Software Engineering 1 Project Report

by
Group 19
Atakan Kaya, s120872
Filipe Silva, s124755
Marc Thomsen, s093022
Marianne Louis-Hansen, s072188

Technical University of Denmark
DTU Compute
02161 Software Engineering 1
13. maj 2013

Indhold

1 Introduction

by Marc Thomsen

The project period of this course consists of the development of a small software system. The topic is a project management and working time registration system which is targeted at companies developing software projects. The purpose of this system is to facilitate the development process for the employees in a software company.

For the project management part this tool aims to display the availability of the employees during a certain period of time, give an overview of the employees working on a specific project and allows each employee to assign them self or other employees to a given task in a project. Each employee should be able to create a project and determine if the project is internal or if it is ordered by another company (external). Other properties of a project includes the start- and end date, a description, the state and the ability to assign a manager for the project can be entered or edited after the creation.

For the working time registration part, the system should be able to keep track of the spent and remaining time of each project, as well as the assigned activities and subtasks.

To ensure that the functionalities implemented work as intended the most important use case scenarios in this system and the corresponding systematic tests are included in the report.

2 Specification requirements

2.1 Key Concepts

By Marianne Louis-Hansen

Activity
The activities are the sub parts of the project. These are determined by the project manager, but need not be in place when a project is created.

Developer
An employee, who is assigned to a project by the project manager, as opposed to a helper, who is an employee assigned to help on a task or activity by an already assigned developer.

End date
This is the expected finish date of the project or the activity. The project as a whole has a finish date and each individual activity has a finish date as well.

Employee
The individuals working on the project or activities, who are not the project manager.

Employee username
Each employee logs into the system with their username, which consists of up to four letters.

Helper
An employee who is assigned by another employee to help him or her with an assigned task or activity.

Project
The overall project, whether internal or external.

Project creation
The project creation is the act of creating a new project.

Project manager
The project manager is the overall coordinator of the project, and divides the project into activities, as well as assigning employees to different activities.

Project name
This is the name given to a project when it is created.

Project ID
When a project is created, it is given a number by the system. This number consists of the year the project is created, and a serial number.

Start date
The date a project or activity is due to begin

State
Projects, activities and tasks all have states, which tells how far along they are. States can be, for example, finished, not yet started, ongoing, or paused.

Task
The smallest unit the project is divided into. Tasks are thus subunits of activities.

Time recording
All time spent working on an activity is registered, with 1/2 hours precision. This goes for everyone who works on an activity, even when not assigned to that specific activity. Vacation time and sick leave and other time spent away from a project is also registered.

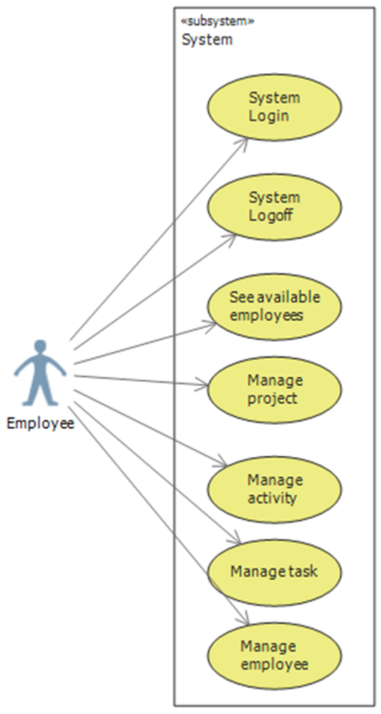
Unavailability
A period of time when the employee cannot be assigned to a new task. This might be because of workload, if the employee already has 20 task assigned in a week, or because he or she has registered an unavailable period in the system, for example for vacation or because of illness.

Work hours
The amount of work hours per activity is the expected amount of time needed to complete the activity.

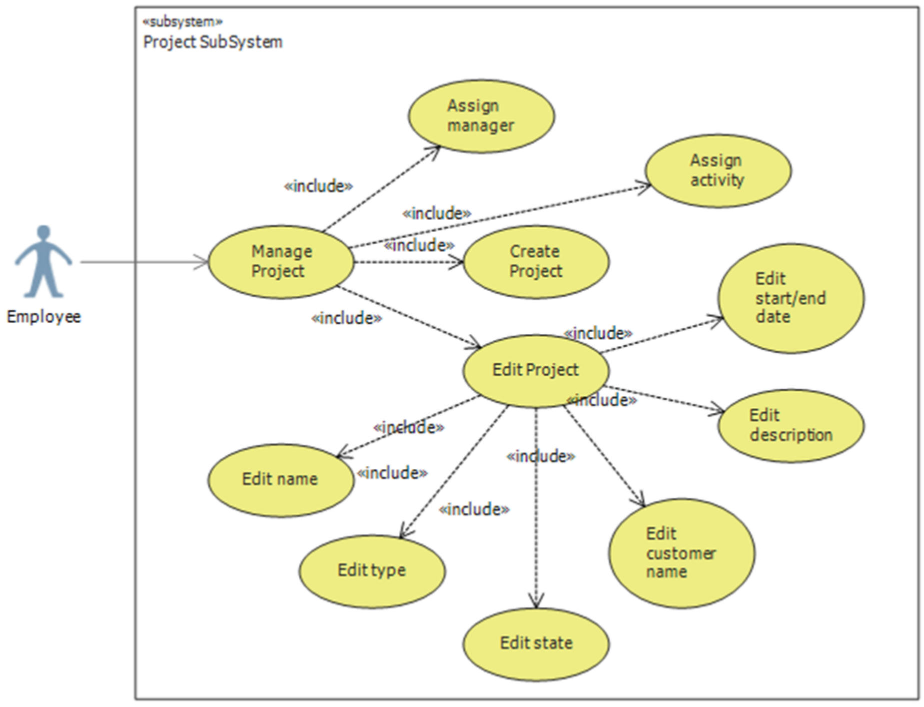
Work remaining
The amount of work still needed on a project, found by subtracting the work already done from the planned amount of work required for the activity.

2.2 Use Case Diagrams

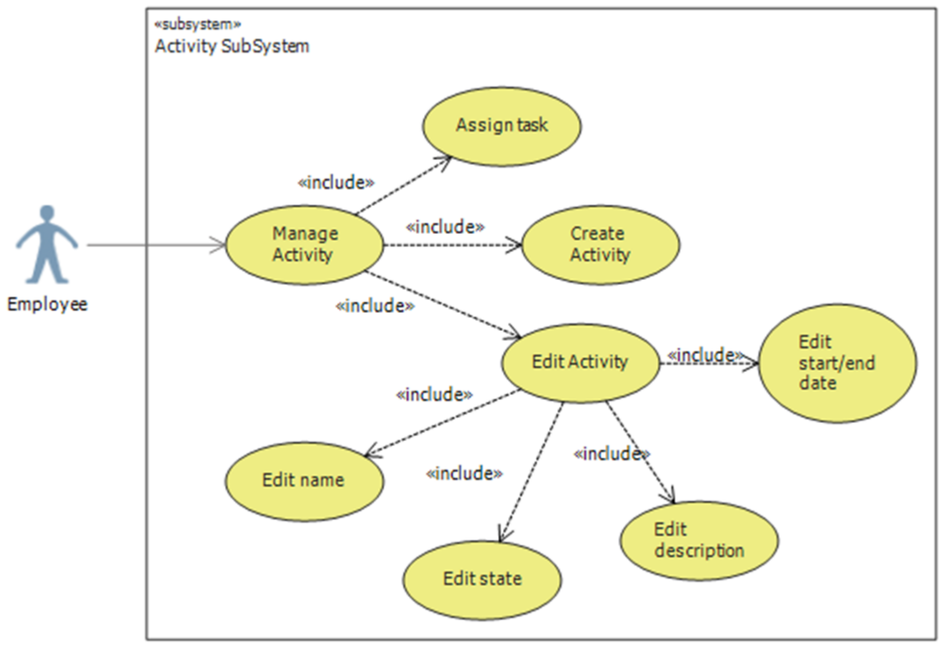
This section contains the use case diagrams for the overall system, and the for the major components of it.



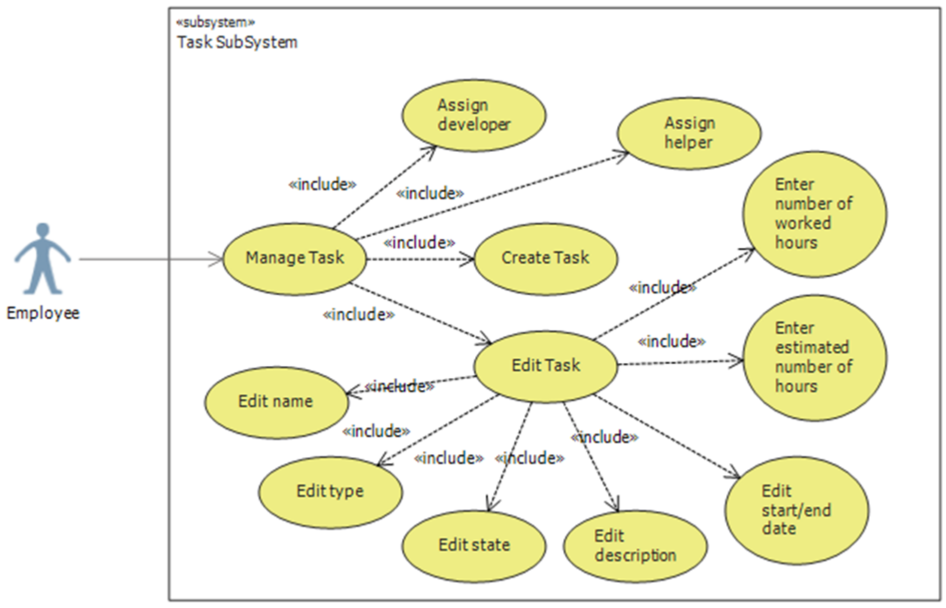
Figur 1: Use Case diagram for overall system. (Atakan Kaya)



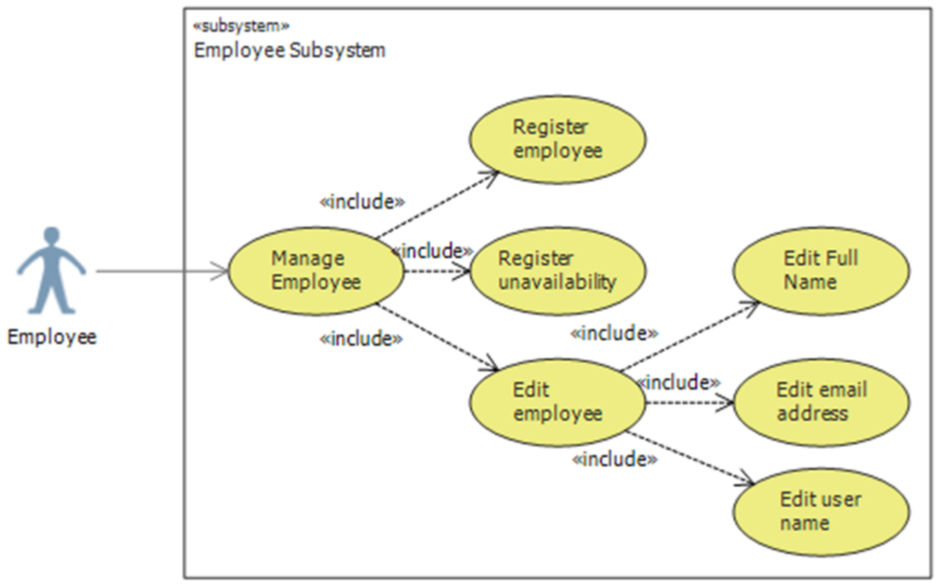
Figur 2: Use Case diagram for Manage Project. (Atakan Kaya)



Figur 3: Use Case diagram for Manage Activity. (Atakan Kaya)



Figur 4: Use Case diagram for Manage Task. (Atakan Kaya)



Figur 5: Use Case diagram for Manage Employee. (Atakan Kaya)

2.3 Detailed Use Cases

This section contains the detailed use cases for the important functions of the system. We chose to do use cases for most of the functionality as we implemented it, thus there are more than the required 8. They are arranged more or less in chronological order, as a user would encounter the actions when using the system.

Use Case 1 by Filipe Silva

Name: Register an employee

Description: Allows an unregistered employee to add himself to the system's list of employees. For that the system requires the username, the full name and the e-mail.

Actor: Unregistered employee.

Precondition: The employee is unregistered in the system's list of employees.

Main scenario:

1. The employee types the full name, the username and the e-mail address, for the registration and, confirms when finished.
2. The system registers the employee

Alternative scenario:

A1 - The employee has an invalid username, longer than 4 characters, or an empty field when finish the insertion of the data.

A2 - Notify the employee about the invalid fields and terminate the use case.

B1 - The username chosen by the employee is already in the system.

B2 - Notify the employee about that fact and terminates the use case.

2.3.1 Use Case 2 by Marc Thomsen

Name: System Log In

Description: provides a log in function to the project management system requiring the username of the employee.

Actor: Employee

Precondition: no employee is already logged into the system.

Main scenario:

1. The employee types his/her username
2. System accepts the username and logs user in, opens the main window

Alternative scenario:

A1 - The employee types his/her username wrong.

A2 - Notify the employee of the fact and start the use case from beginning.

Use Case 3 by Filipe Silva

Name: Create Project

Description: The employee creates a project and the system adds it to the list of projects.

Actor: Employee

Precondition: Employee must be logged in.

Main scenario:

1. Employee provides these mandatory information:
    - Project name
    - Project type (either internal or external)
    - (if project is external) Name of customer company
2. Project is saved and added to the list of projects.

Alternative scenario:

A1 - The input data is not correct (see below).

A2 - The system does not save the project, notifies the user of that fact and terminates the use case.

Note: The input data is correct if:

- All the mandatory information is provided (Project name and customer name cannot be empty string).

Use Case 4 by Marianne Louis-Hansen

Name: Assignment of a project manager to a project

Description: Assigns an employee to a project as a project manager.

Actor: Employee

Precondition: The project - which the manager is assigned to - already exists.

Main scenario:

1. The username of the employee being assigned is entered.
   a. There is no employee assigned as the manager for the project. System saves the project with the new information and notifies the user of this change.
   b. A project manager is already assigned to the project. Remove previous manager and assign new employee as the new project manager.

Alternative scenario:

A1 - The employee being assigned is not a registered user

A2 - Notify the user that the employee is not found and end the use case.

Use Case 5 by Marc Thomsen

Name: Create Activity and Add to a Project

Description: The employee first creates and afterwards associates an activity to a project

Actor: Employee

Precondition: Employee must have been logged in and the project should have been created beforehand.

Main scenario:

1. Employee provides these mandatory information:
   - Project that activity will belong to
   - Activity name
2. The activity is saved and added to the activities of the project.

Alternative scenario:

A1 - The input data is not correct (see below).

A2 - The system does not save the activity, notifies the user of that fact and terminates the use case.

Note: The input data is correct if:

- All the mandatory information is provided (Activity name is not empty string),
- The project which the activity will be associated to already exists.

<u>Use Case 6</u> by Atakan Kaya

Name: Create Task and Add to an Activity

Description: The employee creates a task and adds it to an activity

Actor: Employee

Precondition: Employee must have been logged in and the activity should have been created beforehand. This activity must belong to an existing project.

Main scenario:

1. Employee provides this mandatory information:
    - Activity that the task will belong to
    - Task name
2. Task is saved and added to the tasks of the activity.

Alternative scenario:

A1 - The input data is not correct (see below).

A2 - The system does not save the task, notifies the user of that fact and terminates the use case.

Note: The input data is correct if:

- All the mandatory information is provided (task name should not be empty),
- The activity - the task will be associated to - already exists.

Use Case 7 By Marianne Louis-Hansen

Name: Assign an Employee to a Task

Description: The employee assigns another employee to a task.

Actor: Employee

Precondition: Employee that assigns another employee must have been logged in, employee that is assigned to a task should exist in the system and the task should have been created beforehand. Task must also have a start date and end date.

Main scenario:

1. The employee selects a task.
2. The employee selects another employee to assign to the task.
3. The employee is available at that period.
4. The system saves the information.

Alternative scenario:

A1 - The employee has 20 tasks in any of the weeks between start date and end date of the task.

A2 - The system notifies the user of that fact and terminates the use case.

B1 - The employee that is assigned to the task is not available at that period of time (Vacation, illness, etc.)

B2 - The system does not save the task, notifies the user of that fact and terminates the use case.

C1 - The employee that is assigned to the task is a helper of that task.

C2 - The system notifies the user of that fact and terminates the use case.

Use Case 8 by Atakan Kaya

Name: Edit Project

Description: The employee edits a project

Actor: Employee

Precondition: Employee must have been logged in and the project should exist.

Main scenario:

1. The employee selects a project and changes any of the following properties:
    - Project name
    - Project type (either internal or external)
    - (If project type is external) Name of customer company.
    - Project description
    - Start date
    - End date
    - Project state
2. The system saves the project with the new information.

Alternative scenario:

A1 - The input data is not correct (see below).

A2 - The system does not save the project, notifies the user of that fact and terminates the use case.

B1 - The start date of the project is later than the start date of any of its activities.

B2 - The system notifies the user of that fact, and terminates the use case.

C1 - The end date of the project is earlier than the end date of any of its activities.

C2 - The system notifies the user of that fact, and terminates the use case.

Note: The input data is correct if:

- Project name is not empty string,
- Start date is the Monday of a week,
- End date is a Friday of a week,
- End date is later than start date,

Use Case 9 by Atakan Kaya

Name: Edit Activity

Description: The employee edits an activity

Actor: Employee

Precondition: Employee must have been logged in.

Main scenario:

1. Employee selects an activity of a project and changes any of the following properties:
    - Activity name
    - Activity description
    - Start date
    - End date
    - Activity state
2. The system saves the activity with the new information.

Alternative scenario:

A1 - The input data is not correct (see below).

A2 - The system does not save the activity, notifies the user of that fact and terminates the use case.

B1 - The start date of the activity is later than the start date of any of its tasks.

B2 - The system notifies the user of that fact, terminates the use case.

C1 - The end date of the activity is earlier than the end date of any of its tasks.

C2 - The system notifies the user of that fact, terminates the use case.

D1 - The start date of the activity is earlier than the start date of its project.

D2 - The system notifies the user of that fact, asks the user if the start date of the project should be overwritten by the start date of the activity.

  a. If yes, system overwrites the start date of the project by the start date of the activity and saves the activity.

  b. If no, system asks for a new start date for the activity.

E1 - The end date of the activity is later than the end date of its project.

E2 - The system notifies the user of that fact, asks the user if the end date of the project should be overwritten by the end date of the activity.

  a. If yes, system overwrites the end date of the project by the end date of the activity and saves the activity.

  b. If no, system asks for a new end date for the activity.

Note: The input data is correct if:

- Activity name is not empty string,
- Start date is the Monday of a week,
- End date is the Friday of a week,
- End date is later than the start date.

Use Case 10 by Atakan Kaya

Name: Edit Task

Description: The employee edits a task

Actor: Employee

Precondition: Employee must have been logged in.

Main scenario:

1. Employee selects a task of an activity and changes any of these properties:
    - Task name
    - Activity description
    - Start date

- End date
- Expected number of hours to solve task
- Number of hours worked for the task
- Task state (Go to use case Edit State)

2. System saves the project with the new information.

Alternative scenario:

A1 - The input data is not correct (see below).

A2 - The system does not save the task, notifies the user of that fact and terminates the use case.

B1 - The start date of the task is later than the start date of the project that the task belongs to.

B2 - The system notifies the user of that fact, asks the user if the start date of the project and the activity that the task belongs should be overwritten by the start date of the task.

   a. If yes, the system overwrites the start date of the project and the activity by the start date of the task and saves the task.

   b. If no, the system asks for a new start date for the task.

C1 - The end date of the task is earlier than the end date of the project it belongs to.

C2 - The system notifies the user of that fact, asks the user if the end date of the project and the activity that the task belongs should be overwritten by the end date of the task.

   a. If yes, the system overwrites the end date of the project and the activity by the end date of the task and saves the task.

   b. If no, the system asks for a new end date for the task

D1 - The start date of the task is earlier than the start date of its activity.

D2 - The system notifies the user of that fact, asks the user if the start date of the activity should be overwritten by the start date of the task.

a. If yes, the system overwrites the start date of the activity by the start date of the task and saves the task.

b. If no, the system asks for a new start date for the task.

E1 - The end date of the task is later than the end date of its activity.

E2 - The system notifies the user of that fact, asks the user if the end date of the activity should be overwritten by the end date of the task.

a. If yes, the system overwrites the end date of the activity by the end date of the task and saves the task.

b. If no, the system asks for a new end date for the task.

Note: The input data is correct if:

- Task name is not empty string,
- Start day is the Monday of a week,
- End date is a Friday of a week,
- End date is later than start date,
- Expected number of hours to solve the task and number of hours worked for the task should not be negative and has 1/2 hour accuracy.

Use Case 11 by Filipe Silva

Name: Edit State functionality

Description: The State functionality permits the employee to set a state for a project, an activity or a task. This state has 4 possible values NOTSTARTED, ONGOING, PAUSED and FINISHED. The default state is NOTSTARTED. When an employee changes the state of a project for NOTSTARTED, PAUSED or FINISHED all the activities and tasks of than project will be set to NOTSTARTED, PAUSED or FINISHED, respectively. If an employee changes the state of an activity to those values all the tasks of that activity will be set to those values too.

Actor: Employee

Precondition: Employee must have been logged in.

Main scenario:

1. The employee selects the project, activity of task that he wants to change the state of
2. The employee selects the state that he wants for the project, activity of task.
3. The system saves the change.

Alternative scenario:

A1 - The employee changes the state to ONGOING of a task or activity while the above classes, (Project and activity for task and Project for activity) are not in the ONGOING state.

A2 - The system changes the state of the above classes to ONGOING.

<u>Use Case 12</u> **by Marc Thomsen**

Name: Request Assistance From Another Employee

Description: An employee who is assigned to a task requests assistance from another employee and adds him/her as a helper.

Actor: Employee

Precondition: The employee who is assigned to a task and the employee who is the helper must exist in the system.

Main scenario:

1. Employee selects another employee to help him/her with the task.
2. Employee enters the estimated work hours for the helper.
3. The system saves the information.

Alternative scenario:

A1 - The helper has 20 tasks in any of the weeks between start date and end date of the task.

A2 - The system notifies the user of that fact and terminates the use case.

B1 - The helper is not available at that period of time (Vacation, illness, etc.).

B2 - The system does not save the task, notifies the user of that fact and terminates the use case.

C1 - The helper is the developer or is already in the list of helpers for that task.

C2 - The system does not save the task, notifies the user of that fact and terminates the use case.

Use Case 13 by Filipe Silva

Name: Register Unavailability

Description: The employee registers things such as vacation, illness, courses, etc., which can't be attributed to an individual project.

Actor: Employee

Precondition: Employee must have been logged in.

Main scenario:

1. The employee registers unavailability by typing the reason and entering the start date and end date.
2. The system saves the information.

Alternative scenario:

A1 - The input data is not correct (see below).

A2 - The system does not save the information, notifies the user of that fact and terminates the use case.

B1 - The employee has tasks assigned for him at that period.

B2 - The system notifies the user of that fact and terminates.

C1 - Employee already has unavailabilities at that period.

C2 - The system notifies the user of that fact and terminates.

Note: The input data is correct if:

- Start date is the Monday of a week,

- End date is a Friday of a week,
- End date is later than start date.

Use Case 14 by Atakan Kaya

Name: Get a Report on the Availability of Employees on a Specific Time

Description: The system generates a report on the availability of employees on a specific time. This time will have week basis. The report includes the total number of tasks assigned to the employee and the total number of hours estimated work for that employee.

Actor: Employee

Precondition: Employee that the report is generated about must exist in the system.

Main scenario:

1. Employee (Project Manager) selects a start date (start date indicates the week).
2. System provides a report including these information:
   - Total number of tasks assigned to the employee
   - Total number of hours estimated work for that employee
   - Total number of worked hours for that employee
   - Total number of remaining hours for that employee
   - Employee's availability (if he/she has 20 tasks for that week or has an unavailability or not)

Alternative scenario:

A1 - The input data is not correct (see below).

A2 - The system does not generate a report, notifies the user of that fact and terminates the use case.

Note: The input data is correct if:

- Start date is the Monday of a week.

2.4 Scope limitations and project specification ambiguities

by Marianne Louis-Hansen

There were a couple of ambiguities in the project specification. One of them is that the project manager is the one who edits projects, activities and tasks, and assigns employees to tasks. It does not say, however, who is supposed to assign the project manager, or whether or not it should be possible to edit a project when no manager is assigned. We have chosen to solve these two problems by allowing all registered users of the system to assign a project manager, and to edit a project that has no manager assigned. This is also necessary, due to the information that the project manager is not necessarily selected when the project is created, and thus there will likely be a need to make changes to a project before the project manager is assigned. This is reflected in the use cases, where we do not distinguish between project manager and regular employee, as the regular employee will be able to access the functionality when a project manager has not been assigned. It would therefore only be adding needless complexity to the use cases if we had to distinguish between cases where a manager has been assigned and cases where on has not.

The project specification mentions that the typical employee will be working on less than 10 tasks a week while certain employees must be allowed to be assigned to up to 20. We have not made this distinction in the system, as there is no indication of what the difference between the two types of employees are, and because it would complicate the code unnecessarily. Thus, it would be up to the project manager to determine which employees would be assigned 20 tasks or only 10.

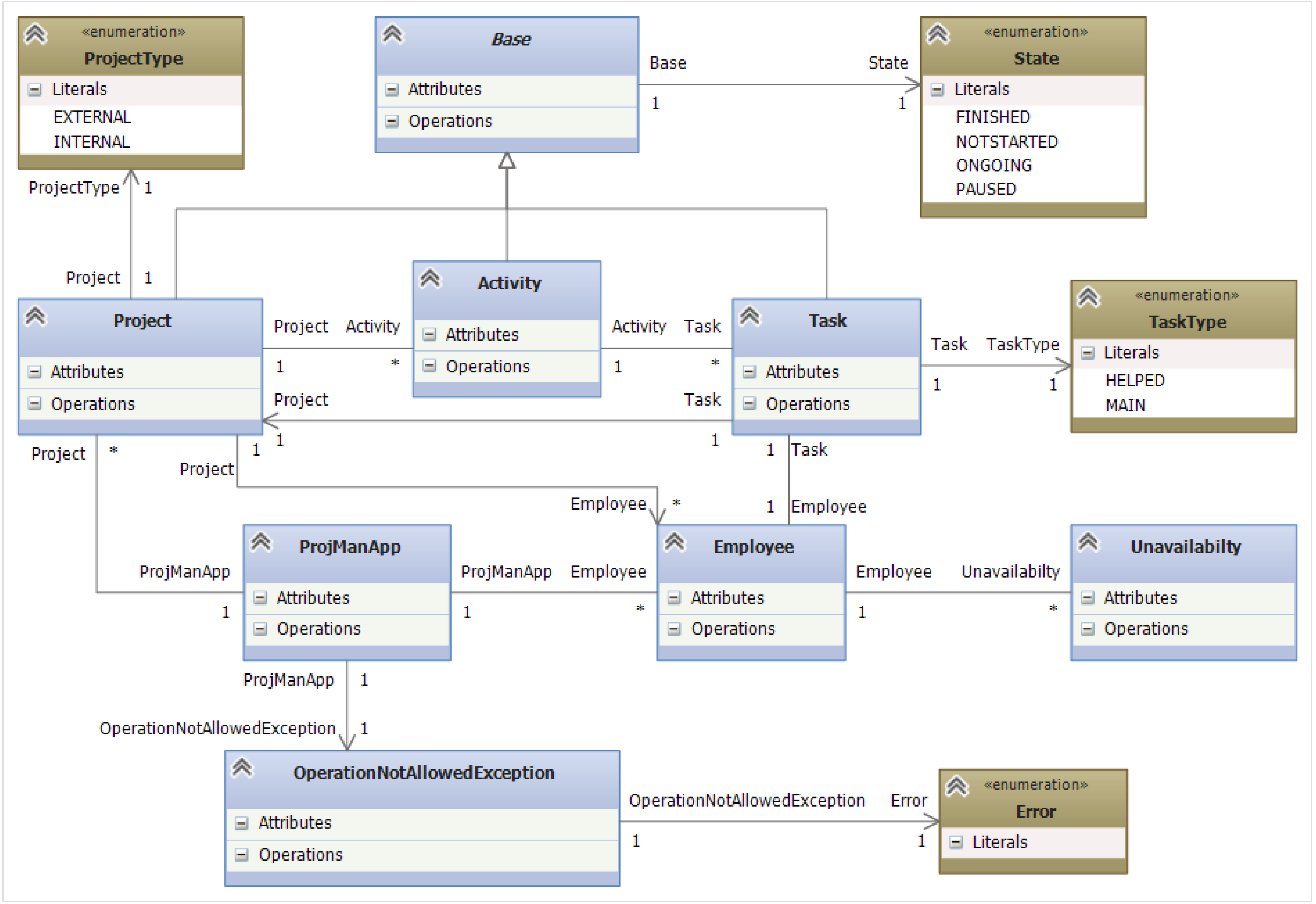
Although it does not say so in the project proposal, we have chosen that all projects, activities and tasks must begin on a Monday and end on a Friday. This was done to simplify the system, and make it easier to handle the employees workload. As the time scale for planning is one week, this seemed the logical solution. Unavailabilities are also scheduled on a weekly scale, as this makes it much easier to integrate with the assignment of task, though it might be a little unrealistic, especially if employees only need one or two days off.

Further, the system does not have a function for deleting information. Thus, we cannot delete projects, activities or tasks. This is only a problem with unavailabilities, since employees will not be able to register for unavailability in a week where they are assigned tasks. We did it this way, because it greatly simplified the code, and we did not have to add functionality for deleting projects or project parts.

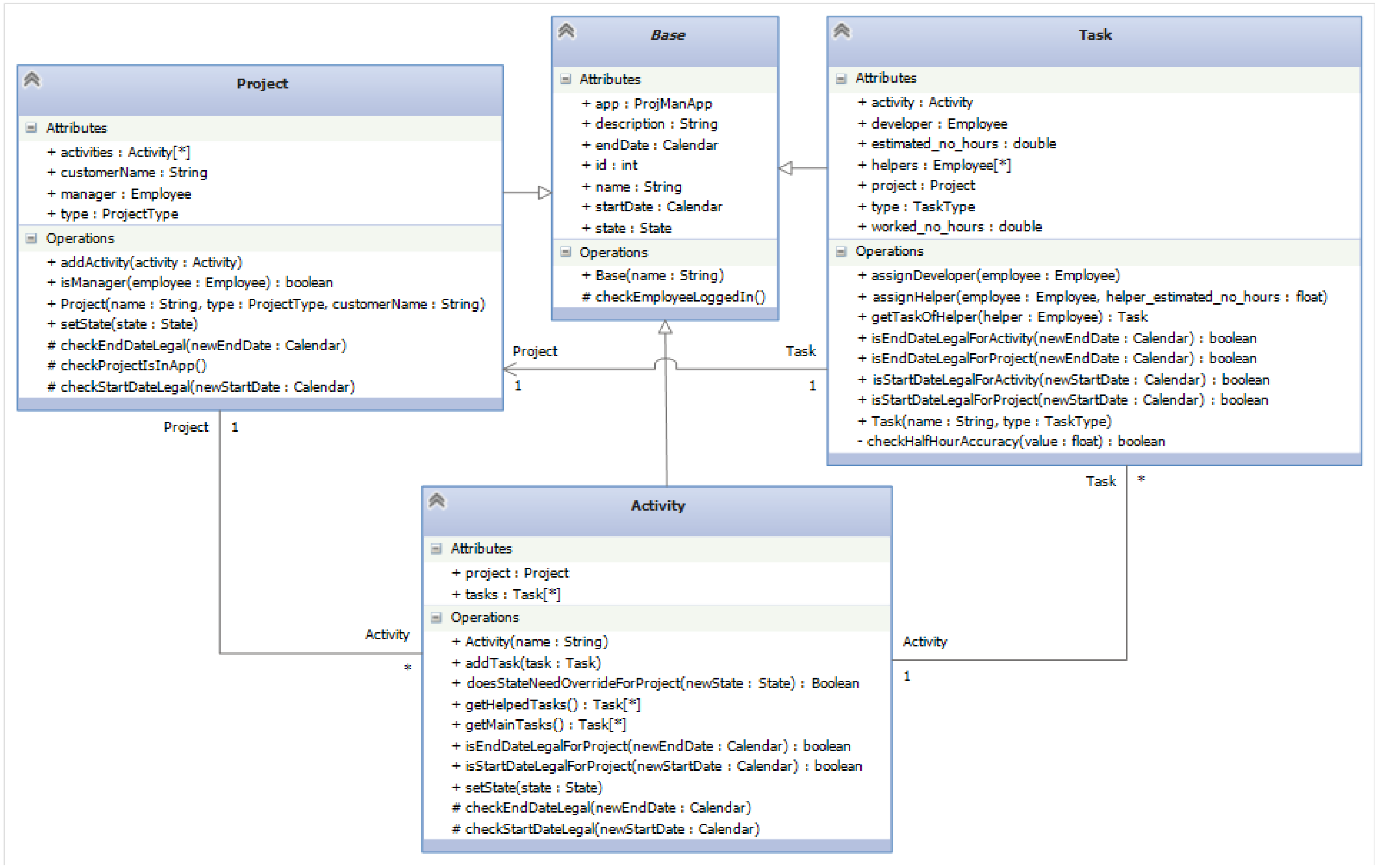
We have otherwise chosen to follow the priorities stated for the system, such that we implemented the project planning first, then the registration of hours and finally the project report. Our system also has a text based user interface, as there was no requirement that it was graphical, and we chose to use the time on the functionality of the product instead.

3 Program design

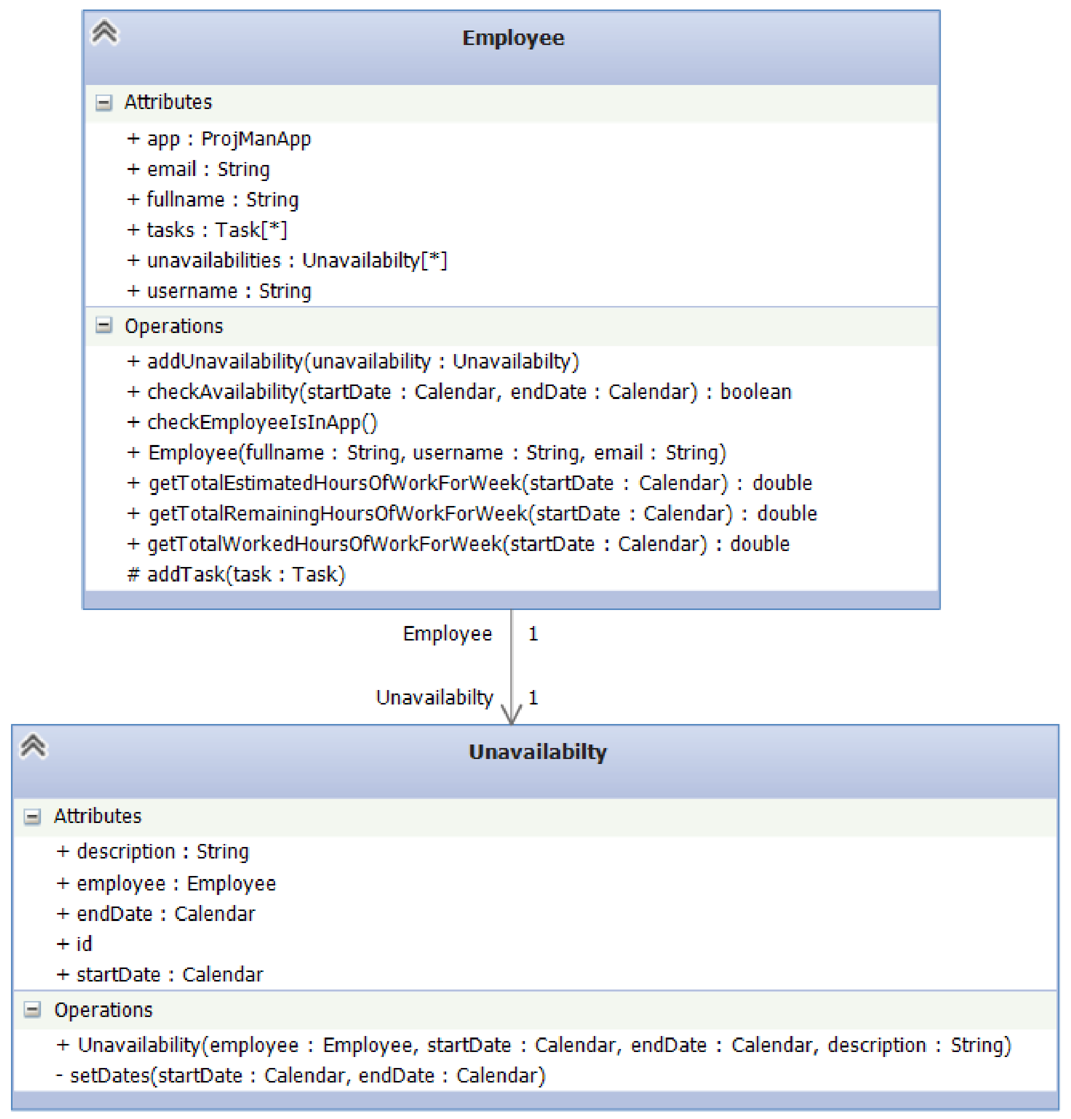
3.1 Class diagram



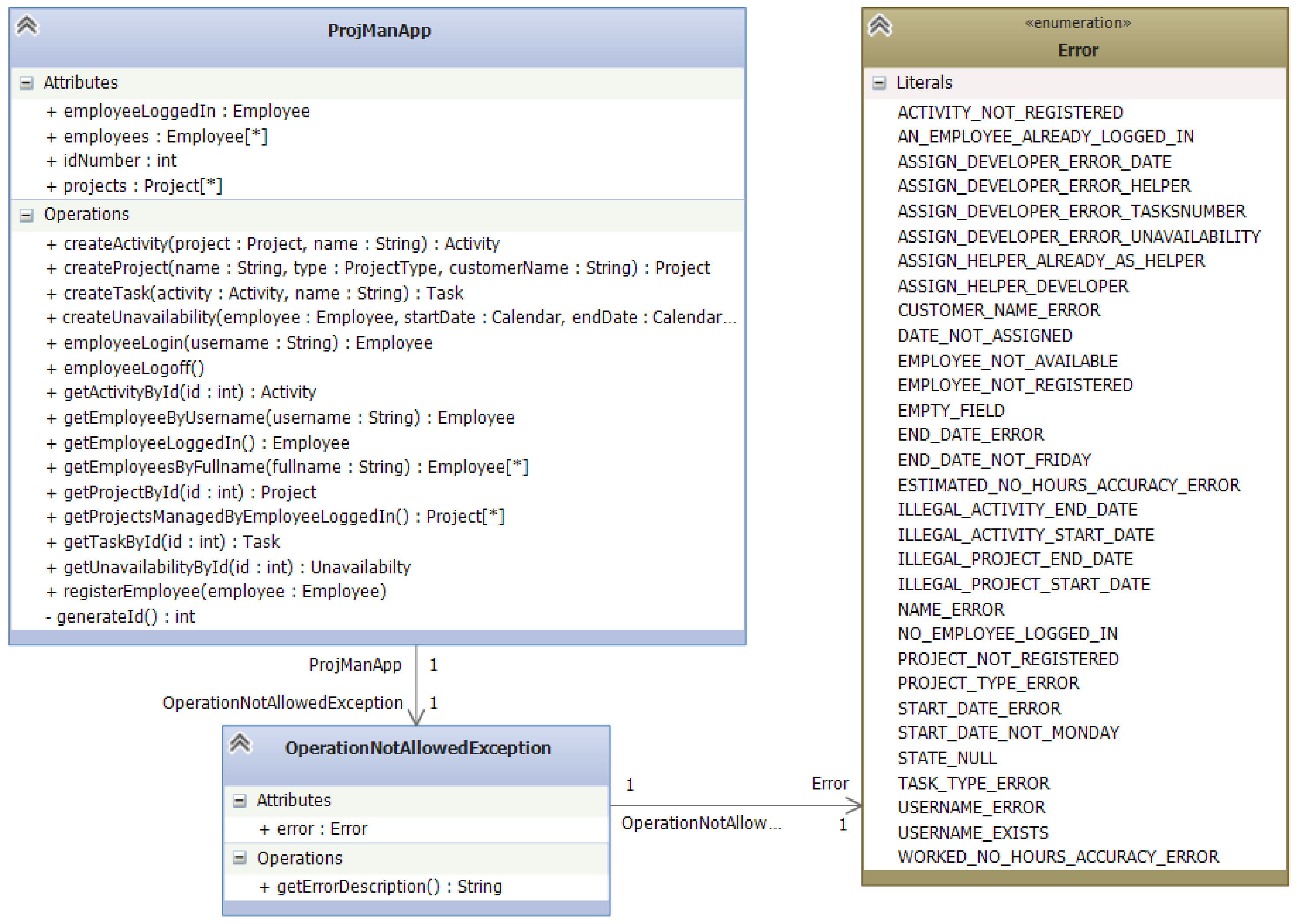
Figur 6: Overall class diagram with associations summarized. (Atakan Kaya)


Project
Attributes
+ activities : Activity[*]
+ customerName : String
+ manager : Employee
+ type : ProjectType
Operations
+ addActivity(activity : Activity)
+ isManager(employee : Employee) : boolean
+ Project(name : String, type : ProjectType, customerName : String)
+ setState(state : State)
# checkEndDateLegal(newEndDate : Calendar)
# checkProjectIsInApp()
# checkStartDateLegal(newStartDate : Calendar)
Base
Attributes
+ app : ProjManApp
+ description : String
+ endDate : Calendar
+ id : int
+ name : String
+ startDate : Calendar
+ state : State
Operations
+ Base(name : String)
# checkEmployeeLoggedIn()
Task
Attributes
+ activity : Activity
+ developer : Employee
+ estimated_no_hours : double
+ helpers : Employee[*]
+ project : Project
+ type : TaskType
+ worked_no_hours : double
Operations
+ assignDeveloper(employee : Employee)
+ assignHelper(employee : Employee, helper_estimated_no_hours : float)
+ getTaskOfHelper(helper : Employee) : Task
+ isEndDateLegalForActivity(newEndDate : Calendar) : boolean
+ isEndDateLegalForProject(newEndDate : Calendar) : boolean
+ isStartDateLegalForActivity(newStartDate : Calendar) : boolean
+ isStartDateLegalForProject(newStartDate : Calendar) : boolean
+ Task(name : String, type : TaskType)
- checkHalfHourAccuracy(value : float) : boolean
Project
1
Task
1
Project
1
Activity
*
Task
*
Activity
1
Activity
Attributes
+ project : Project
+ tasks : Task[*]
Operations
+ Activity(name : String)
+ addTask(task : Task)
+ doesStateNeedOverrideForProject(newState : State) : Boolean
+ getHelpedTasks() : Task[*]
+ getMainTasks() : Task[*]
+ isEndDateLegalForProject(newEndDate : Calendar) : boolean
+ isStartDateLegalForProject(newStartDate : Calendar) : boolean
+ setState(state : State)
# checkEndDateLegal(newEndDate : Calendar)
# checkStartDateLegal(newStartDate : Calendar)

Figur 7: Class diagram for Base, Project, Activity and Task. (Atakan Kaya)

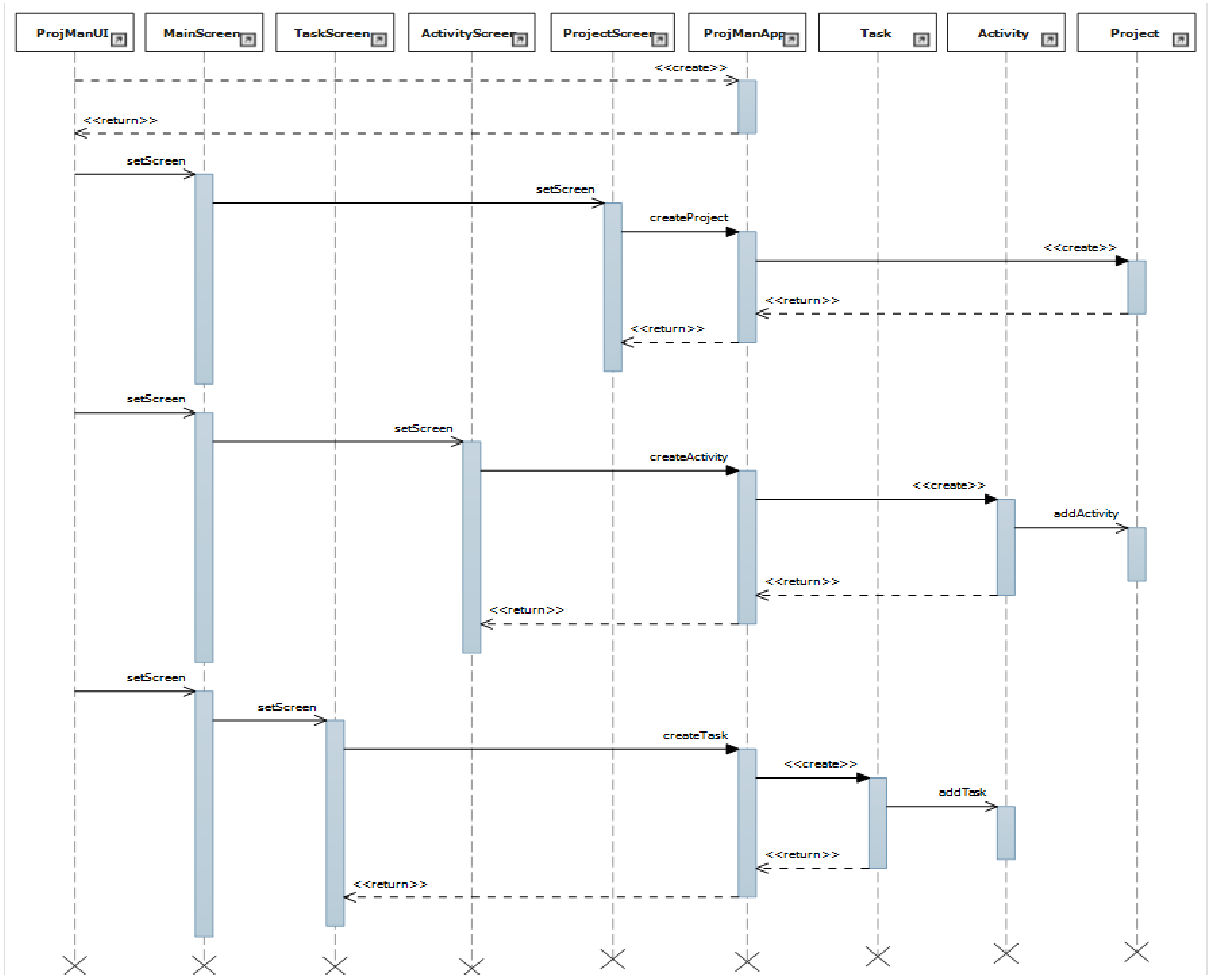

Employee
Attributes
+ app : ProjManApp
+ email : String
+ fullname : String
+ tasks : Task[*]
+ unavailabilities : Unavailabilty[*]
+ username : String
Operations
+ addUnavailability(unavailability : Unavailabilty)
+ checkAvailability(startDate : Calendar, endDate : Calendar) : boolean
+ checkEmployeeIsInApp()
+ Employee(fullname : String, username : String, email : String)
+ getTotalEstimatedHoursOfWorkForWeek(startDate : Calendar) : double
+ getTotalRemainingHoursOfWorkForWeek(startDate : Calendar) : double
+ getTotalWorkedHoursOfWorkForWeek(startDate : Calendar) : double
# addTask(task : Task)
Employee
1
Unavailabilty
1
Unavailabilty
Attributes
+ description : String
+ employee : Employee
+ endDate : Calendar
+ id
+ startDate : Calendar
Operations
+ Unavailability(employee : Employee, startDate : Calendar, endDate : Calendar, description : String)
- setDates(startDate : Calendar, endDate : Calendar)

Figur 8: Class diagram for Employee and Unavailability. (Atakan Kaya)

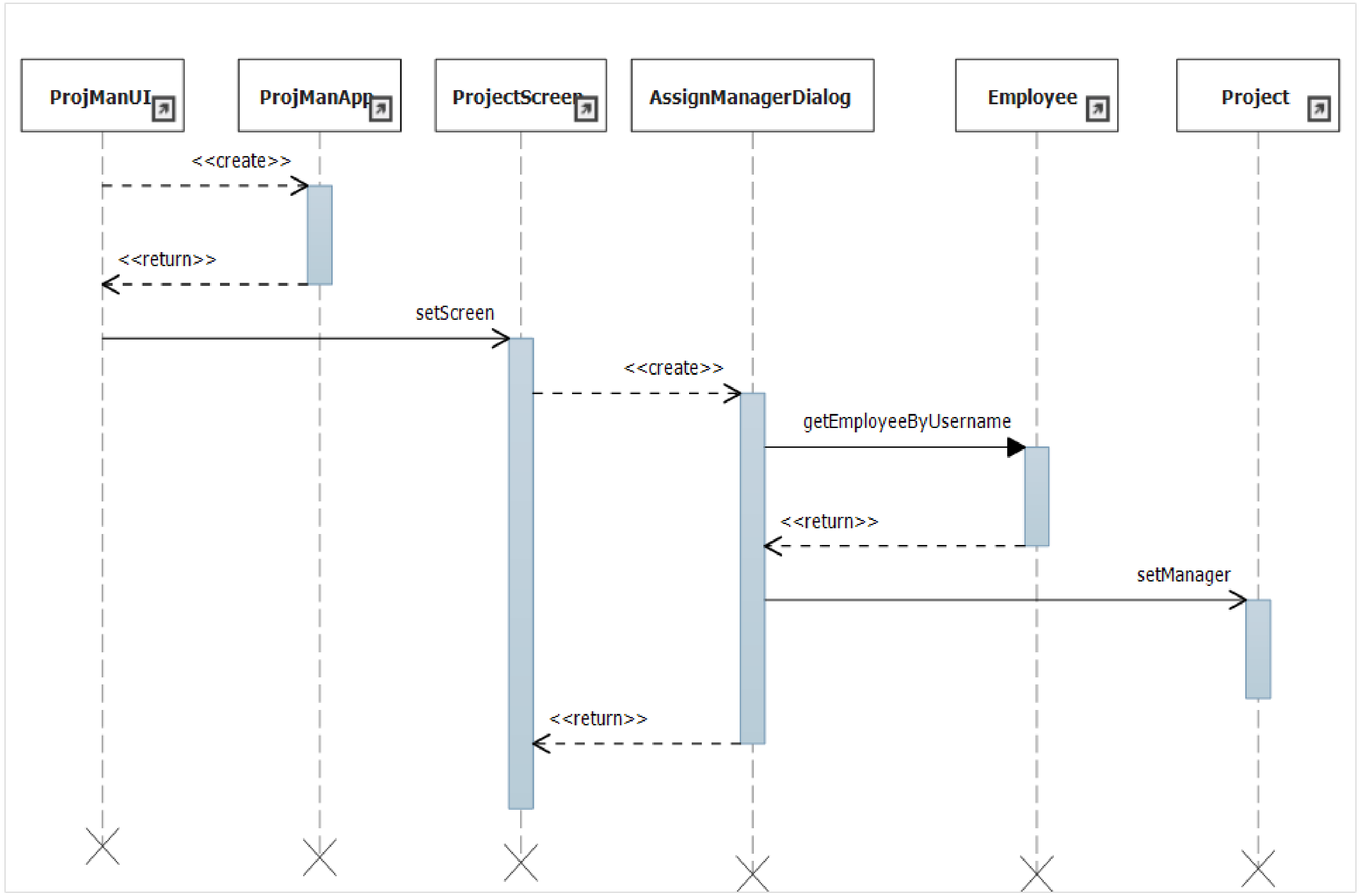

ProjManApp
Attributes
+ employeeLoggedIn : Employee
+ employees : Employee[*]
+ idNumber : int
+ projects : Project[*]
Operations
+ createActivity(project : Project, name : String) : Activity
+ createProject(name : String, type : ProjectType, customerName : String) : Project
+ createTask(activity : Activity, name : String) : Task
+ createUnavailability(employee : Employee, startDate : Calendar, endDate : Calendar...
+ employeeLogin(username : String) : Employee
+ employeeLogoff()
+ getActivityById(id : int) : Activity
+ getEmployeeByUsername(username : String) : Employee
+ getEmployeeLoggedIn() : Employee
+ getEmployeesByFullname(fullname : String) : Employee[*]
+ getProjectById(id : int) : Project
+ getProjectsManagedByEmployeeLoggedIn() : Project[*]
+ getTaskById(id : int) : Task
+ getUnavailabilityById(id : int) : Unavailabilty
+ registerEmployee(employee : Employee)
- generateId() : int
ProjManApp
1
OperationNotAllowedException
1
OperationNotAllowedException
Attributes
+ error : Error
Operations
+ getErrorDescription() : String
1
Error
OperationNotAllow...
1
«enumeration»
Error
Literals
ACTIVITY_NOT_REGISTERED
AN_EMPLOYEE_ALREADY_LOGGED_IN
ASSIGN_DEVELOPER_ERROR_DATE
ASSIGN_DEVELOPER_ERROR_HELPER
ASSIGN_DEVELOPER_ERROR_TASKSNUMBER
ASSIGN_DEVELOPER_ERROR_UNAVAILABILITY
ASSIGN_HELPER_ALREADY_AS_HELPER
ASSIGN_HELPER_DEVELOPER
CUSTOMER_NAME_ERROR
DATE_NOT_ASSIGNED
EMPLOYEE_NOT_AVAILABLE
EMPLOYEE_NOT_REGISTERED
EMPTY_FIELD
END_DATE_ERROR
END_DATE_NOT_FRIDAY
ESTIMATED_NO_HOURS_ACCURACY_ERROR
ILLEGAL_ACTIVITY_END_DATE
ILLEGAL_ACTIVITY_START_DATE
ILLEGAL_PROJECT_END_DATE
ILLEGAL_PROJECT_START_DATE
NAME_ERROR
NO_EMPLOYEE_LOGGED_IN
PROJECT_NOT_REGISTERED
PROJECT_TYPE_ERROR
START_DATE_ERROR
START_DATE_NOT_MONDAY
STATE_NULL
TASK_TYPE_ERROR
USERNAME_ERROR
USERNAME_EXISTS
WORKED_NO_HOURS_ACCURACY_ERROR

Figur 9: Class diagram for ProjManApp, OperationNotAllowedException and Error. (Atakan Kaya)

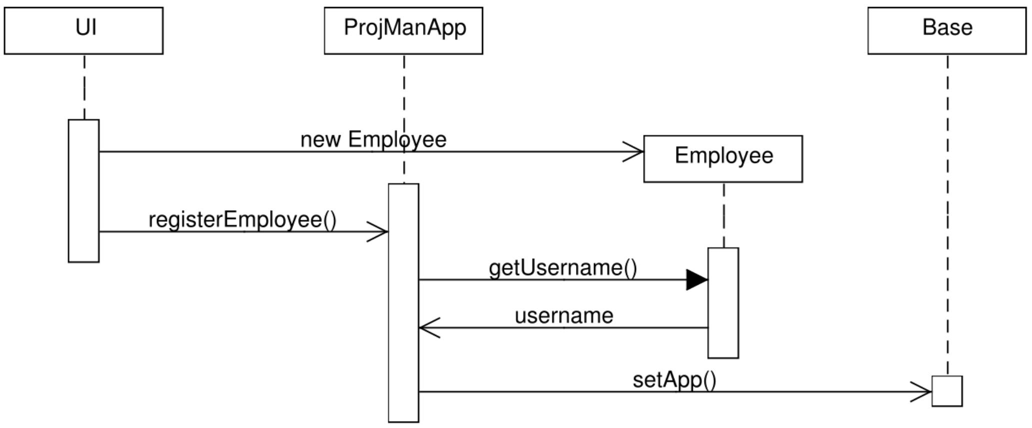
3.2 Sequence diagrams



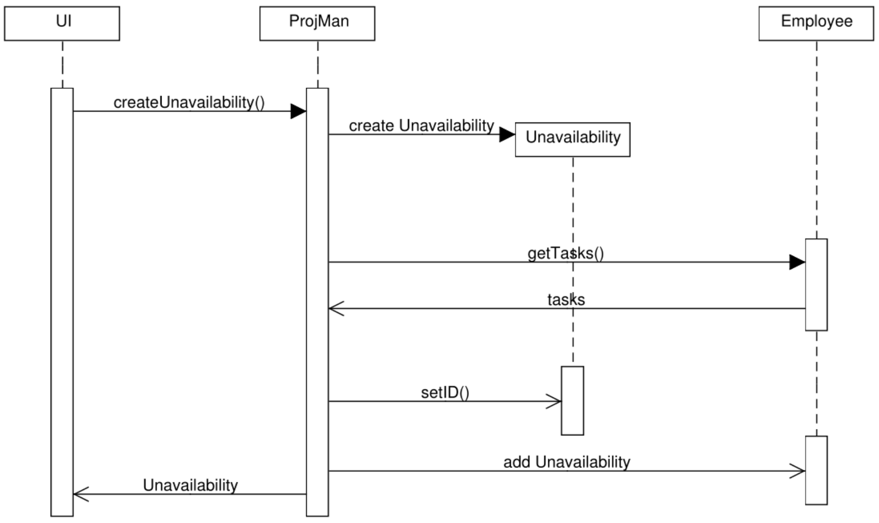
Figur 10: Sequence diagram for Create Task. (Atakan Kaya)



Figur 11: Sequence diagram for Assign Manager. (Atakan Kaya)

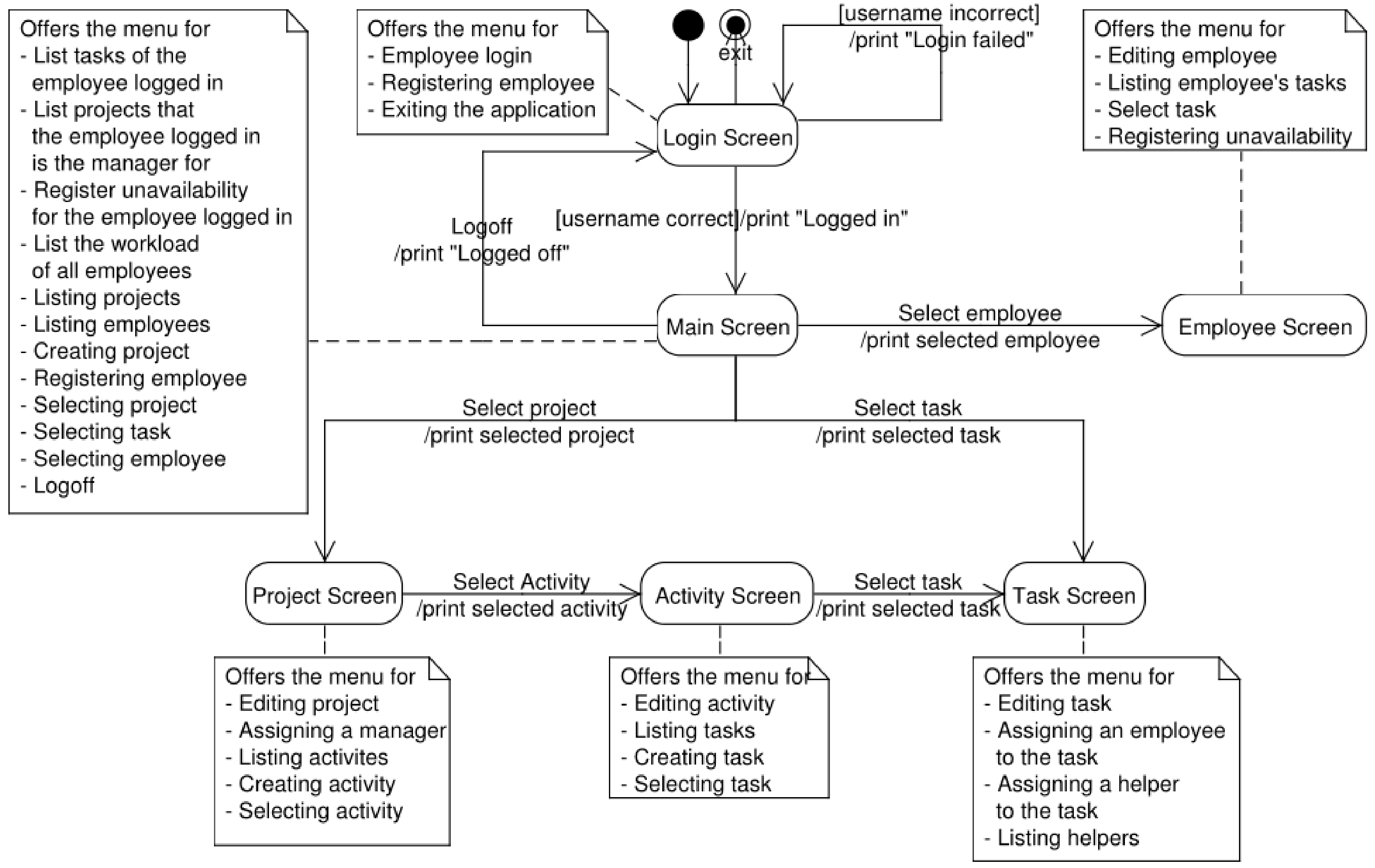


Figur 12: Sequence diagram for Register Employee. (Marianne Louis-Hansen)



Figur 13: Sequence diagram for Register Unavailability. (Marianne Louis-Hansen)

3.3 State Machine for User Interface



Figur 14: State Machine for user interface. (Atakan Kaya)

3.4 Discussion

by Atakan Kaya

As it can be seen from the Figure 6, this project is modelled with 7 classes, 1 abstract class and 4 enumerations. The abstract class Base has been created since some of the fields and the implementation of some methods were being repeated in Project, Activity and Task classes. This choice was done to overcome the redundancy. State enumeration is defined since each of these 3 classes have the one of the states illustrated (with the default value NOTSTARTED). Project Type includes if a project is internal, that is, the project is being developed for Software House A/S and if a project is external, that is, the project is being developed for another company or a customer. Task Type, on the other hand, keeps the information if a task is Main or Helped. If an employee seeks help from another employee, system creates a new task with the type HELPED.

The start and end dates for classes derived from Base have to be Monday of a week and Friday of a week, respectively. This is an implementation restriction since the customer wanted to keep track of the projects in the week number level. However, if customer wants to change remove this restriction, this can be easily achieved in the project since the system keeps them as dates not week numbers. Detailed illustration for Base, Project, Activity and Task classes can be seen in Figure 7.

Employee and Unavailability class together hold the information for employees, their tasks and their unavailabilities. Detailed illustration can be seen in Figure 8. Note that an employee can only register unavailabilities in week level. The choice for that is made since each project's, activity's and task's start and end date denotes week level.

Figure 9 illustrates the main application class, that is the interface through the user interface layer. It gives exceptions in problematic cases. The Exception class keeps the error message by an Error enumeration.

This section also includes sequence diagrams and a state machine. Sequence diagrams illustrate the workthrough between entities. State machine (Figure 14) is created to implement the user interface.

It can be seen that the system generates an id for Employee, Unavailability and the classes derived from Base. This functionality has been added reluctantly since id generation should have been left to the database management system. Since this project does not have a persistency layer, id generation has been implemented. Furthermore, in the project description it is stated that "Project ids have the form year and a serial number, for example 030901". However, this specification has deliberately not taken into account since the system already keeps the dates of the project and the user interface uses id numbers to select entities and that would lower the user experience through interface layer.

In the implementation, because of the time constraints generally the simplest algorithm has been chosen. However, it is presumed that most of the method's complexities are O(n).

4 Systematic tests

by Filipe Silva

For testing the program the acceptance tests were initiated as the program has been developed. To that, the code for the acceptance tests was done and then the source code was done in a way that all the tests passed (Test Driven Development). These tests obtained a 82 percent code coverage, which is a good value if we take into account the size of the source code and that some of the source code isn't a critical section.

After the acceptance test we did the Functional Tests. For that, 8 use case scenarios were chosen. This use case scenarios need to be well known in order to do this type of tests. Finally, the input values were chosen as extreme values for each case, being more likely to find error with these values. Final code coverage of systematic tests is 75 percent. That's due to the fact that a subset of use cases has been chosen to initiate the systematic tests.

The code coverage for both systematic and acceptance tests can be found in Appendix.

Besides the automatic JUnit tests, the manual tests to User Interface were done. For these tests, the use case scenarios were taken into account. They were used as guide for the manual tests since all the use case scenarios were done through the User Interface. These tests permitted to find some bugs in the User Interface, mainly related with some function that were not accessible for the User through the UI and some layout problems.

4.1 System Login Systematic Tests

by Marc Thomsen

Tabel 1: System Login Systematic Tests

Input data set	**Input Property**
A1	Username invalid (null), no employee is logged in
A2	Username invalid (empty string), no employee is logged in
B1	Username invalid (null), an employee is already logged in
B2	Username invalid (empty string), an employee is already logged in
C	Username valid, an employee is already logged in
D	Username valid, no employee is logged in

Tabel 2: System Login Systematic Tests

Input data set	**Contents**	**Expected Output**
A1	{null, false}	USERNAME_ERROR
A2	{”, false}	USERNAME_ERROR
B1	{null, true}	USERNAME_ERROR
B2	{”, true}	USERNAME_ERROR
C	{’Emp’, false}	AN_EMPLOYEE_ALREADY_LOGGED_IN
D	{’Emp’, true}	Login succesful

4.2 Register Employee Systematic Tests

by Marianne Louis-Hansen

Tabel 3: Register Employee Systematic Tests

Input data set	Input Property
A1	Full name valid, username invalid (null), email address valid
A2	Full name valid, username invalid (empty string), email address valid
A3	Full name valid, username invalid (longer than four characters), email address valid
B1	Full name invalid (null), username valid, email address valid
B2	Full name invalid (empty string), username valid, email address valid
C1	Full name valid, username valid, email address invalid (null)
C2	Full name valid, username valid, email address invalid (empty string)
D	Full name valid, username valid, email address valid
E1	(Pre: Username exists) Full name invalid (empty string), username valid, email address valid
E2	(Pre: Username exists) Full name valid, username valid, email address invalid (empty string)
E3	(Pre: Username exists) Full name valid, username valid, email address valid

Tabel 4: Register Employee Systematic Tests

Input data set	Contents	Expected Output
A1	{'Jane Doe', null,'jane@doe.com'}	USERNAME_ERROR
A2	{'Jane Doe', ", 'jane@doe.com'}	USERNAME_ERROR
A3	{'Jane Doe', 'janedoe', 'jane@doe.com'}	USERNAME_ERROR
B1	{null, 'jdoe', 'jane@doe.com'}	EMPTY_FIELD
B2	{", 'jdoe', 'jane@doe.com'}	EMPTY_FIELD
C1	{'Jane Doe', 'jdoe', null}	EMPTY_FIELD
C2	{'Jane Doe', 'jdoe', "}	EMPTY_FIELD
D	{'Jane Doe', 'jdoe', 'jane@doe.com'}	Employee added

Continued on next page

Tabel 4 – continued from previous page

Input data set	Contents	Expected Output
E1	{”, ’jdoe’, ’jane@doe.com’}	EMPTY_FIELD
E2	{’Jane Doe’, ’jdoe’, ”}	EMPTY_FIELD
E3	{’Jane Doe’, ’jdoe’, ’jane@doe.com’}	USERNAME_EXISTS

4.3 Create Project Systematic Tests

by Atakan Kaya

Tabel 5: Create Project Systematic Tests

Input data set	Input Property
A1	Name invalid (null), type valid, customerName valid
A2	Name invalid (empty string), type valid, customerName valid
B	Name valid, type invalid (null), customerName valid
C1	Name valid, type valid (Internal), customerName invalid
C2	Name valid, type valid (External), customerName invalid (null)
C3	Name valid, type valid (External), customerName invalid (empty string)
D1	Name invalid, type invalid, customerName valid
D2	Name invalid, type valid, customerName invalid
D3	Name valid, type invalid, customerName invalid
E	Name invalid, type invalid, customerName invalid
F1	Name valid, type valid (Internal), customerName valid
F2	Name valid, type valid (External), customerName valid

Tabel 6: Create Project Systematic Tests

Input data set	**Contents**	**Expected Output**
A1	{null, ProjectType.EXTERNAL, 'Microsoft'}	NAME_ERROR, Project = null
A2	{", ProjectType.EXTERNAL, 'Microsoft'}	NAME_ERROR, Project = null
B	{'ProjMan', null, 'Microsoft'}	PROJECT_TYPE_ERROR, Project = null
C1	{'ProjMan', ProjectType.INTERNAL, "}	Project({'ProjMan', ProjectType.INTERNAL, 'Software House A/S'})
C2	{'ProjMan', ProjectType.EXTERNAL, null}	CUSTOMER_NAME_ERROR, Project = null
C3	{'ProjMan', ProjectType.EXTERNAL, "}	CUSTOMER_NAME_ERROR, Project = null
D1	{", null, 'Microsoft'}	NAME_ERROR, Project = null
D2	{", ProjectType.EXTERNAL , "}	NAME_ERROR, Project = null
D3	{'ProjMan', null, "}	PROJECT_TYPE_ERROR, Project = null
E	{", null, "}	NAME_ERROR, Project = null
F1	{'ProjMan', ProjectType.INTERNAL, 'Microsoft'}	Project({'ProjMan', ProjectType.INTERNAL, 'Software House A/S'})
F2	{'ProjMan', ProjectType.EXTERNAL, 'Microsoft'}	Project({'ProjMan', ProjectType.EXTERNAL, 'Microsoft'})

4.4 Edit Project Systematic Tests

by Marianne Louis-Hansen

Tabel 7: Edit Project Systematic Tests

Input data set	**Input Property**
A1	Name invalid(null)
A2	Name invalid(empty string)
A3	Name valid
B1	Type invalid
B2	Type valid
C1	Start date invalid (not a monday)
C2	Start date valid
D1	End date invalid (not a friday)
D2	End date valid

Tabel 8: Edit Project Systematic Tests

Input data set	**Contents**	**Expected Output**
A1	null	NAMER_ERROR
A2	{”}	NAMER_ERROR
A3	{ProjMan}	Changes are saved
B1	{”}	PROJECT_TYPE_ERROR
B2	ProjectType.EXTERNAL	Changes are saved
C1	(2013, Calendar.MAY,06)	START_DATE_NOT_MONDAY_ERROR
C2	(2013, Calendar.APRIL,29)	Changes are saved
D1	(2013, Calendar.MAY,12)	END_DATE_NOT_FRIDAY_ERROR
D2	(2013, Calendar.MAY,10)	Changes are saved

4.5 Assign Project Manager Systematic Tests

by Marianne Louis-Hansen

Tabel 9: Assign Project Manager Systematic Tests

Input data set	Input Property
A1	Username invalid (null)
A2	Username invalid (not registered employee)
A3	Valid username (no existing manager
A4	Valid username (overwriting existing manager)

Tabel 10: Assign Project Manager Systematic Tests

Input data set	Contents	Expected Output
A1	{null}	NAME_ERROR
A2	{jodo}	EMPLOYEE_NOT_REGISTERED
A3	{jdoe}	Changes are saved
A4	{mdoe}	Changes are saved

4.6 Edit Task Systematic Tests

by Atakan Kaya

Tabel 11: Edit Task Systematic Tests

Input data set	Input Property
A1	Name invalid (null)
A2	Name invalid (Empty string)
A3	Name valid
B	Description valid

Continued on next page

Tabel 11 – continued from previous page

Input data set	Input Property
C1	Expected number of hours to solve task invalid (negative)
C2	Expected number of hours to solve task invalid (does not have half hour accuracy)
C3	Expected number of hours to solve task valid
D1	Number of hours worked for the task invalid (negative)
D2	Number of hours worked for the task invalid (does not have half hour accuracy)
D3	Number of hours worked for the task valid
E1	Start date illegal (not Monday)
E2	Start date illegal (later than end date)
E3	The start date of the task is earlier than the start date of the project that the task belongs to. (Don't overwrite)
E4	The start date of the task is earlier than the start date of the project that the task belongs to. (overwrite)
E5	The start date of the task is earlier than the start date of its activity.) (Don't overwrite
E6	The start date of the task is earlier than the start date of its activity. (overwrite)
E7	Start date legal
F1	End date illegal (not Friday)
F2	End date illegal (earlier than start date)
F3	The end date of the task is later than the end date of the project that the task belongs to. (Don't overwrite)
F4	The end date of the task is later than the end date of the project that the task belongs to. (overwrite)
F5	The end date of the task is later than the end date of its activity. (Don't overwrite)
F6	The end date of the task is later than the end date of its activity. (overwrite)

Continued on next page

Tabel 11 – continued from previous page

Input data set	Input Property
F7	End date legal

Tabel 12: Edit Task Systematic Tests

Input data set	Contents	Expected Output
A1	null	NAME_ERROR
A2	”	NAME_ERROR
A3	’Implement systematic tests’	Name changed
B	’First create tables then implement’	Description changed
C1	-5	ESTIMATED_NO_HOURS_ACCURACY_ERROR
C2	10.2	ESTIMATED_NO_HOURS_ACCURACY_ERROR
C3	10.5	Estimated no hours changed
D1	-5	WORKED_NO_HOURS_ACCURACY_ERROR
D2	10.2	WORKED_NO_HOURS_ACCURACY_ERROR
D3	10.5	Worked no hours changed
E1	1/5/2013	START_DATE_NOT_MONDAY
E2	29/4/2013 (Pre: enddate=26/4/2013)	START_DATE_ERROR
E3	29/4/2013 (Pre: project _startdate=6/5/2013)	isStartDateLegalForProject returns false
E4	29/4/2013 (Pre: project _startdate=6/5/2013)	isStartDateLegalForProject returns false, new start date of task, its activity and its project: 29/4/2013
E5	29/4/2013 (Pre: activity _startdate=6/5/2013)	isStartDateLegalForActivity returns false
E6	29/4/2013 (Pre: activity	isStartDateLegalForActivity returns false,

Continued on next page

Tabel 12 – continued from previous page

Input data set	Contents	Expected Output
	_startdate=6/5/2013)	new start date of task, its activity: 29/4/2013
E7	22/4/2013	Start date changed
F1	15/4/2013	END_DATE_NOT_FRIDAY
F2	3/5/2013 (Pre: startdate=6/5/2013)	END_DATE_ERROR
F3	3/5/2013 (Pre: project _enddate =19/4/2013)	isEndDateLegalForProject returns false
F4	3/5/2013 (Pre: project _enddate=19/4/2013)	isEndDateLegalForProject returns false, new end date of task, its activity and its project: 3/5/2013
F5	3/5/2013 (Pre: activity _enddate =19/4/2013)	isEndDateLegalForActivity returns false
F6	3/5/2013 (Pre: activity _enddate =19/4/2013)	isEndDateLegalForActivity returns false, new end date of task, its activity: 3/5/2013
F7	1/6/2013	End date changed

4.7 Assign An Employee to A Task Systematic Tests

by Marc Thomsen

Tabel 13: Assign An Employee to A Task Systematic Tests

Input data set	Input Property
A	The assigned employee is unavailable
B	The assigned employee is available, employee is a helper
C1	The assigned employee is available, tasks = 20, employee is already a helper
C2	The assigned employee is available, tasks = 20, employee is not a helper
D1	The assigned employee is available, tasks < 20, employee is already a helper

Continued on next page

Tabel 13 – continued from previous page

Input data set	**Input Property**
D2	The assigned employee is available, tasks < 20, employee is not a helper

Tabel 14: Assign An Employee to A Task Systematic Tests

Input data set	**Contents**	**Expected Output**
A	{createUnavailability for same week }	ASSIGN_DEVELOPER_ERROR _UNAVAILABILITY
B	{createUnavailability for another week, assignHelper }	ASSIGN_DEVELOPER_ERROR _AS_HELPER
C1	{createUnavailability for another week, tasks().size=20, assignHelper }	ASSIGN_DEVELOPER_ERROR _TASKSNUMBER
C2	{createUnavailability for another week, tasks().size=20 }	ASSIGN_DEVELOPER_ERROR _TASKSNUMBER
D1	{createUnavailability for another week, tasks().size<20, assignHelper }	ASSIGN_DEVELOPER_ALREADY ASSIGN_DEVELOPER_ERROR _HELPER
D2	{createUnavailability for another week, tasks().size<20}	The employee has been successfully assigned to the task

4.8 Edit State Functionality Systematic Tests

4.8.1 Edit Project State Systematic Tests

by Filipe Silva

Tabel 15: Edit Project State Systematic Tests

Input data set	**Input Property**
A	States that changes the state of Activities and Tasks of the Project
B	States that don't change state of the Activities and Tasks of the Project

Tabel 16: Edit Project State Systematic Tests

Input data set	**Contents**	**Expected Output**
A1	{State.NOTSTARTED}	The state of the project, all the activities and tasks of it = State.NOTSTARTED
A2	{State.PAUSED}	The state of the project, all the activities and tasks of it = State.PAUSED
A3	{State.FINISHED}	The state of the project, all the activities and tasks of it = State.FINISHED
B	{State.ONGOING}	The state of the project = State.ONGOING

4.8.2 Edit Activity State Systematic Tests

by Filipe Silva

Tabel 17: Edit Activity State Systematic Tests

Input data set	**Input Property**
A	States that changes the state of the Tasks of the Activity
B	States that don't change the state of the Tasks or Project of the Activity
B1	Pre condition: Project.state = State.ONGOING
C	States that change the state of the Activity's Project

Continued on next page

Tabel 17 – continued from previous page

Input data set	Input Property
C1	Pre condition: Project.state = State.NOTSTARTED
C2	Pre condition: Project.state = State.PAUSED
C3	Pre condition: Project.state = State.FINISHED

Tabel 18: Edit Activity State Systematic Tests

Input data set	Contents	Expected Output
A1	{State.NOTSTARTED}	The state of the activity and all the tasks of it = State.NOTSTARTED
A2	{State.PAUSED}	The state of the activity and all the tasks of it = State.PAUSED
A3	{State.FINISHED}	The state of the activity and all the tasks of it = State.FINISHED
B	{State.ONGOING}	The state of the activity = State.ONGOING
C1	{State.ONGOING}	The state of the activity and its project = State.ONGOING
C2	{State.ONGOING}	The state of the activity and its project = State.ONGOING
C3	{State.ONGOING}	The state of the activity and its project = State.ONGOING

4.8.3 Edit Task State Systematic Tests

by Filipe Silva

Tabel 19: Edit Task State Systematic Tests

Input data set	**Input Property**
A	States that don't change the state of the Task's Activity or Project
A1	Pre condition: Project.state and Activity.state= State.ONGOING
B	States that change the state of the Task's Project
B1	Pre condition: Project.state = State.NOTSTARTED
B2	Pre condition: Project.state = State.PAUSED
B3	Pre condition: Project.state = State.FINISHED
C	States that change the state of the Task's Activity
C1	Pre condition: Activity.state = State.NOTSTARTED
C2	Pre condition: Activity.state = State.PAUSED
C3	Pre condition: Activity.state = State.FINISHED

Tabel 20: Edit Task State Systematic Tests

Input data set	**Contents**	**Expected Output**
A1	{State.NOTSTARTED}	The state of the task = State.NOTSTARTED
A2	{State.PAUSED}	The state of the task = State.PAUSED
A3	{State.FINISHED}	The state of the task = State.FINISHED
A4	{State.ONGOING}	The state of the task = State.ONGOING
B1	{State.ONGOING}	The state of the task and its project = State.ONGOING
B2	{State.ONGOING}	The state of the task and its project = State.ONGOING
B3	{State.ONGOING}	The state of the task and its project = State.ONGOING
C1	{State.ONGOING}	The state of the task and its activity = State.ONGOING
C2	{State.ONGOING}	The state of the task and

Continued on next page

Tabel 20 – continued from previous page

Input data set	Contents	Expected Output
C3	{State.ONGOING}	its activity = State.ONGOING The state of the task and its activity = State.ONGOING

4.9 Register Unavailability Systematic Tests

by Filipe Silva

Tabel 21: Register Unavailability Systematic Tests

Input data set	Input Property
A	All the fields are correct
A1	Register one Unavailability
A2	Register two Unavailabilities
B	Start date is not the Monday of a week
B1	Start date is a Sunday
B2	Start date is a Thuesday
C	End date is not a Friday of a week
C1	End date is a Saturday
C2	End date is a Thursday
D	End date is before than start date
D1	End date is one week before
D2	End date is two weeks before
E	Employee has tasks for the period of the unavailability
E1	Employee has one task for the period of the unavailability
E2	Employee has two tasks for the period of the unavailability
F	Employee has unavailabilities for the period of the unavailability
F1	Employee has one another unavailability for the period of the unavailability

Continued on next page

Tabel 21 – continued from previous page

Input data set	Input Property
F2	Employee has two another unavailabilities for the period of the unavailability

Tabel 22: Register Unavailability Systematic Tests

Input data set	Contents	Expected Output
A	{Employee, 6/5/2013, 10/5/2013, 'Reason'}	Unavailability registed in the employee's list of unavailabilities.
B1	{Employee, 5/5/2013, 10/5/2013, 'Reason'}	Unavailability not saved and the employee is informed about the wrong start date.
B2	{Employee, 7/5/2013, 10/5/2013, 'Reason'}	Unavailability not saved and the employee is informed about the wrong start date.
C1	{Employee, 6/5/2013, 11/5/2013, 'Reason'}	Unavailability not saved and the employee is informed about the wrong end date.
C2	{Employee, 6/5/2013, 9/5/2013, 'Reason'}	Unavailability not saved and the employee is informed about the wrong end date.
D1	{Employee, 13/5/2013, 6/5/2013, 'Reason'}	Unavailability not saved and the employee is informed about the wrong dates.
D2	{Employee, 13/5/2013, 29/4/2013, 'Reason'}	Unavailability not saved and the employee is informed about the wrong dates.
E	{Employee, 29/4/2013, 10/5/2013, 'Reason'}	Unavailability not saved and the employee is informed about having tasks for that period.
F	{Employee, 29/4/2013, 10/5/2013, 'Reason'}	Unavailability not saved and the employee is informed about having unavailabilities for that period.

5 Discussion

For this project concerning the development of a planning- and scheduling tool different software development processes (methods) exists. As presented in the lecture for this course the following methods is widely used in software companies.

The Waterfall Model

This model was originally a hardware-oriented model and was presented first time in 1956. It has ever since been subject to criticism because of its sequential design process. It does not take into account that the process can be dynamic and the specification of the different phases in the process can change over time. The supporting argument for this process is that the time spent early by ensuring that the requirements and design are correct can lead to greater economy later in the process.

The Iterative development process

This proces focus on developing a given system through repeated cycles and in smaller fragments one by one. The process contains three steps: the initialization step, the iteration step and the Project Control list. The advantage of this method is that the learning comes from both the development and the use of the system and therefore earlier versions of the system is an important source of learning.

The Agile development process

In this group the agile development process was chosen. In both the original project plan made before the beginning of the project period and the final project plan this process can be identified. When the project plan was to be made an agreement was reached about the priority of the functionalities specified in the project description. This prioritizing was based on the Resource Triangle presented in the lecture in week 7. It focuses on the three major factors that exist during software development namely: *Quality*, *Time/Resources* and *Functionality* which is the three nodes forming the triangle. However the idea behind this concept is that only two of them can be fixed at the same time. Therefore it was decided in this group to give the parts *Quality* and *Time/Resources* high priority and the *Functionality*-part low priority. Based on this a subset of the functionalities was selected and hereafter listed by relevance in the project plan.

The interchanging between the use of the words "aktiviteter"and "opgaver"in the project description led to some confusion of the difference between activities and tasks for this group. That is the reason why the idea of dividing the **projects** into subprojects called **activities** and again dividing activities into subactivities called **tasks**.

With the Resource Triangle in mind the performing of the systematic tests of the already implemented functionalities was of higher priority than focusing on implementing all the functionalities specified in the project description. As a result of this choice of prioritizing it was realized that some of the latest functionalities in the project plan had to be excluded during the process. Furthermore in the original project plan it was intended that for every week report writing should be done. Although in the second week of the project period it was determined that writing the report is roughly comparable to the Waterfall process. The general expectation was that in the end of the project period a higher work load was to be scheduled to revise most of the sections in the report. Therefore all the tasks involving the report writing in the first four weeks was moved to the fifth and final week in the final plan.

6 Conclusion

by Filipe Silva

With this project we were able to learn another side of the software development. Until this course we only had experience in the code part of the software development. During this project we did all the testing, both acceptance and functional, and documentation, as use case scenarios or systematic tests tables. By doing this approach on the software development we learned a really important part when it comes to organize our software during the development phase. As development technique the use of the iteration process gave us the possibility of weekly re-think our priorities and change some details, both in the use case scenarios and source code. Besides that gave us a good weekly feedback about the state of the project.

The way we organize during this project is essential if we want to archive the trust, ease-of-use and stability in a software project.

A Appendix

A.1 Original Project Plan

PROJECT PLAN FOR GROUP 19

Number of persons	Hours a week per person	Hours a week	Number of weeks	Hours for the whole project
4	5.6	22.2	5	111

Week Number	User Story/Tasks	Ideal man hour	Man hour with load factor	Total hours in week	Total hours
Week 1 **08/04/2013 – 14/04/2013**	Creating the base structure of the report	0.5	1	1	1
	Create a glossary and define key concepts	1	2	3	3
	Define detailed use case scenarios and draw use case diagrams	2	4	7	7
	Register employee functionality	1	2	9	9
	Implement the system login functionality	0.5	1	10	10
	Writing the introduction	1	2	12	12
	Draw class diagrams	2	4	16	16
	Define detailed use case scenarios and draw use case diagrams	2	4	20	20
	Implement the creation of a project functionality	0.5	1	21	21
	Create an activity and add that activity to the project	0.5	1	22	22
	Create a task and add it to an activity	0.5	1	23	23
	Do the syst. tests for the use cases in this iteration	2	4	27	27
Week 2 **15/04/2013 – 21/04/2013**	Define detailed use case scenarios and draw use case diagrams	2	4	4	31
	Assign an employee as a project manager to a project	0.5	1	5	32
	Draw sequence diagrams	2	4	9	36
	Writing about the design	2	4	13	40
	Add start and end time for a project	0.5	1	14	41
	Add start and end time for an activity	0.5	1	15	42
	Add start and end time for a task	0.5	1	16	43

	Implement the assignment of an employee(s) to a task	0.5	1	17	44
	Implement the edit project state functionality	0.5	1	18	45
	Implement the edit activity state functionality	0.5	1	19	46
	Implement the edit task state functionality	0.5	1	20	47
	Do the syst. tests for the use cases in this iteration	3	6	26	53
Week 3 **22/04/2013 – 28/04/2013**	Define detailed use case scenarios and draw use case diagrams	1	2	2	55
	Implement the register of the estimated hours of work for a task	1	2	4	57
	Implement the list of the available employees in the week(s)	2	4	8	61
	Implement the register of the number of hours worked on a task	1	2	10	63
	Implement the register of the unavailability of employees (vacation, etc)	1	2	12	65
	Implement the assignment of helpers	1	2	14	67
	Do the syst. tests for the use cases in this iteration	3	6	20	73
Week 4 **29/04/2013 – 05/05/2013**	Define detailed use case scenarios and draw use case diagrams	2	4	4	77
	Implement the generation of an overview report on project (remaining time to finish the project, export text file)	3	6	10	83
	Implement the generation of a report on the weekly progress of the project	3	6	16	89
	Do the syst. tests for the use cases in this iteration	3	6	22	95
Week 5 **06/05/2013 – 12/05/2013**	Define detailed use case scenarios and draw use case diagrams	0.5	1	1	97
	Implement the “See the schedule of an employee” functionality	1	2	3	98
	Implement the “System Logoff” functionality	0.5	1	4	99
	Design a user interface (console)	5	10	14	109
	Write the conclusion of the report	1	2	16	111

Group 19:	Atakan Kaya s120872
	Filipe Silva s124755
	Marc Thomsen s093022
	Marianne Louis-Hansen s072188

A.2 Final Project Plan

Week Number	User Story/Tasks	Ideal man hour	Man hour with load factor	Total hours in week	Worked Time	Assigned to
Week 1 **08/04/2013 – 14/04/2013**	Creating the base structure of the report	0.5	1	1	1	Marianne
	Create a glossary and define key concepts	1	2	3	2	Marianne
	Define detailed use case scenarios and draw use case diagrams	2	4	7	4	Group
	Register employee functionality	1	2	9	2	Filipe
	Implement the system login functionality	0.5	1	10	1	Marc
	Draw class diagrams	2	4	16	4	Atakan
	Implement the creation of a project functionality	0.5	1	21	1	Filipe
	Create an activity and add that activity to the project	0.5	1	22	1	Atakan
	Create a task and add it to an activity	0.5	1	23	1	Atakan
	Do the syst. tests for the use cases in this iteration	2	4	27	2	Group
Week Number	**User Story/Tasks**	**Ideal man hour**	**Man hour with load factor**	**Total hours in week**	**Worked Time**	**Assigned to**
Week 2 **15/04/2013 – 21/04/2013**	Define detailed use case scenarios and draw use case diagrams	2	4	4	4	Group
	Assign an employee as a project manager to a project	0.5	1	5	1	Marianne
	Draw sequence diagrams	2	4	9	4	Group
	Add start and end time for a project	0.5	1	14	2	Atakan
	Add start and end time for an activity	0.5	1	15	3	Atakan
	Add start and end time for a task	0.5	1	16	3	Atakan
	Implement the assignment of an employee(s) to a task	0.5	1	17	1	Marianne
	Implement the edit project state functionality	0.5	1	18	2	Filipe

	Implement the edit activity state functionality	0.5	1	19	2	Filipe
	Implement the edit task state functionality	0.5	1	20	2	Filipe
	Implement the register of the estimated hours of work for a task	1	2	22	2	Marc
	Implement the register of the number of hours worked on a task	1	2	24	2	Marc
	Do the syst. tests for the use cases in this iteration	3	6	30	3	Group
Week Number	**User Story/Tasks**	**Ideal man hour**	**Man hour with load factor**	**Total hours in week**	**Worked Time**	**Assigned to**
Week 4 **29/04/2013 – 05/05/2013**	Define detailed use case scenarios and draw use case diagrams	3	6	**6**	3	Group
	Assign an employee as a project manager to a project	0.5	1	**7**	1	Marianne
	Implement the assignment of an employee(s) to a task	0.5	1	8	1	Marianne
	Implement state functionality	4	8	16	4	Filipe
	Implement the register of the unavailability of employees (vacation, etc)	2	4	20	2	Filipe
	Implement the list of the available employees in the week(s)	2	4	24	3	Atakan
	Implement the assignment of helpers	1	2	26	3	Marc
	Do the syst. tests for the use cases in this iteration	6	12	38	4	Group
Week Number	**User Story/Tasks**	**Ideal man hour**	**Man hour with load factor**	**Total hours in week**	**Worked Time**	**Assigned to**
Week 5 **06/05/2013 – 12/05/2013**	Define detailed use case scenarios and draw use case diagrams	0.5	1	1	1	Group
	Implement the “See the schedule of an employee” functionality	1	2	3	2	Marianne
	Implement the “System Logoff” functionality	0.5	1	4	0.5	Marc
	Do the syst. tests for the use cases in this iteration	2	4	8	1	Group
	Design a user interface (console)	5	10	18	10	Atakan
	Write the report	6	12	30	8	Group

A.3 Code Coverage Results

Tabel 23: Code Coverage for Acceptance Tests

Element	Coverage	Covered Inst	Missed Inst	Total Inst
dtu.projman.app	82,0	2.221	487	2.708
Employee.java	53,8	142	122	264
Employee	53,8	142	122	264
getTotalEstimatedHoursOfWorkForWeek(Calendar)	0,0	0	31	31
getTotalWorkedHoursOfWorkForWeek(Calendar)	0,0	0	31	31
checkAvailability(Calendar, Calendar)	0,0	0	27	27
getTotalRemainingHoursOfWorkForWeek(Calendar)	0,0	0	18	18
setEmail(String)	66,7	10	5	15
setFullname(String)	66,7	10	5	15
getApp()	0,0	0	3	3
equals(Object)	91,3	21	2	23
Employee(String, String, String)	100,0	22	0	22
addTask(Task)	100,0	6	0	6
addUnavailability(Unavailability)	100,0	6	0	6
checkEmployeeIsInApp()	100,0	26	0	26
getEmail()	100,0	3	0	3
getFullname()	100,0	3	0	3

Continued on next page

Tabel 23 – continued from previous page

Element	**Coverage**	**Covered Inst**	**Missed Inst**	**Total Inst**
getTasks()	100,0	3	0	3
getUnavailabilities()	100,0	3	0	3
getUsername()	100,0	3	0	3
setApp(ProjManApp)	100,0	4	0	4
setTasks(List)	100,0	4	0	4
setUsername(String)	100,0	18	0	18
Activity.java	69,5	155	68	223
Activity	69,5	155	68	223
getHelpedTasks()	0,0	0	26	26
getMainTasks()	0,0	0	26	26
doesStateNeedOverrideForProject(State)	0,0	0	12	12
isEndDateLegalForProject(Calendar)	87,5	14	2	16
isStartDateLegalForProject(Calendar)	87,5	14	2	16
Activity(String)	100,0	9	0	9
addTask(Task)	100,0	15	0	15
checkEndDateLegal(Calendar)	100,0	27	0	27
checkStartDateLegal(Calendar)	100,0	27	0	27

Continued on next page

Tabel 23 – continued from previous page

Element	**Coverage**	**Covered Inst**	**Missed Inst**	**Total Inst**
getProject()	100,0	3	0	3
getTasks()	100,0	3	0	3
setProject(Project)	100,0	8	0	8
setState(State)	100,0	35	0	35
Task.java	86,9	358	54	412
Task	86,9	358	54	412
assignHelper(Employee, double)	76,8	73	22	95
doesStateNeedOverrideForActivity(State)	0,0	0	12	12
doesStateNeedOverrideForProject(State)	0,0	0	12	12
Task(String, TaskType)	80,0	20	5	25
getHelpers()	0,0	0	3	3
assignDeveloper(Employee)	100,0	106	0	106
checkHalfHourAccuracy(double)	100,0	20	0	20
getActivity()	100,0	3	0	3
getDeveloper()	100,0	3	0	3
getEstimated_no_hours()	100,0	3	0	3
getProject()	100,0	3	0	3

Continued on next page

Tabel 23 – continued from previous page

Element	**Coverage**	**Covered Inst**	**Missed Inst**	**Total Inst**
getType()	100,0	3	0	3
getWorked_no_hours()	100,0	3	0	3
isEndDateLegalForActivity(Calendar)	100,0	16	0	16
isEndDateLegalForProject(Calendar)	100,0	16	0	16
isStartDateLegalForActivity(Calendar)	100,0	16	0	16
isStartDateLegalForProject(Calendar)	100,0	16	0	16
setActivity(Activity)	100,0	6	0	6
setEstimated_no_hours(double)	100,0	15	0	15
setProject(Project)	100,0	6	0	6
setState(State)	100,0	15	0	15
setWorked_no_hours(double)	100,0	15	0	15
Project.java	82,2	185	40	225
Project	82,2	185	40	225
Project(String, ProjectType, String)	76,7	33	10	43
checkEndDateLegal(Calendar)	81,5	22	5	27
checkProjectIsInApp()	78,3	18	5	23
checkStartDateLegal(Calendar)	81,5	22	5	27

Continued on next page

Tabel 23 – continued from previous page

Element	**Coverage**	**Covered Inst**	**Missed Inst**	**Total Inst**
isManager(Employee)	0,0	0	5	5
setManager(Employee)	66,7	10	5	15
setType(ProjectType)	61,5	8	5	13
addActivity(Activity)	100,0	11	0	11
getActivities()	100,0	3	0	3
getCustomerName()	100,0	3	0	3
getManager()	100,0	3	0	3
getType()	100,0	3	0	3
setCustomerName(String)	100,0	6	0	6
setState(State)	100,0	43	0	43
ProjManApp.java	91,4	424	40	464
ProjManApp	91,4	424	40	464
getProjectsManagedByEmployeeLoggedIn()	0,0	0	28	28
getEmployeesByFullname(String)	85,2	23	4	27
getActivityById(int)	93,8	30	2	32
getProjectById(int)	90,0	18	2	20
getTaskById(int)	95,5	42	2	44

Continued on next page

Tabel 23 – continued from previous page

Element	**Coverage**	**Covered Inst**	**Missed Inst**	**Total Inst**
getUnavailabilityById(int)	93,8	30	2	32
createActivity(Project, String)	100,0	35	0	35
createProject(String, ProjectType, String)	100,0	29	0	29
createTask(Activity, String)	100,0	36	0	36
createUnavailability(Employee, Calendar, Calendar, String)	100,0	71	0	71
employeeLogin(String)	100,0	30	0	30
employeeLogoff()	100,0	4	0	4
generateId()	100,0	8	0	8
getEmployeeByUsername(String)	100,0	21	0	21
getEmployeeLoggedIn()	100,0	3	0	3
getEmployees()	100,0	6	0	6
getProjects()	100,0	3	0	3
registerEmployee(Employee)	100,0	19	0	19
Error.java	91,3	390	37	427
Error	91,3	390	37	427
toString()	0,0	0	13	13
getCode()	0,0	0	3	3

Continued on next page

Tabel 23 – continued from previous page

Element	**Coverage**	**Covered Inst**	**Missed Inst**	**Total Inst**
getDescription()	100,0	3	0	3
State.java	71,6	73	29	102
State	70,0	49	21	70
Base.java	92,0	298	26	324
Base	92,0	298	26	324
equals(Object)	59,0	23	16	39
setEndDate(Calendar)	94,6	87	5	92
setStartDate(Calendar)	94,6	87	5	92
Base(String)	100,0	23	0	23
checkEmployeeLoggedIn()	100,0	13	0	13
getApp()	100,0	3	0	3
getDescription()	100,0	3	0	3
getEndDate()	100,0	3	0	3
getId()	100,0	3	0	3
getName()	100,0	3	0	3
getStartDate()	100,0	3	0	3
getState()	100,0	3	0	3

Continued on next page

Tabel 23 – continued from previous page

Element	**Coverage**	**Covered Inst**	**Missed Inst**	**Total Inst**
setApp(ProjManApp)	100,0	4	0	4
setDescription(String)	100,0	6	0	6
setId(int)	100,0	4	0	4
setName(String)	100,0	17	0	17
setState(State)	100,0	13	0	13
ProjectType.java	62,1	41	25	66
ProjectType	58,0	29	21	50
TaskType.java	62,1	41	25	66
TaskType	58,0	29	21	50
Unavailability.java	90,0	99	11	110
Unavailability	90,0	99	11	110
setDates(Calendar, Calendar)	92,5	62	5	67
getDescription()	0,0	0	3	3
getEmployee()	0,0	0	3	3
Unavailability(Employee, Calendar, Calendar, String)	100,0	16	0	16
getEndDate()	100,0	3	0	3
getId()	100,0	3	0	3

Continued on next page

Tabel 23 – continued from previous page

Element	**Coverage**	**Covered Inst**	**Missed Inst**	**Total Inst**
getStartDate()	100,0	3	0	3
setDescription(String)	100,0	4	0	4
setEmployee(Employee)	100,0	4	0	4
setId(int)	100,0	4	0	4
OperationNotAllowedException.java	60,0	15	10	25
OperationNotAllowedException	60,0	15	10	25
getErrorDescription()	0,0	0	10	10
OperationNotAllowedException(Error)	100,0	8	0	8
getError()	100,0	3	0	3
setError(Error)	100,0	4	0	4

Tabel 24: Code Coverage for Systematic Tests

Element	**Coverage**	**Covered Inst**	**Missed Inst**	**Total Inst**
dtu.projman.app	75,3	2.038	670	2.708
ProjManApp.java	55,2	256	208	464
ProjManApp	55,2	256	208	464
getTaskById(int)	0,0	0	44	44
getActivityById(int)	0,0	0	32	32
getUnavailabilityById(int)	0,0	0	32	32
getProjectsManagedByEmployeeLoggedIn()	0,0	0	28	28
getEmployeesByFullname(String)	0,0	0	27	27
getProjectById(int)	0,0	0	20	20
createActivity(Project, String)	71,4	25	10	35
createTask(Activity, String)	72,2	26	10	36
createProject(String, ProjectType, String)	82,8	24	5	29
createUnavailability(Employee, Calendar, Calendar, String)	100,0	71	0	71
employeeLogin(String)	100,0	30	0	30
employeeLogoff()	100,0	4	0	4
generateId()	100,0	8	0	8
getEmployeeByUsername(String)	100,0	21	0	21

Continued on next page

Tabel 24 – continued from previous page

Element	Coverage	Covered Inst	Missed Inst	Total Inst
getEmployeeLoggedIn()	100,0	3	0	3
getEmployees()	100,0	6	0	6
getProjects()	100,0	3	0	3
registerEmployee(Employee)	100,0	19	0	19
Employee.java	56,1	148	116	264
Employee	56,1	148	116	264
getTotalEstimatedHoursOfWorkForWeek(Calendar)	0,0	0	31	31
getTotalWorkedHoursOfWorkForWeek(Calendar)	0,0	0	31	31
checkAvailability(Calendar, Calendar)	0,0	0	27	27
getTotalRemainingHoursOfWorkForWeek(Calendar)	0,0	0	18	18
setTasks(List<Task>)	0,0	0	4	4
getApp()	0,0	0	3	3
equals(Object)	91,3	21	2	23
Employee(String, String, String)	100,0	22	0	22
addTask(Task)	100,0	6	0	6
addUnavailability(Unavailability)	100,0	6	0	6
checkEmployeeIsInApp()	100,0	26	0	26

Continued on next page

Tabel 24 – continued from previous page

Element	**Coverage**	**Covered Inst**	**Missed Inst**	**Total Inst**
getEmail()	100,0	3	0	3
getFullname()	100,0	3	0	3
getTasks()	100,0	3	0	3
getUnavailabilities()	100,0	3	0	3
getUsername()	100,0	3	0	3
setApp(ProjManApp)	100,0	4	0	4
setEmail(String)	100,0	15	0	15
setFullname(String)	100,0	15	0	15
setUsername(String)	100,0	18	0	18
Activity.java	60,5	135	88	223
Activity	60,5	135	88	223
getHelpedTasks()	0,0	0	26	26
getMainTasks()	0,0	0	26	26
doesStateNeedOverrideForProject(State)	0,0	0	12	12
checkEndDateLegal(Calendar)	63,0	17	10	27
checkStartDateLegal(Calendar)	63,0	17	10	27
isEndDateLegalForProject(Calendar)	87,5	14	2	16

Continued on next page

Tabel 24 – continued from previous page

Element	Coverage	Covered Inst	Missed Inst	Total Inst
isStartDateLegalForProject(Calendar)	87,5	14	2	16
Activity(String)	100,0	9	0	9
addTask(Task)	100,0	15	0	15
getProject()	100,0	3	0	3
getTasks()	100,0	3	0	3
setProject(Project)	100,0	8	0	8
setState(State)	100,0	35	0	35
Task.java	84,0	346	66	412
Task	84,0	346	66	412
assignHelper(Employee, double)	76,8	73	22	95
doesStateNeedOverrideForActivity(State)	0,0	0	12	12
doesStateNeedOverrideForProject(State)	0,0	0	12	12
Task(String, TaskType)	80,0	20	5	25
assignDeveloper(Employee)	95,3	101	5	106
getHelpers()	0,0	0	3	3
getType()	0,0	0	3	3
isEndDateLegalForProject(Calendar)	87,5	14	2	16

Continued on next page

Tabel 24 – continued from previous page

Element	Coverage	Covered Inst	Missed Inst	Total Inst
isStartDateLegalForProject(Calendar)	87,5	14	2	16
checkHalfHourAccuracy(double)	100,0	20	0	20
getActivity()	100,0	3	0	3
getDeveloper()	100,0	3	0	3
getEstimated_no_hours()	100,0	3	0	3
getProject()	100,0	3	0	3
getWorked_no_hours()	100,0	3	0	3
isEndDateLegalForActivity(Calendar)	100,0	16	0	16
isStartDateLegalForActivity(Calendar)	100,0	16	0	16
setActivity(Activity)	100,0	6	0	6
setEstimated_no_hours(double)	100,0	15	0	15
setProject(Project)	100,0	6	0	6
setState(State)	100,0	15	0	15
setWorked_no_hours(double)	100,0	15	0	15
Error.java	91,3	390	37	427
Error	91,3	390	37	427
toString()	0,0	0	13	13

Continued on next page

Tabel 24 – continued from previous page

Element	**Coverage**	**Covered Inst**	**Missed Inst**	**Total Inst**
getCode()	0,0	0	3	3
getDescription()	100,0	3	0	3
State.java	71,6	73	29	102
State	70,0	49	21	70
Base.java	92,0	298	26	324
Base	92,0	298	26	324
equals(Object)	59,0	23	16	39
checkEmployeeLoggedIn()	61,5	8	5	13
setState(State)	61,5	8	5	13
Base(String)	100,0	23	0	23
getApp()	100,0	3	0	3
getDescription()	100,0	3	0	3
getEndDate()	100,0	3	0	3
getId()	100,0	3	0	3
getName()	100,0	3	0	3
getStartDate()	100,0	3	0	3
getState()	100,0	3	0	3

Continued on next page

Tabel 24 – continued from previous page

Element	Coverage	Covered Inst	Missed Inst	Total Inst
setApp(ProjManApp)	100,0	4	0	4
setDescription(String)	100,0	6	0	6
setEndDate(Calendar)	100,0	92	0	92
setId(int)	100,0	4	0	4
setName(String)	100,0	17	0	17
setStartDate(Calendar)	100,0	92	0	92
Project.java	88,4	199	26	225
Project	88,4	199	26	225
setCustomerName(String)	0,0	0	6	6
checkEndDateLegal(Calendar)	81,5	22	5	27
checkProjectIsInApp()	78,3	18	5	23
checkStartDateLegal(Calendar)	81,5	22	5	27
isManager(Employee)	0,0	0	5	5
Project(String, ProjectType, String)	100,0	43	0	43
addActivity(Activity)	100,0	11	0	11
getActivities()	100,0	3	0	3
getCustomerName()	100,0	3	0	3

Continued on next page

Tabel 24 – continued from previous page

Element	**Coverage**	**Covered Inst**	**Missed Inst**	**Total Inst**
getManager()	100,0	3	0	3
getType()	100,0	3	0	3
setManager(Employee)	100,0	15	0	15
setState(State)	100,0	43	0	43
setType(ProjectType)	100,0	13	0	13
ProjectType.java	62,1	41	25	66
ProjectType	58,0	29	21	50
TaskType.java	62,1	41	25	66
TaskType	58,0	29	21	50
Unavailability.java	87,3	96	14	110
Unavailability	87,3	96	14	110
setDates(Calendar, Calendar)	92,5	62	5	67
getDescription()	0,0	0	3	3
getEmployee()	0,0	0	3	3
getId()	0,0	0	3	3
Unavailability(Employee, Calendar, Calendar, String)	100,0	16	0	16
getEndDate()	100,0	3	0	3

Continued on next page

Tabel 24 – continued from previous page

Element	**Coverage**	**Covered Inst**	**Missed Inst**	**Total Inst**
getStartDate()	100,0	3	0	3
setDescription(String)	100,0	4	0	4
setEmployee(Employee)	100,0	4	0	4
setId(int)	100,0	4	0	4
OperationNotAllowedException.java	60,0	15	10	25
OperationNotAllowedException	60,0	15	10	25
getErrorDescription()	0,0	0	10	10
OperationNotAllowedException(Error)	100,0	8	0	8
getError()	100,0	3	0	3
setError(Error)	100,0	4	0	4