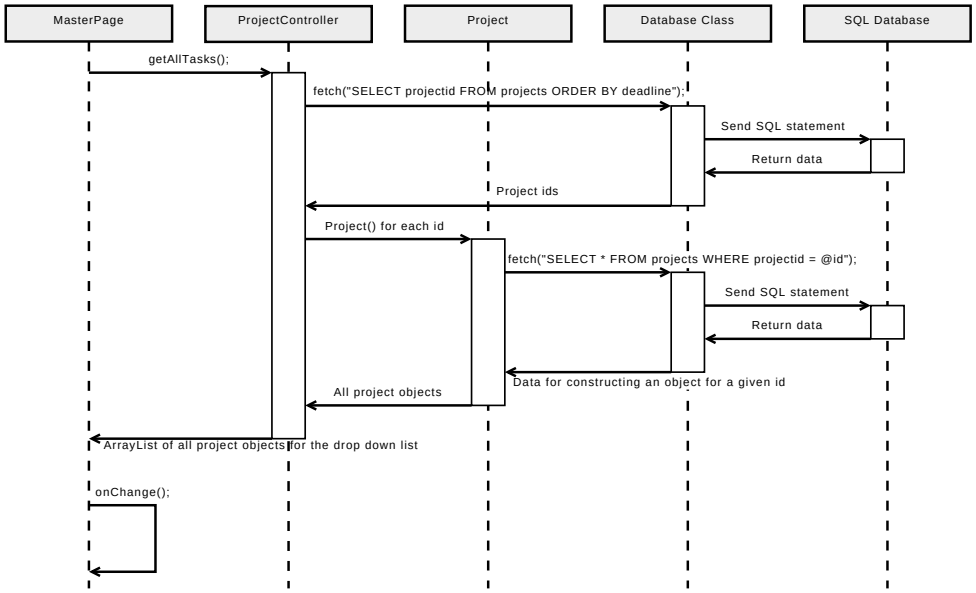


Figure 7.3: Project sequence diagram.

7.6 Source Control Integration

Subversion offers several hooks, that can be used to integrate with other systems. One of those hooks are called the "post commit"-hook, that is a script that will be called after each commit to the subversion repository. This can we used to send information to our *ProjectTracker* system. We created a Bash script as shown in source code 7.12.

```
1 #!bin/bash
2 REPOS="$1"
3 REV="$2"
4 auth=$(svnlook author -r $REV $REPOS)
5 changed=$(svnlook changed -r $REV $REPOS)
6 log=$(svnlook log -r $REV $REPOS)
7 log=${log//\#/%23}
8
9 wget -q --spider http://host/svn.aspx?revision=$REV&author=$auth&log=
      $log&changed=$changed
```

Bash Script 7.12: Post Hook Script.

This sends the information to the web system using simple GET parameters.

To receive the information on the web server, we have a script (`svn.aspx`) to process the subversion commit statements, to find relevant commits, as

seen in source code 7.13.

```
if(Regex.IsMatch(log, "#[0-9]+"))
{
User user = UserController.getUserByUsername(author);
string taskid =
        ((Match)Regex.Match(log, "#([0-9]+)")).Groups[1].ToString();
bool isDone = Regex.IsMatch(log, "done|finished");

Int32 timeUsedHours = 0;
Int32 timeUsedMinutes = 0;

Match m = Regex.Match(log, "([0-9]+):([0-9]+)");
if(m.Success){
        timeUsedHours = Convert.ToInt32(m.Groups[1].ToString());
        timeUsedMinutes = Convert.ToInt32(m.Groups[2].ToString());
}

string logmsg =
        "Revision: " + revision + " \n\n " + log + " \n\n " + changed;
NoteController.createNote(taskid, logmsg,
        (timeUsedHours*60)+timeUsedMinutes, user);

if(isDone)
{
        Task t = TaskController.getTask(taskid);
        t.Status = GeneralTask.StatusCode.Done;
        TaskController.update(t);
}
}
```

Source Code 7.13: SVN Commit Reciever.

This makes it possible to add notes to a task with a message (the same message as in the commit statement) and furthermore it allows developers to add time used on the task, and even close the task if the developer is done working on it.
It is assumed that the subversion username and *ProjectTracker*username are identical.

7.7 System Development Method

In the beginning of the project we discussed different methods and tools that we would use in the software development process. Based on what we chose this section of the report is about how we used these methods and tools in the process.

Development Method

The development method that we chose to use was the evolution model. This was based on the ability to change the design in every iteration. So we were not dependent on a specific system design. We worked individually developing the code on our local PCs connecting to a central database. The code was frequently synchronized using a SVN server. This enabled us to work on different things independently and still share the same raw data in the database. Since we did not have a detailed plan of the iterations, we did not get the full benefit from the development method. Instead we determined some goals that should be done at a specific date.

Most of our programming took place in the group room so whenever we had an issue or wanted to edit a file we asked if anyone was using that given file. This method of coding enabled us to program continuously, only taking breaks from the programming when the design changed. By doing this we automatically made user tests before uploading our changes to the subversion system, so we were sure that what we committed was working and did not crash the system.

Based on this we have made a simple example shown on figure 7.4, which visualizes how our programming process took place.

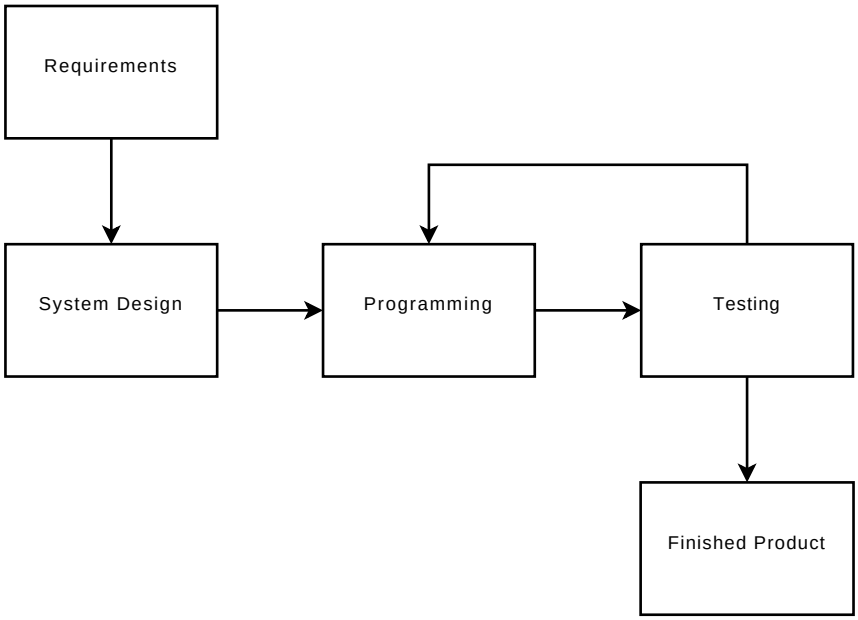


Figure 7.4: Software Development Method.

Review and Feedback

We have used the review and feedback method called formal review. We held five review meetings during the development, where the developers had studied code in advance. Discussions were made about the code to improve quality and consistency in the structure. We tried to keep the discussions at a professional level to ensure objectivity and allow the developers to express their opinion.

7.8 Our Use of *ProjectTracker*

In this section we will discuss our experiences using *ProjectTracker*. We will also explain how we used our system in the project development. We aimed to begin using our system as soon as we could. This required our program to have some of the core functionalities done. Focusing on the core elements of the system first we would have a simple management system up and running fairly fast. That is why we created functionality to create and edit projects and tasks first. The second thing we wrote was a user management that would allow us to create and edit users. With these basic functionalities we could start creating a user account for the group members and start creating tasks. Realizing we could use the Project Manager for our project report as well, we created two separate projects in our system. One for *ProjectTracker* itself and one for our project report.
In the project we began adding tasks for other functionalities of the program such as notes, work hours management and competencies. The report project was used to distribute writing tasks among the group members. We would find something to do for each member, then add tasks and assign them respectively.

Chapter 8

Reflection

8.1 Introduction

In this chapter we reflect on some major themes of our development process. We use the experience we have gained by developing the system to consider the advantages and disadvantages of our actions, choices of tools and methods. This chapter also describes the group's use of the system during the development. When considering the data we entered into *ProjectTracker*, what turned out useful and what did not?

8.2 Own Experiences

We found that it required some discipline to keep *ProjectTracker* up to date. When work started or ended on a task it was not always entered in the system immediately and work hours spent were rarely kept track of. This caused our data and estimates not to be completely accurate at all times. This might have something to do with the way we handle work hours. In our system we had to enter hours every time we had worked on a task and the only thing we would get in return was a rough estimate on how far we were in the progress. Because our estimated work hours was nothing but qualified guesses, our progress estimation was not very useful. Maybe we should have set up a number of work hours available for each developer and compare that with the load of tasks assigned to a developer. Then the entering of work hours after each day could be used to correct the work hours available ongoing. Finding ourselves not using our own system we came to some conclusions about a project management system in general. A simple system is required if a project manager should be useful. If a developer have to use a project management system every day it has to take up as little as his time as possible. It should take no more than a few minutes to get a overview over the tasks a developer is supposed to work on. There should be consistency between the effort put into the project manager and the useful output of the manager. If we should use our *ProjectTracker* in a future project we should first of all make sure that everyone uses it with discipline. Maybe some of the features should be revised to make it faster and easier to use. Usability could actually play a big role here, but because usability was down prioritized in our project,
ProjectTracker is not very user friendly and a bit crude. If it is complex and hard to record the work at the end of the day, it will simply not be done. That is why we think usability should actually have a higher priority in a project manager.

8.3 Software Testing

In this section we reflect on our use of unit testing during the development of our system. We explain our speculations on how things might have been different if we had executed unit testing otherwise.

NUnit

As mentioned earlier in section 7.4 on page 50 we had used NUnit tests in this project. We had many problems to get it working properly, so the Unit tests were written late in the development phase.

We experienced how editing code in one place could introduce bugs in another place. This happened several times this semester. If we had started getting NUnit working before writing code, we could have saved time on debugging. Simple unit tests would have caught some of these bugs.

Even though we had so many problems setting up NUnit we think that unit tests is a convenient tool for a software developing team. Developers do not have to worry about breaking existing code, because the unit tests should react if something is wrong. Considering this we should try using the Test Driven Development method for maximizing the benefits of unit tests in a future project.

Mock Objects

In our implementation of NUnit we did not use Mock-objects, at least not in the ideal manner. We used another approach to simulate a Mock database. If we had to start over on this project we definitely would think differently. We would have to think in interfaces. To use Mock-objects it is required to write interfaces for each class. When having an interface it is possible to create a Mock-object from that interface and determine what the functions should return. This way, the code that is being tested is isolated, which means that if an error occurs, it is easy to locate.

8.4 Open Source vs. Microsoft

We have chosen to work with the open source implementation of the .NET Framework called Mono, this includes working with

- Mono (.NET Framework and Compiler).
- MonoDevelop (Integrated Development Environment).

- XSP (Development Webserver).
- Linux (Operating System).

In this section we will discuss our experiences using these tools, and compare them to their commercial alternatives.

Mono

Mono is a cross platform, open source .NET development framework. (See more in section 6.3 on page 42) The Microsoft alternative to this is their .NET Framework implementation, which only runs on the Microsoft Windows operating system. Our experience using mono itself were actually quite good, all the C# language features that we used worked and all the ASP.NET components worked just as expected, even using only MSDN as reference. To our surprise Mono offered a connector to the SQLite database as standard, we only needed to include the relevant reference to the project.

MonoDevelop

We used MonoDevelop in our project, which is a Linux and OSX IDE for developing, mostly C# and VB.NET. The Microsoft counterpart to this is Visual Studio. With MonoDevelop we really had a experience of using "an unfinished product", many features were missing.
Basic features such as code completion only worked half the time, and when working, the inline help and reference did not work. Help and reference within mostly read "To be added".

Another basic thing for a modern IDE, syntax highlighting were missing. On all ASP pages we had no syntax highlighting nor did we have code completion. All ASP pages were handcrafted in a "normal plain text editor" within MonoDevelop. This had an impact on development, as many errors were hard to find. A simple "red underlining" like in Visual Studio would have pointed out immediately. All ASP errors were only detected because of runtime errors seen in the browser.

Most modern IDEs include a debugger, to inspect code when running, with breakpoints - some even allows one to edit code when running, all of these exotic debugging features were all missing in MonoDevelop, so our debugging time took considerably longer than necessary.

MonoDevelop included integrated support for NUnit and XSP, which worked just fine. We used this heavily in the project. One thing to note about our experience with MonoDevelop is, that we used version 1.0, the first release of MonoDevelop. This is a rather old release (March 2008) [11]. The

next stable release of MonoDevelop is just around the corner, version 2.0. According to the Development Roadmap many of our remarks should be corrected/added, including [10]:

- Integrated debugger
- New code completion engine
- ASP.NET code completion
- New editor, color schemes, code folding

XSP

XSP is a Mono ASP.NET web server, it provides a minimalistic web server for testing ASP.NET sites made with Mono. This is integrated with MonoDevelop, such that a click on "Run" opens a web browser on the local XSP server. We do not have much to say about this server. It worked well out of the box. We never did any configuration on it, it just worked, which is good for a development server.

Linux

Without going into any details, Linux is a operating system that is open source, the obvious Microsoft candidate is Windows. Covering Linux is way out of scope of the report. But all the project members were competent and experienced Linux users, so using Linux in respect to this project did not give problems. We used many Linux's developer friendly capabilities such as the bash shell, make files, source control and many more.

8.5 System Development Method

As mentioned in section 7.7 on page 55 we wrote what we had done with the development methods we had chosen to use. In this section we will reflect on our choices and explain what we should or could have done.

Development Method

The experience we had with the evolution model was good, it worked well with the project specification, we consider using this method in a future project if the project specification is as strict as this one was. Coding in the group room work well, this is a thing most of us did the last project, and is something that we will do in future projects. By having a central database and the system running locally on our computers we were able to code on our own pc without interfering with each other, so this is a method we also

would use in a future project.

Instead of a timetable we set goals and determined which dates the goals should be completed, no use of a timetable with the development method that we chose, turned out to work well and we were done with the programming on time, we are considering doing this again if we are using the evolution model as a development method.

Coding Standards

As mentioned in section 5.2 on page 37 we set up some coding standard, that we wanted to uphold in order to produce readable and understandable code. We would like to use this opportunity to reconsider rule two. We found that if there is a need for a comment it should be refactored instead. This will improve the readability and avoid outdated comments.

Review and Feedback

Our choice for review and feedback was formal peer review, with this method we arranged five group meetings where we talked about code that we had studied in advance, this was a good way to gain insight in all of the code that we have written, and is something that we consider doing in a future project.

Chapter 9

Backup Strategy

9.1 Introduction

For every production system, there must be a fully prepared and tested backup plan, in case of system failure.
In our case the most vital part is the data stored in the database server, as it is the live data, and the most crucial to save. The files themselves on the webserver are not important, as there are always a backup on the source revision server system that can be copied back to the webserver if anything happens. We have chosen that our backup strategy should include files on the server, because the long term idea for the system is to have documentation, attached files on issues, and so on. And thereby the system will store those files on the file system.

9.2 Backup of file system

As our system is running a Linux operating system, we need a backup solution that will run on that platform. Backup Ninja is a system for Unix systems to run coordinated backups, in different formats and destinations. Backup Ninja supports the following types [4]:

- Secure, remote, incremental file system backup (rdiff)
- Encrypted remote backup (duplicity)
- CD/DVD/ISO Backups
- Incremental rsync

We will use the secure remote incremental file system backup using rdiff backup. This can be configured to backup using a SSH connection to a remote site, at specified intervals. Another positive feature is that it is incremental, meaning that only, for instance, once a week take a full backup and then afterwards the next 6 days, you only make "incremental" backups. The rdiff tool finds the difference between the last transfered file and the current one on thefile system. This keeps the network traffic at a minimum which is very good, especially when the data size gets large on the servers. This also saves disk space on the backup server, because you can save a backlog of, for instance, seven days, and only use much less disk space. If a file has not been changed, if simply were not take up any additional disk space or network traffic.
Using this we have an efficient and secure remote backup with backlog. We will recommend running the file system backup every night, because the files will not change particularly, compared to database data.

9.3 Backup of database

To insure that no data gets lost in case of emergency. We want a separate plan for backing up database data. As the database data is the most important, and the data that gets edited and changed by far the most. If this data is destroyed, this is where the users will experience the most loss.
To insure minute to minute backup of the data, we will use MySQL replication capabilities to replicate our data to a secondary MySQL installation, for instance on the backup server. This will seamlessly keep a up-to-date copy of our data. The only drawback of this method is, that we do not have a backlog, but this can be tackled using Backup Ninjas MySQL backup functionality, and then keep a day-to-day backlog [17].

Chapter

10

Future Development

10.1 Commercial Perspective

Our support tool has been developed with the goal of aiding university based projects. The system can however be used in other contexts with no modifications. With some modifications it could aid specific context more.

Here are some modifications that will improve the system in a commercial context:

- Improved time tracking
- Reports and statistics
- Integration with IDE
- Integration with billing system
- Customer role/login

Improved time tracking

To further improve the time tracking system we can add categorization of time, such that time can be entered on different categories, like "Development", "Support" and others that are appropriate to the company. Some companies have different rates depending on the type of work being done, and it can be used for statistics.
Furthermore there could be options to tell if the time used is included within the tender, or part of the guarantee or if it should be billed directly to the customer.

Reports and statistics

To fully utilize the data in the system, making reports and statistics can be really helpful to management. The system should provide different useful reports that can be generated with different parameters to give the best and most useful report.

Integration with IDE

To maximize developer productivity the system can be integrated with the company's IDE. This will give the developer a sidebar in the IDE where notes, time and other aspects of the support tool can be controlled directly. The sidebar can even track the time used, if the developer clicks "start" on the issue, and then later adds a note or closes the task, the sidebar plugin can already have the time pre filled. For the Eclipse IDE a whole framework already exists for integrating support tools[2]. This integration can also be helpful to non-commercial contexts.

Integration with billing system

The system can export time used into the company billing system, to automatically or semi-automatically help smoothen the billing process. The billing-department can then run the import, and verify the data and send out the bills. This will minimize billing errors, in form of lost time, or simple typos.

Customer role/login

For companies a customer role is essential to be effective and minimize the time project managers use in the phone with the customer. With a customer role, the customer representative can watch the status of the project and submit feature requests or bugs. This technique will minimize misunderstandings because every communication is in writing. The customer role should, of course, only have access to the relevant projects, and there should be possibility for staff to add private issues and private notes, for internal communication that is hidden from the customer.

Chapter

11

Conclusion

11.1 Evaluation

In this section we will discuss the use of our product in the next semesters and during the current semester. The evaluation criteria for this project are whether we choose to use it in the next semester and how beneficial we consider our system to be. Considering that all essential features are implemented, using the system in the following semester would be both possible and useful. Not all the optional requirements are implemented, and we believe that this is what makes the difference between the system just being useful to being very beneficial. This may also be what makes the difference when we make our choice between our own *ProjectTracker* and a fully tested free open source version with all this functionality implemented.
During the development of the project, our use of the system was sparse, as described in section 8.2 on page 59, because *ProjectTracker* was at a limited state, and not all features were implemented.
This has influenced our use of the system in different ways. The lack of response from the system has been a factor. If the system could notify us by email on relevant events, it would have helped on the motivation for using the system, because even though we used the system regularly, this feature would help work flow. For instance if a developer was waiting for a specific task to be done, he would be alerted and he would know immediately when to continue his work. During the development, we found that we lacked test and review notifications. This would have motivated a more consequent and strict use of reviews in general and might raise the quality of even informal reviews.

11.2 Conclusion

This project started with fixed and predefined requirements. It meant we did not spent time finding our own topic and that problem analyzing could be commenced immediately. We summarized the requirements into a "Problem Formulation" that states what the actual problem is.
We continued analyzing the requirements in relation to quality goals, user roles and functionality requirements.
Afterwards we began to analyze the problem and found the overall structure. We selected appropriate design patterns and determined how to model our data. We based our database schematics on our data models and we determined how our final product should work and behave.
When our analysis work were complete, we looked into different development methods, coding standards and review methods. Finally we made two choices based on our research; Evolution Model and formal reviews.
In the beginning of the phase we researched a number of potential tools and systems, such as programming languages and databases. Based on this

we choose to use the C# language, the Mono framework and the MySQL database.
During development we documented our process focusing on challenging and vital parts of the system. Among other things, we documented how we encountered a serious of performance issues and how we solved it, and what problems we had in relation to unit testing.
We reflected upon our own process and choices, and suggested ideas to further development of the product.

Bibliography

[1] Ralph Johnson Erich Gamma, Richard Helm and John Vlissides. *Design patterns : elements of reusable object-oriented software*. Addison Wesley, 1994.

[2] Eclipse Foundation.
http://www.eclipse.org/mylyn/.

[3] Paul Hamill. *Unit Test Frameworks*. O'Reilly, 2005.

[4] Riseup Labs. Backup ninja homepage.
http://dev.riseup.net/about-riseup-labs/.

[5] Steve McConnell. *Code Compelete, Second Edition*. Microsoft Press, 2004.

[6] Microsoft.
http://www.microsoft.com/NET/.

[7] Microsoft.
www.microsoft.com/SQL/default.mspx.

[8] Microsoft.
http://www.asp.net.

[9] Sun Microsystems.
http://java.sun.com/blueprints/patterns/MVC-detailed.html.

[10] Mono. Mono development roadmap.
http://monodevelop.com/Development_Roadmap.

[11] Mono. Mono news page.
http://monodevelop.com/Main_Page.

[12] Microsoft MSDN. Sql server connection pooling (ado.net).
http://msdn.microsoft.com/en-us/library/8xx3tyca.aspx.

[13] Peter Axel Nielsen.
http://cs.aau.dk/~pan/SWP_E08.html.

[14] Pat Niemeyer and Jonathan Knudsen. *Learning Java*. O'Reilly, 2005.

[15] Kurt Nørmark. Oop course homepage.
http://www.cs.aau.dk/~normark/oop-08/html/oop.html.

[16] Novell.
http://www.mono-project.com/Main_Page.

[17] Stefan Hinz Paul DuBois and Carsten Pedersen. *MySQL 5 Certification Study Guide*. MySQL AB Press, 2005.

[18] Defense Acquisition University Press.
http://www.dau.mil/pubs/pdf/SEFGuide%2001-01.pdf.

[19] George Schlossnagle. *Advanced PHP*. Sams, 2004.

[20] Jon Skeet. Implementing the singleton pattern in c#.
http://www.yoda.arachsys.com/csharp/singleton.html.

[21] Man Lung Yiu. Software architecture course homepage.
http://www.cs.aau.dk/~mly/dbsa08/.

Appendices

Appendix

A

Requirements

A.1 Introduction

The requirements are divided into four categories: core, optional, live data and evaluation criteria.

A.2 Core

The core requirements are the minimum that needs to be done in order to fulfill the requirements.

Group members

Record the group members, including competencies and work hours available.

User Roles

The user roles should be developer, project manager and administrator.

- Developer
  These are the main or usual users. Developers work with the tool in a common way, very regularly. Every standard (non administration) use case is accessible from this role. They need a username and password combination to verify. For their work, masks for editing and overviews over personal information and tasks are needed. In particular, developers can:
  - Login and logout the system
  - Edit profile details
  - View all tasks (including progress)
  - View assigned tasks
  - Record work done
  - Change the assigned task status
- Project Manager
  The project managers are allowed to do things that usual users are not, like edit project details (e.g. deadline) or editing every task in the project he is managing. In particular, project managers can:
  - Login and logout the system
  - Edit profile details
  - Assign and schedule tasks
  - Create/Edit/Archive any task

  - View all task details
  - Edit project

- Administrator
  This is the administrator of the tool. He can do everything including those functionalities the developer and the project manager haves. Administrators are furthermore allowed to change user roles on an user and manage users. Besides the specific functionalities from the developer and the project manager, the administrators can also:

  - Login and logout the system
  - Edit profile details
  - Create/Edit/Archive users
  - Create/Edit/Archive projects
  - Assign project to project manager
  - Alter users user roles

Interface

Different user interfaces are required for the different system roles. All of these user interfaces should be web-based, i.e., accessible from a web browser.

- Developer
  For a developer, the web page should provide a login page and then redirect to another page that displays all the tasks assigned to the developer. This page should provide the task related functionality:

  - View assigned tasks
  - View progress of concerned parts of the project
  - Record work done
  - Enter/update personal data

  Each option may lead to its own specific web page.

- Project Manager
  For a project manager, the web interface should provide a login page and then redirect to another page that display all the projects the project manager has. On this page, project related functionalities are required:

  - Assign a developer
  - Edit a project
  - Allocate tasks

  - Schedule tasks
  - Revise tasks
  - View the overall progress of the project

  Each option may lead to its own specific web page.

- Administrator
  For an administrator, the web interface should provide a login page and then redirect to another page that display all the users. On this page, user related functionalities are required:

  - Add/Edit/Archive user
  - Add/Edit/Archive project
  - Assign project manager
  - Alter users user role

  Each option may lead to its own specific web page.

Projects

The system should handle multiple projects within the same system. Such that the same user can actively work across multiple projects.

Main activities

Record the main activities and tasks of the project, including competencies required, work estimates in hours, task status, completion deadlines and the persons responsible.

Task

A task should have the following properties:

- Unique Task ID (ie, #2023)
- Title
- Status
- Description
- Estimate
- Assigned person
- Deadline
- Priority

- Task notes

A task can initially be offered to a group of developers based on their competencies. When one of those developers accepts the task it will be assigned to that developer only.

Schedule

Schedule the work activities of the project. Each task should have a deadline and assigned person that are responsible.

Monitoring

Monitor the progress of the project, including reporting work done, including the status of tasks, visual representation of progress and alerts for overdue tasks. Monitoring should include the following:

- A simple calendar representation of tasks completed on time and overdue
- A representation of the percentage of the project completed, calculated from work hours allocated to completed tasks.
- Status of the tasks, represented (for example) as green, orange, red (traffic light notation) - calculated through an algorithm relating the hours of work remaining and days available remaining.

It should also provide some reporting facilities including comparison of estimated hours for tasks with actual used hours, workload distribution (comparison of the efforts of the developers).

A.3 Optional

Estimation

Project managers and developers should be able to estimate the time tasks and the whole projects will take. I.e the project manager would fill the project with tasks and based on an algorithm, the time required to finish the project would be estimated. With this feature the project managers would be able to calculate a precise finish date.

Source Control Integration

Optionally developers should be able to change the status of tasks from their source control system. I.e., a developer can write a commit note like "This fixes task #2023", which should add a note to the task saying the developer has completed the task #2023 and what source files where affected. The

same should be true for messages like "Worked on task #2023 and spend 2:30". This should add a note on the task and add 2 hours and 30 minutes to the task, as work done by the developer.

Source control options:

- Add note to a task (with time spend)
- Complete a task (with time spend)

Test & Review

A feature to automatically run unit tests, both manually, from the web interface and automatically every night, with a visual presentation of the test results. It should be possible to add the option to send a message when one or more tests fails. The message will be sent to the project leader and the developer(s) who caused the failure.

If the project manager marks a task as "needs review" and then adds a developer as the reviewer, then the task can only be completed when the reviewer has approved the work.

Division of Tasks

With this implementation the developer has the choice to divide a task into a number of smaller tasks. These subtasks can the be viewed, added and altered a task's details page. This will make larger tasks easier to cope.

Documentation

It should be possible to add documentation files on a per project basis. There should be an upload facility with in the web interface to accomplish this.

Bug Reports

It should be possible to report bugs as new tasks. This option is important for maintaining a project after its release. Depending on the severity of the bug the task will have different priority levels. A major bug will be prioritized over a minor bug. Bugs should be created within the taskgroup that it belongs.

A.4 Live Data Requirements

The project support tool should be populated with live data from the project we are working on.

A.5 Evaluation Criteria

Whether we ourselves use the tool in our next project; we should create a tool which we ourselves would find beneficial to use.

Appendix

B

Database Diagram

The Entity Relationship diagram is shown in figure B.1.

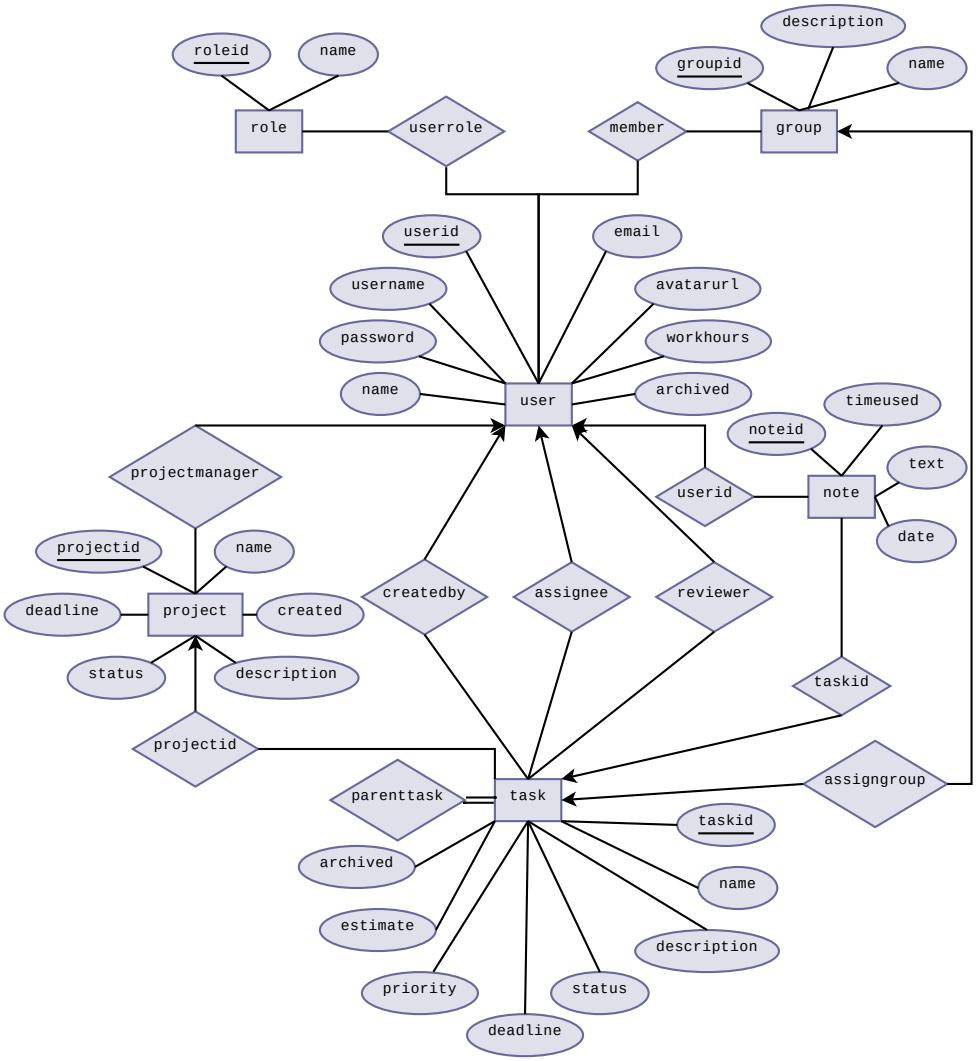


Figure B.1: The Database ER Diagram.

Appendix

C

Object creation graph

Object creation graph is shown in figure C.1.

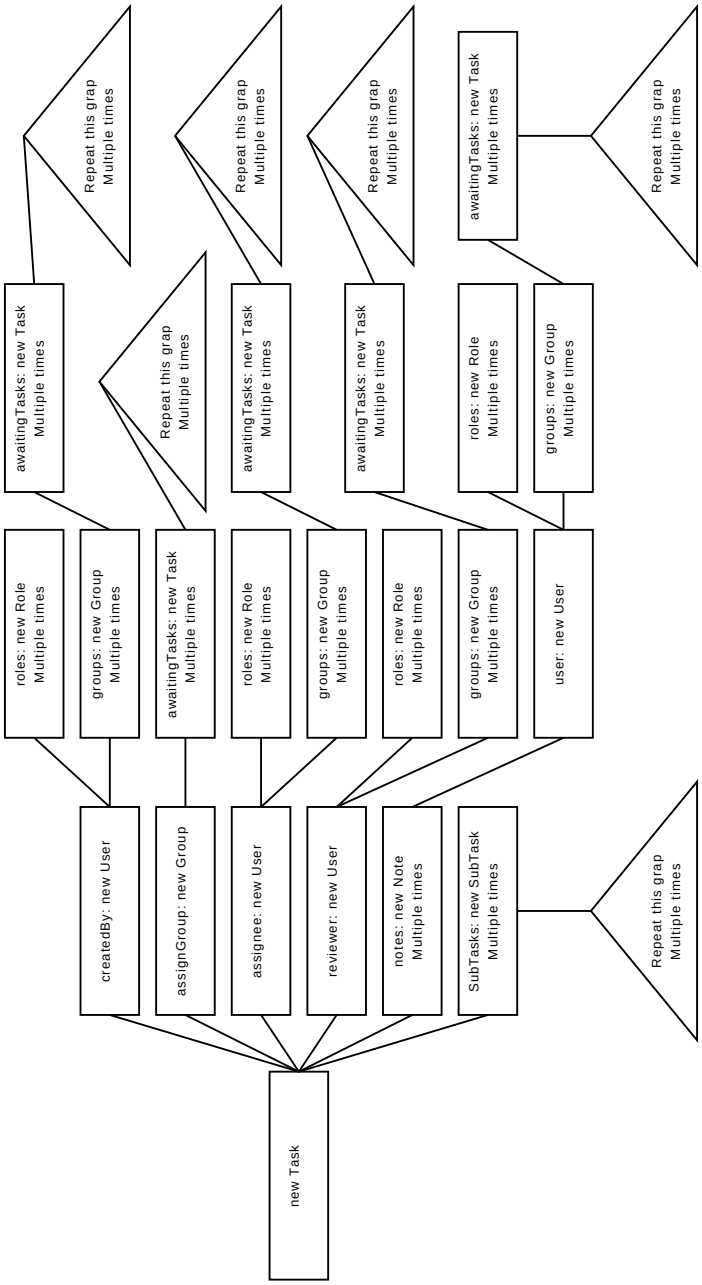
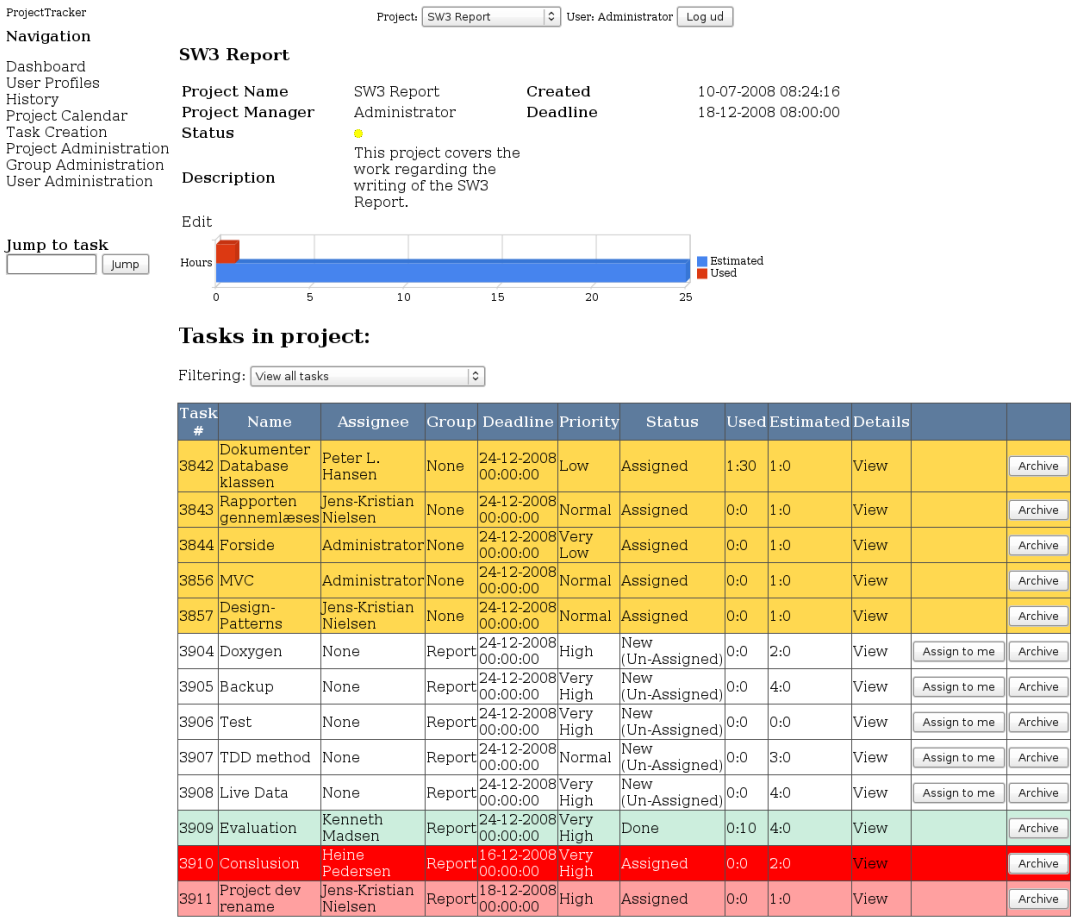


Figure C.1: Object creation graph.

Appendix

D

Screenshots



Task #	Name	Assignee	Group	Deadline	Priority	Status	Used	Estimated	Details		
3842	Dokumenter Database klassen	Peter L. Hansen	None	24-12-2008 00:00:00	Low	Assigned	1:30	1:0	View		Archive
3843	Rapporten gennemlæses	Jens-Kristian Nielsen	None	24-12-2008 00:00:00	Normal	Assigned	0:0	1:0	View		Archive
3844	Forside	Administrator	None	24-12-2008 00:00:00	Very Low	Assigned	0:0	1:0	View		Archive
3856	MVC	Administrator	None	24-12-2008 00:00:00	Normal	Assigned	0:0	1:0	View		Archive
3857	Design-Patterns	Jens-Kristian Nielsen	None	24-12-2008 00:00:00	Normal	Assigned	0:0	1:0	View		Archive
3904	Doxygen	None	Report	24-12-2008 00:00:00	High	New (Un-Assigned)	0:0	2:0	View	Assign to me	Archive
3905	Backup	None	Report	24-12-2008 00:00:00	Very High	New (Un-Assigned)	0:0	4:0	View	Assign to me	Archive
3906	Test	None	Report	24-12-2008 00:00:00	Very High	New (Un-Assigned)	0:0	0:0	View	Assign to me	Archive
3907	TDD method	None	Report	24-12-2008 00:00:00	Normal	New (Un-Assigned)	0:0	3:0	View	Assign to me	Archive
3908	Live Data	None	Report	24-12-2008 00:00:00	Very High	New (Un-Assigned)	0:0	4:0	View	Assign to me	Archive
3909	Evaluation	Kenneth Madsen	Report	24-12-2008 00:00:00	Very High	Done	0:10	4:0	View		Archive
3910	Conslusion	Heine Pedersen	Report	16-12-2008 00:00:00	Very High	Assigned	0:0	2:0	View		Archive
3911	Project dev rename	Jens-Kristian Nielsen	Report	18-12-2008 00:00:00	High	Assigned	0:0	1:0	View		Archive

Figure D.1: Dashboard page.

ProjectTracker

Navigation

Dashboard
User Profiles
History
Project Calendar
Task Creation
Project Administration
Group Administration
User Administration

Jump to task

Jump

Project: SW3 Report User: Administrator Log ud

Administrator

Edit your profile

Frederik Højlund

User ID: 31

Email: vegavov@gmail.com

Currently deleted/achieved: No

Roles
Administrator

Competences
HTML
CSS
C#
ASP
MySQL
Report

Figure D.2: User Profiles page.

ProjectTracker

Navigation

Dashboard
User Profiles
History
Project Calendar
Task Creation
Project Administration
Group Administration
User Administration

Jump to task

Jump

Project: SW3 Report User: Administrator Log ud

History Log:

Filtering: View Info

Log entry
01-12-2008 10:55:31 [INFO] Attempt to access protected page. Redirecting to login page. [IP:127.0.0.1]
01-12-2008 10:55:34 [INFO] admin logged in
03-12-2008 09:23:37 [INFO] admin logged in
03-12-2008 09:36:38 [INFO] Attempt to access protected page. Redirecting to login page. [IP:127.0.0.1]
03-12-2008 09:36:41 [INFO] admin logged in
03-12-2008 11:50:22 [INFO] admin logged in
04-12-2008 09:52:11 [INFO] admin logged in
10-12-2008 14:03:42 [INFO] admin logged in
10-12-2008 14:10:11 [INFO] UPDATE tasks SET name = @name, deadline = @deadline, description = @desc, assignee = @assignee, reviewer = @reviewer, priority = @priority, status = @status, projectid = @projectid WHERE taskid = @id
10-12-2008 14:14:24 [INFO] Attempt to access protected page. Redirecting to login page. [IP:127.0.0.1]
10-12-2008 14:14:28 [INFO] admin logged in
10-12-2008 14:15:05 [INFO] UPDATE tasks SET archived = (NOT archived) WHERE taskid = @id
10-12-2008 14:15:10 [INFO] UPDATE tasks SET archived = (NOT archived) WHERE taskid = @id
10-12-2008 14:15:15 [INFO] UPDATE tasks SET archived = (NOT archived) WHERE taskid = @id
10-12-2008 14:15:39 [INFO] UPDATE tasks SET name = @name, deadline = @deadline, description = @desc, assignee = @assignee, reviewer = @reviewer, priority = @priority, status = @status, projectid = @projectid WHERE taskid = @id
10-12-2008 14:17:11 [INFO] admin logged in
10-12-2008 14:17:14 [INFO] admin logged in
10-12-2008 14:18:51 [INFO] Attempt to access protected page. Redirecting to login page. [IP:127.0.0.1]
10-12-2008 14:18:53 [INFO] admin logged in
10-12-2008 14:19:22 [INFO] UPDATE tasks SET name = @name, deadline = @deadline, description = @desc, assignee = @assignee, reviewer = @reviewer, priority = @priority, status = @status, projectid = @projectid WHERE taskid = @id
10-12-2008 14:20:23 [INFO] INSERT INTO tasks (projectid, name, description, assignee, deadline, reviewer, priority , status, parenttask, createdby, assigngroup, estimate) VALUES (@projectid, @name, @description, @assignee, @deadline, @reviewer, @priority , @status, @parenttask, @createdby, @assigngroup, @estimate)
10-12-2008 14:20:24 [INFO] new task created: "MEGA BUG"

Figure D.3: History page.

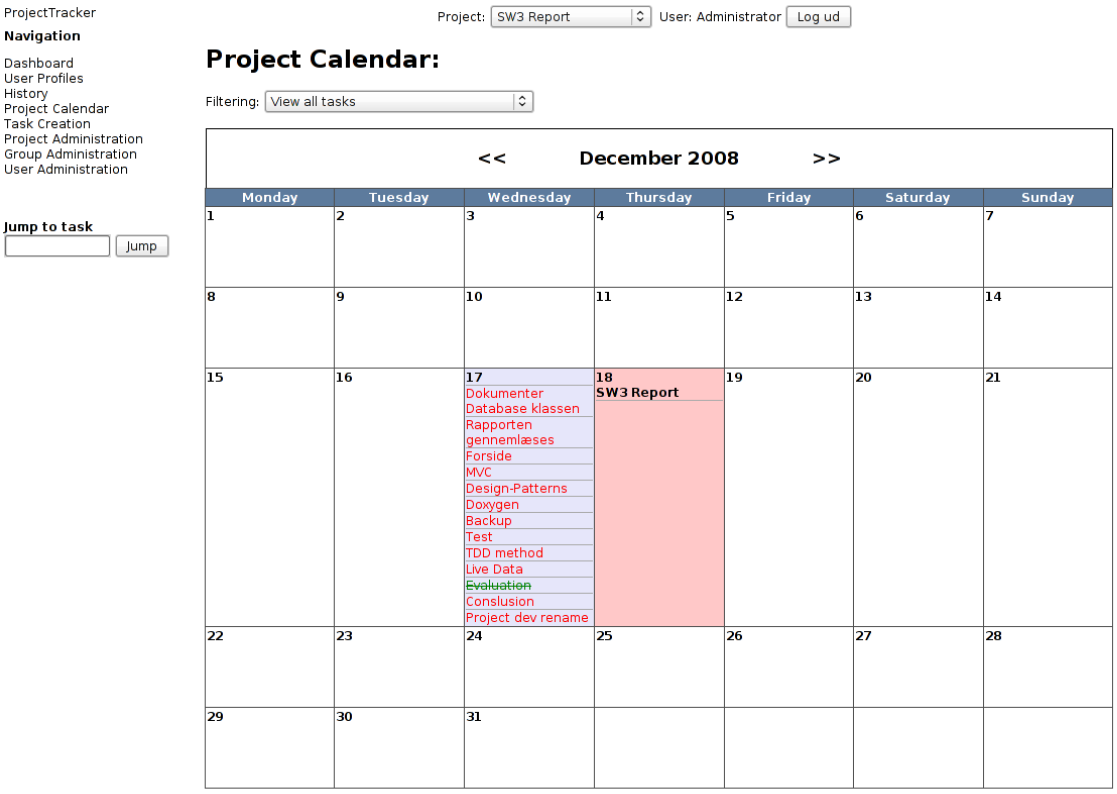


Figure D.4: Calendar page.

Figure D.5: Task Creation page.

Figure D.6: Project Administration page.

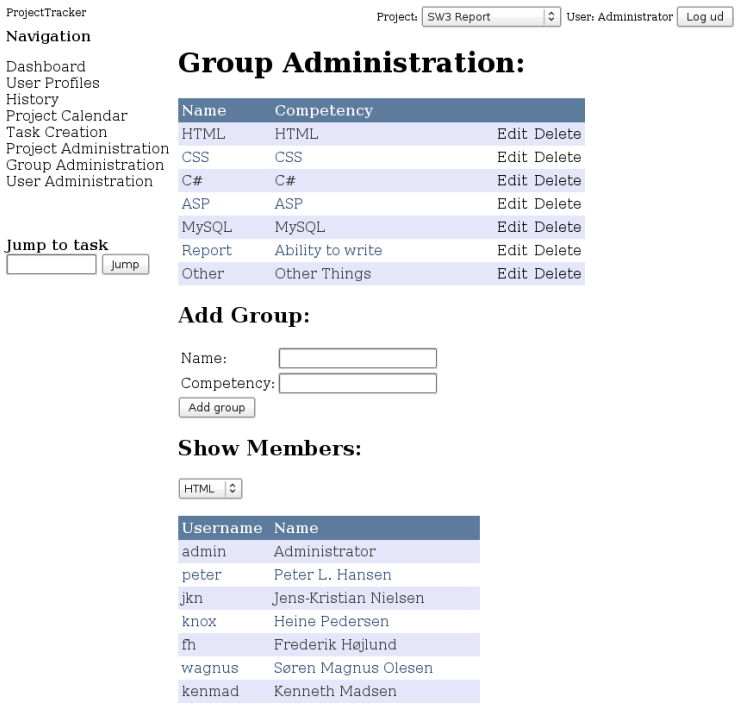


Figure D.7: Group Administration page.

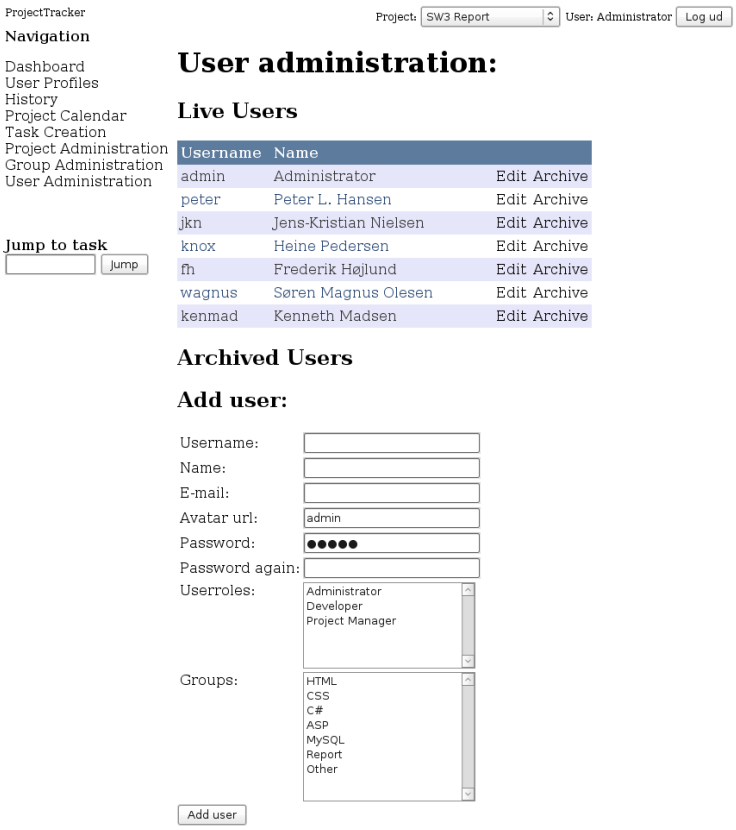


Figure D.8: User Administration page.