Rettelser

The Faculties of Engineering, Science and Medicine
Software
Selma LagerlÃ¶fs Vej 300
Telephone +45 9940 9940
Fax +45 9940 9798
http://www.cs.aau.dk/

Title:
Hopla Helpdesk

Theme:
Programming

Project period:
SW3, fall semester 2010

Project group:
S305A

Participants:

Alex Bondo Andersen

Kim Jakobsen

Magnus Stubman Reichenauer

Kristian Kolding Foged-Ladefoged

Lasse Rørbæk Nielsen

Rasmus Veiergang Prentow

Synopsis:

Der kleine symopsis mit dem hairy stomach

Advisor:
Nadeem

Page count: 96

Appendices count: 0

Finished: 17/12–2010

Preface

This report is written to document the 3rd semester project, compiled by group s305a - Software Engineering students from Department of Computer Science at Aalborg University.

The DVD contains the complete source code of Hopla Helpdesk, the database[1], the tests and report as PDF.

We have chosen to utilize the knowledge from the PE courses System Analysis and Design (SAD) and Object Oriented Programming (OOP), by designing our system using the Object Oriented Analysis & Designand programming Hopla Helpdeskin an object oriented way.

We would like to give thanks to supervisor Nadeem Iftikhar, Department of Computer Science at Aalborg University for constructive and technical feedback.

[1]FiXme Dødelige: skriv hvad det er vi sender med af db'en

Contents

1 Introduction

Organizing and solving problems can be a difficult task. The fact that each service employee need to cooperate with each other to solve the problems, induces a complex organization problem. The goal is to solve all problem as fast as possible. However the problem should be solved in a prioritized order, with the most important problems first. Furthermore should two service employees not try to solve the same problem independently, because this would be redundant work. Hopla Helpdesk is about solving this particular problem. Hopla Helpdesk purpose is – in short – to distribute specific problems to the group of people who are best at solving them, while solving other subproblems such as not overburdening one service employee with all the problems etc.
Hopla Helpdesk does not only serve as a one-way information stream about existing problem, but also delivers a system where the people who solve the problems can report back to the client with solutions or questions regarding the problem. All this while other clients who might have a similar or identical problem, can subscribe and thereby follow the problem from the beginning to the end.

The result of using Hopla Helpdesk this way, is a database full of problems, with – hopefully – attached solutions. Hopla Helpdesk uses these informations in a preventive way, which displays possible solutions to a client, and in some cases, these solved trivial problems with attached solutions might help the client instantly, keeping the workload of the people solving problems down, as well as the clients spent time to a minimum.
Productivity goes up, wasted time goes down.

Part I

Analysis

2 Task

This chapter presents the subject, our goal, and how we planed to model the system. It will present the purpose of our project, our system definition which defines how our project should end up, and a context where the system is seen from a perspective of the end user.

2.1 Purpose

When maintaining a work environment different problems are bound to arise. Computers will break down, printers will need installation and light bulbs will need replacement. Larger organizations will most likely have people hired to do this job exclusively. We develop this system with the purpose of easing the process of solving problems, as well as distributing knowledge as to how to solve trivial problems that will reoccur, and keeping track of already known problems. Thereby relieving the maintenance staff of these trivial tasks, effectively increasing the productivity of the employees.

2.2 System Definition

To better be able to define our system we will use the FACTOR method[10, p. 39]. This model contains different parts of the requirements, and it will help us derive the system definition.

Functionality
Our program will contain features aimed at easing the problem-solving process for service employees. Therefor the system will have the following features:

- Problem distribution
  The system should distribute problems to the relevant service department and balance the workload of service employees so all have a equal workload.

- Problem categorization
  Categorize problem and thereby making it easy to find relevant problems.

- Ranking of problems
  Unsolved Problems are displayed on a ranked list, depending on their importance.
- Keep track
  The system should be able to keep track of all the information regarding a problem, this being
  - When it approximately will be solved
  - Who is assigned to solve the problem
  - What the deadline is, if the staff approved it, and if it is exceeded
  - What tags and categories are related to the problem
  - Comments related to the problem
- Knowledge saved
  The system should be able to recognize common problems and recommend similar problem to the user, and by that saving the service employee the bother of solving common problems.
- Statistics
  Allow the supervisor to monitor the work process of the employees.

Applicationdomain
This system will be applicable to office environments which deals with solving of problems.

Conditions
The problem submitting users are not required to have any expert knowledge to use the helpdesk. The service personal will have to learn to use the staff interface to solve problems. The administrators will have to learn to use the functionality in the administrator interface.

Technology
In the development of our program we are going to use the programming language C# together with the Model-View-Controller framework 2 that exists within the ASP.NET framework. We will set up a development server and use AnhkSVN and Microsoft SQL server along with visual studio 2010.

The end result will be a web interface running on a webserver, with underlying SQL database.

Objects
Problems, solution, clients, staffmembers, admins, departments, categories and tags.

Responsibility
Our system is responsible for keeping track of all technically related problems within an organization. It is also responsible for distributing tasks amongst employees and supplying statistics on their progress to their supervisors. Finally

it is also responsible for enabling users and technical employees to communicate about a problem and the following solution.

Using these FACTOR criteria, we arrive at the following system definition:

- The system should be web-based so that we can create an application capable of running without prior installation.

- The system should contain prioritized tags, enabling us to determine the importance of problems

- The system should keep track of the problems Estemated time of compleation, comments and what employee is responsible for the problem. This will enable us to create detailed statistic about problem solving efficiency and individual employees.

- The system should be able to save and suggest prior problems and their solutions to users.

- The system should take into account, the workload of the employees to be able to estimate how long a problem will take to finish.

- The system should only use dynamic data, in order to run in different environments.

- The system should contain a structure that can handle problems, categories, tags, employees, supervisors, ordinary users and department.

2.3 Context

Our system will target a work environment or an educational environment. The analysis and design process will be based upon a university, but the system should be easily implemented at any other work environment.

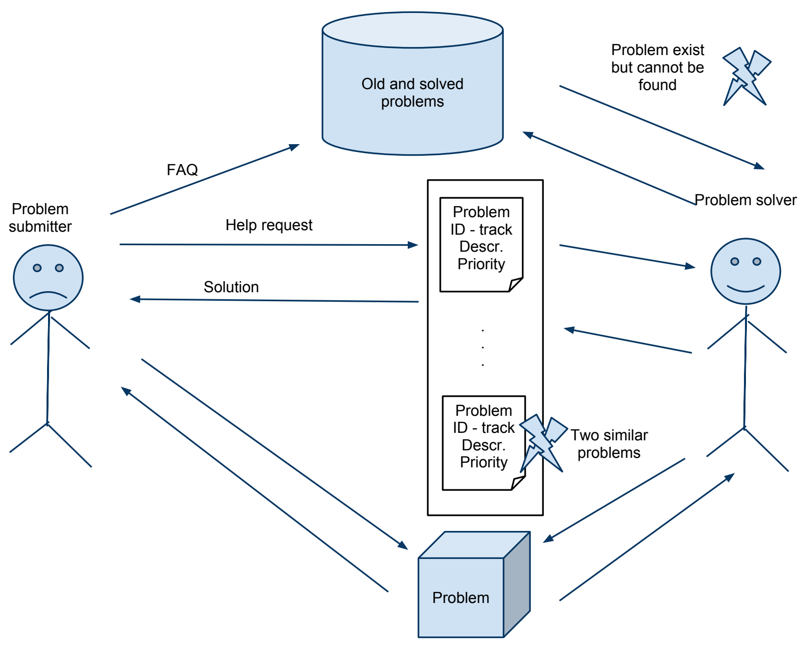


Figure 2.1: *Rich picture of our system*

To fully understand the context we draw a rich picture which can be seen on figure 2.1. The central aspects on the rich picture is that both the problem submitter and the problem solver can act on the problem, and will communicate through the system. From the rich picture we see a few conflicts in the environment:

- The first is that problem exist but cannot be found.
- The second is that there could exist two similar problem.
- The central objects in the system are the problems and solutions.

2.3.1 Problem Domain

A central phenomena in our system will be to add a problem to the system. This will occur when a client finds a problem in the organization and wants to submit it, in order to get it solved.

Another important phenomena in our system is when a problem is solved. This is initiated by a staffmember and will result in a notification to the clients who are subscribing to the problem.

Chapter 3 gives a detailed analysis of the problem domain and will elaborate on the phenomena.

2.3.2 Application Domain

The people who will be acting on the system is the problem solver and submitter. Most likely the submitter will be a student in a university environment and the solver will be the technical staff. But the submitter could be a staff as well. Beside the two main actors there is an admin who can administrate the staff, clients, and the system itself.

A detailed analysis of the application domain is presented in chapter 4, where the actors and there use cases are further described.

The purpose of this project is shown in this chapter along with the system definition and the context of which the system should be viewed.

3 Problem Domain

In order to determine the nature of our solution we have to analyze the context in which it is going to be used. The classes and events described in this chapter are reflections of how the system is in the physical world. This chapter also includes a description of the behavior of each class, in order to understand the different classes better.

3.1 Classes and Events

In the process of finding classes and events we made up several classes and events some of which are not used in the actual program. To come up with the classes and events we talk about different possible scenarios and based on that we developed our class diagram. This section describes all the classes and events in the final iteration of the process and all the redundant classes and events are omitted. On figure 3.2 the relations between classes and events are illustrated.

3.1.1 Classes

The following is a list of the classes from the class diagram 3.1.

Problem Problem is the basic building brick in our system. It is used to hold information about the specific problems which it represents.

Deadline The deadline can be approved or not approved and is a part of the problem.

Comment A comment belongs to a problem.

Solution Contains state and information about a given solution

Person This contains attributes about the users in the system, name, address, phone number ect. We will look into the details later in the report.

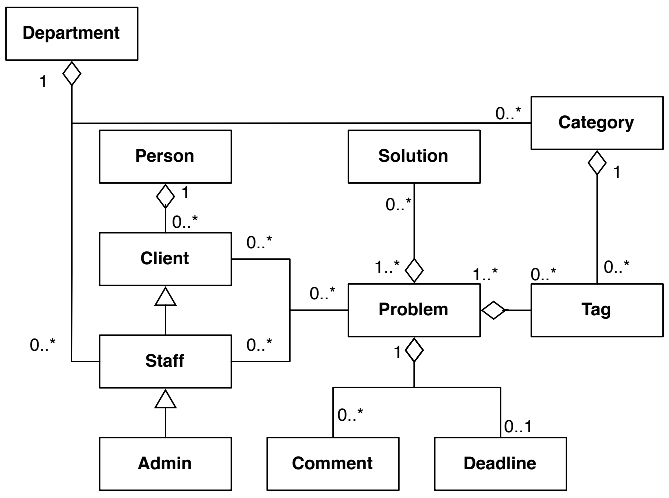


Figure 3.1: Class diagram

Client A client can be subscribed to a problem and be associated with the comments that the actor client posts.

Staff The staff class inherits from the client class. staff can be assigned to problems.

Admin The admin is not associated with any other classes and is only used to administrate the users.

Department Contains information about staff and categories.

Category Categories contain Tags

Tag Tags is used to determine the problems category and thereby department.

3.2 Structure

The class diagram is based on the classes and events, which was described earlier in chapter 3.1. The class **person** aggregates a role. The role class itself is removed since only **client** inherited from it. **staff** inherits from **client** and **admin** inherits from **client**. Alternatively the role pattern [10, p. 80] could have been used. This prescribes that **admin**, **client**, and **staff** all would have been a subclass of *role*. We considered that it made more sense that they inherited from each other, since all the privileges **staff** has the **admin** has as well. Same applies for **staff** and **client**.

For the class person different relations appear depending on his role.

- **clients** can subscribe to **problems**.

	Classes				
Events	Problem	Solution	Staff	Department	Person
Problem added	✓			✓	
Problem solved	✓	✓		✓	
Problem updated	✓		✓	✓	
Problem assigned	✓		✓	✓	
Problem unassigned	✓		✓	✓	
Problem deleted	✓			✓	
User replied	✓		✓	✓	
Staff replied	✓				
Department created				✓	
Department closed				✓	
Role assigned				✓	✓
Role unassigned				✓	✓
Person created					✓
Solution found	✓	✓	✓		✓
Solution assigned	✓	✓	✓		✓
Problem categorized	✓			✓	

Figure 3.2: *Problem-domain analysis event table*

- **staff** can be assigned to **problems** and **staff** belongs to a department.

- **admin** does not have any relations. But is necessary for administration of users.

A problem consist of none or many *comments*, none or many *solutions* and, one or many *tags*. A **tag** belongs to a category and a **category** belongs to a **department**. The class diagram is illustrated at figure 3.1

3.3 Behavioral Pattern

In this section of the report, we will describe the behavior of the two major classes **Problem** and **Person**. These two are the only classes which can have more states.

3.3.1 Problem

Figure 3.3 shows the behavior of a problem. Note that you can solve the problem by attaching one or more solutions. A problem can have an unlimited number of solutions. You can at all times delete the problem, thus the arrow points from the edge of the box.

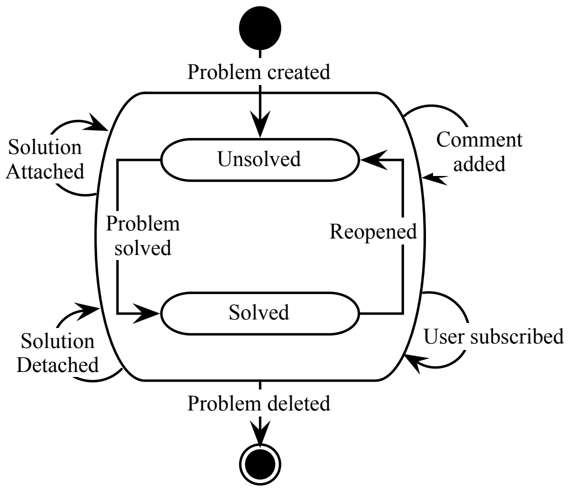


Figure 3.3: *The statechart of the problem class*

3.3.2 Person

As shown in figure 3.4, a person is assigned a role in the system as soon as he is created. He can only have one role at a time. The roles inherit from each other. Meaning that a admin can do the same as the client, but clients can do the same as admin.

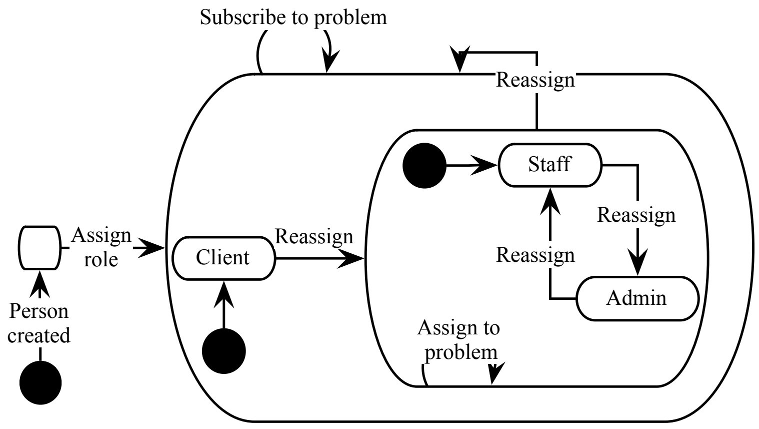


Figure 3.4: *The class person's statechart*

The classes and events which will be used to model the reality are described in this chapter. These descriptions includes a definition of each class and event, the structure in which the classes resides, and the behavior of each class.

4 Application Domain

In this chapter the application domain will be analyzed and our choices regarding the application domain explained. The purpose of this chapter is to determine the system's usage requirements.

4.1 Usage

We have identified three actors and picked out the six most relevant use cases. The three actors are: client, staff, and admin. The use cases we picked are: Submit problem, My problems, Worklist, Solve problem, Administrate, and Statistics.

	Actor		
Use case	Client	Staff	Admin
Submit problem	✓	✓	✓
My problems	✓	✓	✓
Worklist		✓	✓
Solve problem		✓	✓
Administrate			✓
Statistics			✓

Figure 4.1: *Actor & use case table*

Figure 4.1 shows the relationship between use cases and the actors of our system. All three roles are able to "Submit problems" and see "My problems". The staff and admin have a "Worklist" and can "Solve problem". The admin can "Administrate" the system and access the "Statistics"

The use cases in figure 4.1 are described in subsection 4.1.2.

4.1.1 Actors

The system has two primary actors: client and staff. Client is the lowest privileges human actor, his primary use case is to submit new problems. The client is also able to track his problems and communicate with the staff.

The staff members are more privileged and can solve problems. All staff members can act as clients if they themselves has a problem which should be addressed to another department. E.g. the lightbulp in the IT-administrators office is broken and the maintenance officer should fix it.

These two actors are described in details in figure 4.2 and 4.3. Beside staff and client the system has another actor called admin. The admin is a more privileged staff or a manager of the staffmembers.

The admin can manage tags, categories, departments and view statistics of each staffmember, the actor admin is decribed in details in figure 4.4.

Client
Goal: A person who has a problem, and his goal is to get his problem(s) solved. **Characteristics:** The clients are employees or students with different knowledge and experience with similar systems. **Examples:** Client A prefers face-to-face communication with the working staff whenever he has got a problem. Client B prefers web/mail communication as a substitution of face-to-face communication so he does not have to leave his working space in order to get help.

Figure 4.2: *Description of the actor client.*

Staff
Goal: The staff solves the clients problems and use the system as a taskmanager. **Characteristics:** The staff are employees and has various levels of technical knowledge. **Examples:** Staff A prefers to speak to his manager and the client face-to-face. Staff B enjoys getting his daily tasks from a computer system,

Figure 4.3: *Description of the actor staff.*

4.1.2 Use Case

The use cases from figure 4.1 are described below.

Admin
Goal: The admin is system and staff manager. **Characteristics:** Responsible for the Hopla Helpdesk or staff manager. **Examples:** Admin A is a software engineer and are responserable for maintaining the system by management departments, category, and tags. Admin B is the boss of the department and evaluate staffmembers based on the statistics generated by the helpdesk.

Figure 4.4: *Description of the actor admin.*

Submit problem The use case submit problem is only used by the actor client. Except for cases when staff or admin acts as client. A use case diagram is shown in figure 4.5.

- **Use Case:** Submit problem is initialized when a client has a problem and wishes to submit that problem to the system in order to get help from the staff. First he has to select a category and choose one or more tags, he can select tags from more then one category. When the client is done selecting tags the system compares the selected tags with other problems. If similar problems is found the client is presented with these. If one of these matches his particular problem, he can subscribe to the problem if it is unsolved or read the solution(s) if the problem is closed, thus hoping that this will lead to solving the clients problem. If no similar problem was found the client creates a problem with a title, description and the previously selected tags. Hereafter the problem gets assigned to a staff.
- **Objects: Problem, tag, comment, category, deadline, client, and staff**.
- **Functions:** Get problem tags, Search for problems, Create problem, Subscribe / unsubscribe to problem, and Get Est. Time Consumption of Problem.

My problems The use case my problems is used by clients, staffs, and admins. It show a list all problems submitted by the user. From there details can be viewed for each problem.

Statistics The use case statistics is accessible by the admin. It shows statistics about how much time each staff use to solve problems.

Solve problem The use case solve problem is the staffs primary usage of the system. When the staff get his worklist he can select a problem and solve the problem. A statechart diagram is shown in figure 4.6.

- **Use Case:** The use case is initialized when the staff wants to check his worklist. The staff is then presented with a list of unsolved problems

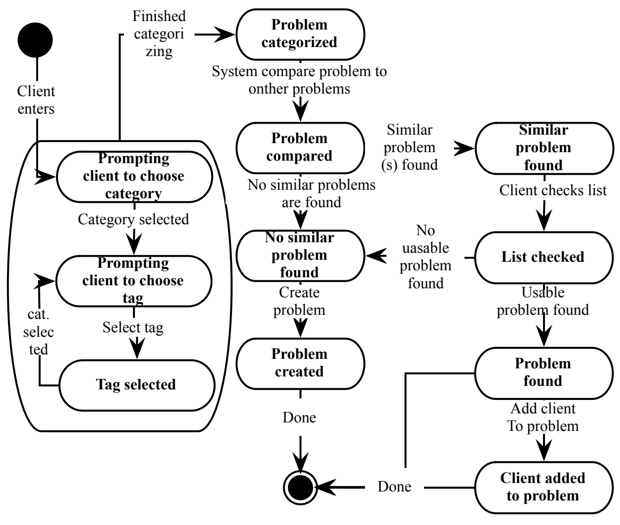


Figure 4.5: *A state chart diagram of the use case submit problem.*

assigned to him. The staff can then click on one of the problems to read the problem, see status of it, add comments to it, search the database for similar problem, reassign it, or write a new solution.

- **Objects:Problem, tag, comment, category, deadline, client, staff, department, and solution**
- **Functions:**Get problem tags, Search for problems, Create problem, Subscribe / unsubscribe to problem, Get est. time consumption of problem, Manage tag times, Get expected time of completion of problem, Get tag average time consumption, Get sorted worklist and Approve deadline.

Administrate The use case administrate is used by the admin to administrate persons, tags, categories, departments and view statistics about the specific staffmembers. A statechart diagram is depicted on figure 4.7.

- **Use Case:** The use case starts as the admin enters the site. The admin can manage people by change role, department, email and delete the person from the system. The admin is also able to create new departments and add categories thereto. To each category new tags can be added. Tags have a priority which can be changed at anytime. Tags cannot be removed when they have been used, but they can be hidden so they cannot be used to categories new problems.
- **Objects: Problem, tag, comment, category, deadline, client, staff, department, and solution**
- **Functions:** Get problem tags, Search for problems, Create problem, Subscribe / unsubscribe to problem, Get est. time consumption of problem,

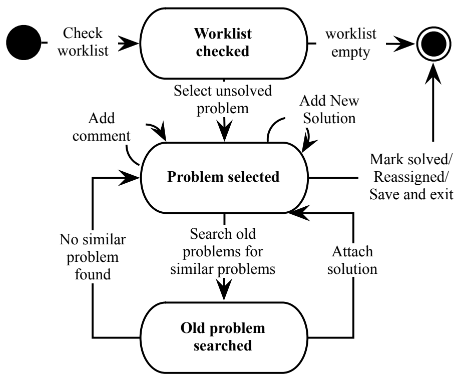


Figure 4.6: *A state chart diagram of the use case solve problem.*

Manage tag times, Get expected time of completion of problem, Get tag average time consumption, Get sorted worklist, Approve deadline, Create department, Create category, Create tag, Hide / show category, Hide / show tag, Delete person, Reset person password, Balance workload, Get statistics, and Distribute problem.

4.2 Function

The purpose of this section is to "determine the system's information processing capabilities" [10, p. 137]. Ultimately resulting in "a complete list of functions with specification of complex functions." [10, p. 137] All functions with a medium or complex complexity are explained below.

Search for problems This function searches the model for problems with specific tags, then presents them in an ordered list assorted after problems with most similar tags. It is both *read* and *calculate* because it reads in the model and compute which problems are most similar. This function is *complex*.

Get tag average time Consumption of a tag This function calculates the average time consumption of a tag. It has to divide the number of problem solved and the total amount of time all problems with this tag required to be solved.

Manage tag time This function takes the time a staff has used to solve a problem and adds to the related tags time consumption and increments the solved problems property.

Get estimated time consumption of problem This function calculates the amount of time a specific problems acquires to be solved. This is based

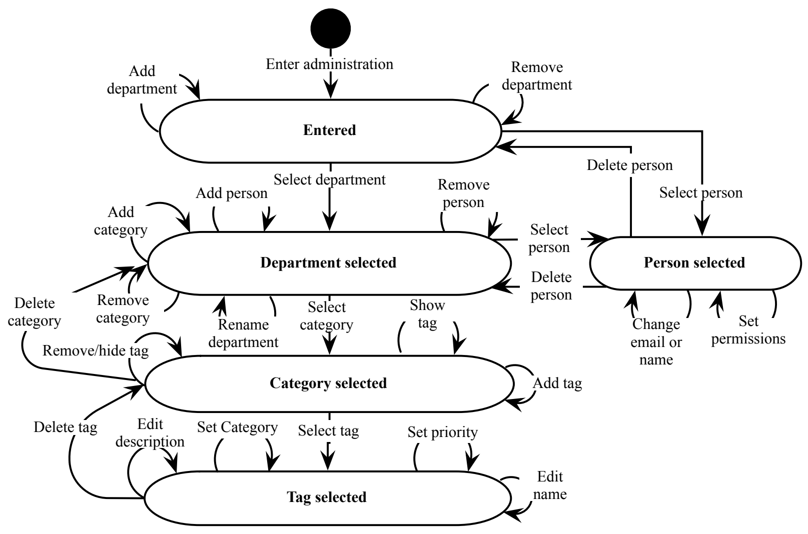


Figure 4.7: *A statechart diagram for the use case administrate.*

Name	Complexity	Operations
Get problem tags	Simple	Read
Search for problems	Complex	Read/Calculate
Create problem	simple	Update
Reassign staff to problem	Simple	Read/Update/Signal
Subscribe / unsubscribe to problem	Simple	Update
Attach / detach solution to problem	Simple	Read/Update
Manage tag times	Medium	Update/Calculate
Get est. time consumption of problem	Medium	Read/Calculate
Get expected time of completion of problem	Medium	Read/Calculate
Get tag average time consumption	Simple	Read/Calculate
Get sorted worklist	Medium	Read/Calculate
Approve deadline	Simple	Update
Create department	Simple	Update
Create category	Simple	Update
Create tag	Simple	Update
Hide / show category	Simple	Update
Hide / show tag	Simple	Update
Delete person	Simple	Update
Reset person password	Medium	Update/Signal
Balance workload	Complex	Calculate
Get statistics	Medium	Read/Calculate
Distribute problem	Medium	Read/Calculate/Update

Figure 4.8: *Function list*

upon the related tags. The tags has a property describing the average time consumption.

Get expected time of completion of problem This function takes all the problems of the assigned staff member which are expected to be solved prior this problem and adding up the amount each is estimated time each problem will consume.

Get sorted worklist This function returns the work list of a staff member in sorted order after priority.

Reset person password If a person forgets his password, then he can get a new password send to his mail. The password is randomly generated. This function updates the model to match the computed password. This function is *medium*.

Balance workload This function compares the workload of staff and distributes their problems equally among other staffmembers in the same department. This function calculate the workload of staffs and then moves problems in the most optimal way. This function is Complex.

Get statistics The statistics function calculate how long each staff use to solve a problem in average. The function can also find the total average for departments. This function is medium.

4.3 User Interfaces

Our system's user interface is divided into three sub-interfaces – one for each human actor. These interfaces are described in the following subsections.

4.3.1 Client Interface

The clientinterface is illustrated in figure **??**. After login, the client is presented with the *main* window.

Main

The clients main window allows the clientto choose what to do in the helpdesk system. The "Main" window have three buttons:

- The "Commit Problem" button which sends the user to the "Categorize new problem" window.
- The "My problems" button which sends the user to "Search" window.
- The "Search for problems" button which sends the user to "Search" window.

Search

The search windows is used to search for problems containing specific tags or problems posted by the client self. The “Search” window contains the following elements:

- The “Categories and tags” frame, see paragraph below.
- A settings panel.
- A search button that triggers the search, see paragraph below.
- A list containing the problems matching the settings and tags.

A problem can be double clicked to view details about the problem in the “Client problem view” windows.

Settings The “Settings” frame contains configurations options for the search. The frame contains the following elements:

- My problems
- Unsolved problems
- Solved problems
- Number of search results

When a search is preformed these options are checked and they will effect the result of the search.

Categories and tags This frame is used in a few other windows as well. The frame contains:

- A list of categories and the tags attached to it.
- A checkbox for each tag.

Client problem view

This window contains information about the selected problem. The window contains the following:

- A info field which contains relevant information about the problem e.g. the description of the problem.
- A subscription check-box, clients subscribed to the problem will receive notifications when then problem changes.
- A text field with all the previous entered comments if any.
- A field to write new comments in.
- The button “Submit comment” which submits the comment.
- A field with non or more solutions written by the staff members.

Categorize New Problem

To add a new problem the client need to select the tags which best describes the problem. The window contains the following elements:

- The frame “Categories and tags”.
- The button “Search for this kind of problems” which sends the client to the window “Search for similar problems”

Search for similar problems

A search is preformed with the selected tags from the previous window, a list with similar problems are displayed. This window includes the following:

- The window “Search” with all it elements.
- A button called “No problem suffice” which loads the window “Create new problem”.

The found problems can be double clicked and the window “Client problem view”.

Create new problem

The “Create new problem” window’s purpose is to describe, categorize, suggest deadline and submit the problem. The window contains the following elements:

- A text field where information about the problem can be entered.
- The frame “Categories and tags” enables the clientto change and add tags so the problem is best describe.
- A calender where a deadline can be suggested to the staffmember.
- “Create” submits the problem and send the user to the “Client problem view”.

4.3.2 Staff Interface

The staff interface is illustrated in figure 4.9. After login, the staff is presented with the *main* window.

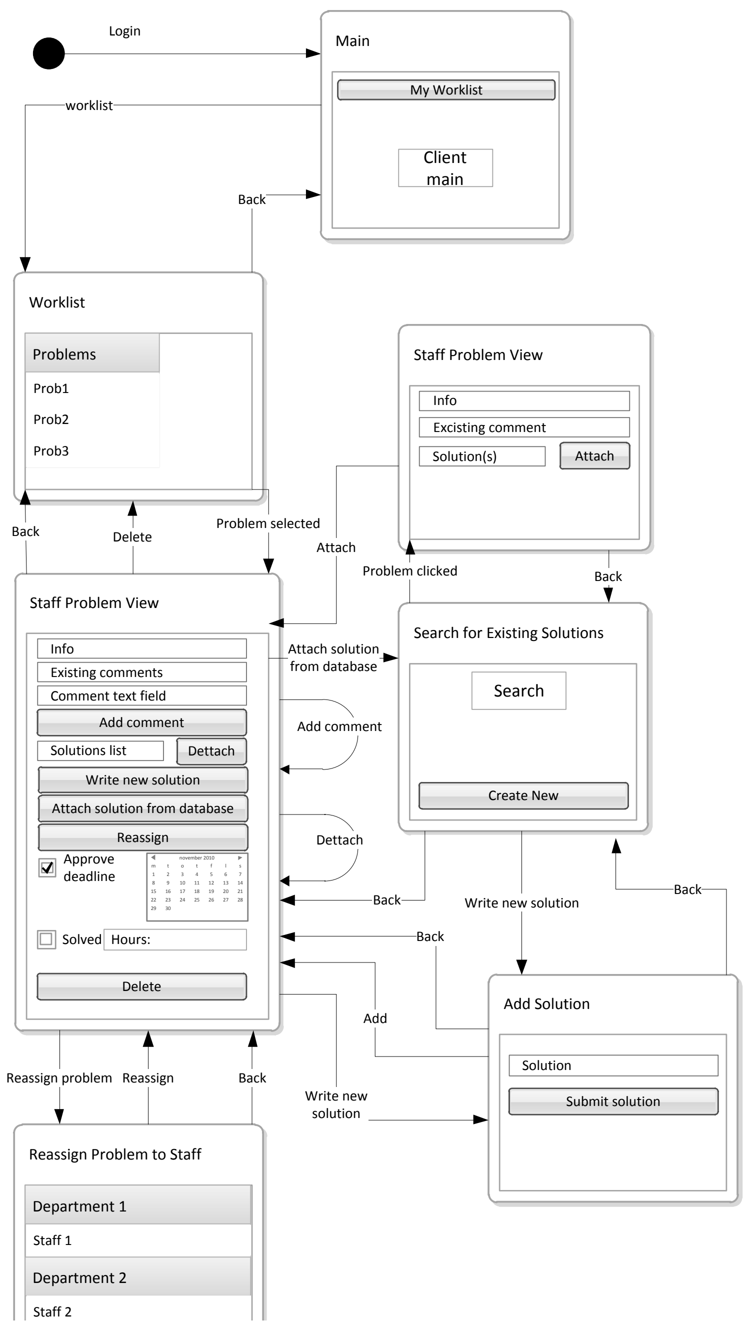


Figure 4.9: *Staff Interface*

Main

The staff main window give the staffmember access to all the functionality from the staffactor and from the clientactor. The staffhave one button:

- “My Worklist” which directs the staffmember to the “Worklist” window.

plus the menu button from the clients main menu.

Worklist

The “Worklist” is a list of all the problem assigned to a specific staffmember. The window has the following elements:

- A list with all the problems assigned to the staffmember who is signed in.

The list show the following properties: Name, Deadline, Priority, and ETC. When a problem is clicked the window “Staff problem view” opens.

Staff problem view

This window shows all the information related to a specific problem. The “Staff problem view” features the following:

- The window contains a text field which contains information about the problem
- A text field with all the existing comments related to the problem.
- A text field where new comments can be entered.
- The button “Create comment” to send the entered comment.
- A text field containing none or more solutions.
- The button “Attach solution from database” which send the staff to “Search for existing solutions”.
- The “Reassign” button which opens the “Reassign problem to staff” window.
- A checkbox to mark the problem as solved.
- A calender to set deadlines.
- The checkbox “Approve deadline” approves a suggested deadline if one is suggested.
- The button “Write new solution” which sends the staff to “Add solution”

Search for existing solutions

This window enables the staff to search for existing solutions among existing problems. The “Search for existing solutions” window contains the following elements:

- The “Search” window from the client interface 4.3.1 is reused, and it enables the staff to search for problems.
- The button “Create New” which sends the staffto the “Add solution” window.

If a problem is double click the staff is send to “Staff problem view”.

Staff Problem View

The “Staff Problem View” show properties of the selected problem. This window contains the following:

- A text field with relevant information about the problem.
- A text field with all the existing comments related to the problem.
- A text field containing none or more solutions.
- The button “Attach” inserts the selected solution into the solutions list from “Staff Problem View”, and the staff is send to the “Staff Problem View” window.

Add Solution

The “Add Solution” window allows the staff to enter a new solution. The window contains the following elements:

- A text field where the new solution can be entered.
- The “Add” button which sends the entered solution to the Solution list in the “Staff Problem View” window, the staffis send to the “staff Problem View” window.

4.3.3 Admin Interface

The admin interface is illustrated in figure **??**. After login, the admin is presented with the *main* window. All the window have a back function which enables the admin to return to the previous window.

Main

The “Main” adminwindow gives access to administration options and the functionality from the staffand clientmain windows. The window contains:

- The button “Manage Deparments” directs the admin to the “Department Administration” window.
- The button “Manage People” directs the admin to the “Person Administration” window.

- The button “statistics” directs the admin to the “statistics” window.

plus the buttons from the staff and client “main” window.

Person administration

When the “person Administrate” window is opened, the adminis able to browse through all staffmembers. The window contain the following element:

- A list with all the persons in the system.

Department administration

This window shows a list of staff and categories for the selected department. The “Department administration” contains the following:

- A dropdown menu where department can be selected.
- A list of staff members who are a part of the selected department.
- A list of categories which is related to the department.

When the staff is clicked the admin is send to the “person view”. When a category is clicked the admin is send to the “Category view”.

Person view

The “person view” shows information about a selected person. The window contains the following elements:

- A info field where relevant information is displayed about the person
- A dropdown where roles can be set.
- A dropdown where departments can be set.
- The button “delete” which deletes the person and sends the user back to the previous window.

Category view

The “Category View” window shows information about the selected category and allows it to be modified. The window contains the following:

- An info field where relevant information is displayed.
- A list with all the tags belonging to the category.
- The “Create new tag” which send the admin to “Tag View” window.
- The “hide” button which hide category and its tags.

Each tag have an “edit” button which allows a tag to be modified, the button sends the adminto the “Tag view” window.

Tag view

This window is used to create new tags and edit already existing tags. It contains the following:

- A info field which can be edited.
- An “Hide” button which hide the tag.
- An “save” button which save the changes made.

When the hidden button is pressed then the admin is directed back to the “category view” window.

This chapter shows the Application Domain Analysis for our system. First the usage of our system is described, then the functions which are used to manipulate data in Problem Domain or signal actors in the Application Domain is presented and defined, lastly the User Interfaces are described.

Part II

Design

5 Task

This chapter presents the purpose of our system in a short and precise text and gives the quality goals to which the project can be evaluate against during design, implementation, and after deployment.

5.1 Purpose

Hopla Helpdesk should ease the problem solving process in the applied environment. It should balance the problems between the staff based on their workload and the problems priority.[1] The system should support a method for the staff and client to communicate, enable the users of the system to search through existing problems and monitor their own problems, be accessible everywhere through the internet and organize the already solved problems in a way which makes them easy to browse through.

5.2 Quality Goals

The design of Hopla Helpdesk is specified from the following quality goals: Usable, secure, efficient, consistence, reliable, maintainable, testable, comprehensible, reusable, portable, and interoperable. The definition of these quality goals – or simply criteria – is shown in figure 5.1. [10, p. 178]

5.2.1 Quality Goals

Since it is not possible to prioritize each criterion equally, we have chosen to use a four step priority scale: Very important, important, less important, and irrelevant. Each criterion defined in figure 5.1 is prioritized in figure 5.2. Further the figure 5.2 shows a column labeled “Easily Fulfilled” which specifies whether a given criterion will be fulfilled without much effort.

How we have prioritized the criteria is based on our system definition, which is found in chapter 2.2. The reasoning for the priority of each criteria in figure 5.2 is shown bellow.

[1] FiXme Dødelige: futurework: skriv at vi kunne udvide workload balancer saa den ogsaa tager hoejde for deadlines

Criterion	***Definition***
Usable	The end user can easily use the system
Secure	Precautions against unauthorized access
Efficient	How well the resources available are being used
Consistence	How correct the data in the model is
Reliable	The degree of the systems accessibility
Maintainable	The cost of locating and fixing system errors
Testable	The cost to ensure performs intentionally
Flexible	How easily the system can be setup to fit the structure of an institution.
Comprehensible	How easy it is for the end user to understand the system
Reusable	The potential for using parts of this system in another system
Portable	How much effort needed to change the platform of the system
Interoperable	How well the system cooperates with other systems

Figure 5.1: *Definition of criteria*

	Very Important	Important	Less Important	Irrelevant	Easily Fulfilled
Usable		✓			
Secure			✓		
Efficient				✓	
Consistence		✓			
Reliable			✓		
Maintainable				✓	
Testable		✓			
Flexible	✓				
Comprehensible		✓			
Reusable				✓	
Portable				✓	✓
Interoperable			✓		

Figure 5.2: *The criteria with a priority*

Usable It is important that our Hopla Helpdesk is user friendly because it can be used in any organization, it is however not very important since we during this project primarily will tailor it to the university's system. Further more we are more concerned with the functionality of the system than the usability, hence; important

Secure For the Hopla Helpdesk security is less important. We want to distinguish between clients staffs and admins – the clients are e.g. not allowed to solve problems or choose who should be assigned to what problem. We do however not have any sensitive information, so we take no measures to prevent data interception or any other serious security flaw.

Efficient We do not care for the efficiency of our system, but only that it works which lead us to irrelevant for this criterion

Consistence It is important that end users can see which problems are solved and which are not. Further more the Hopla Helpdesk model should not contain duplicates, since it could compromise the integrity of the statistics which the system generates. This lead us to prioritize consistence as important.

Reliable The reliability of the Hopla Helpdesk system is not of great interest to us. We do not actively do anything to increase the reliability of the system, neither do we intensionally decrease it. We want to pay our attention to other criteria instead, therefore this criterion is prioritized less important.

Maintainable Since we not intend to maintain the system after it is finished, it is prioritized as irrelevant.

Testable We want our system to work and to make sure it does, we will run tests. Therefore we want our system to be testable.

Flexible It is very central to our system that it is flexible, because we want it to be generic – that it can be adapted to any organization without much or any cost. To insure this we added features to manage departments, manage persons, manage categories, and manage tags at runtime. We have chosen this to be a very important criteria.

Comprehensible Since the Hopla Helpdesk system is supposed to be generic, it should be easy for the user to understand it. However our focus is primarily on functionality, so we have prioritized it important.

Reusable Since we do not care much for the system after it is deployed, we do not care whether or not it is reusable, hence irrelevant.

Portable Our systems portability can be divided into two, the client side and the server side. We do not care about the portability of the server side, because we will rather focus on the portability on the client side and the functionality of the system. Therefore it is considered irrelevant. The end users will access our system through a browser, so we assume that it can be easily fulfilled since there are many browsers for different platforms. [8][7]

Interoperable If it is possible we want to be able to use an existing database for authentication to our system. This is however the only other system which we plan on cooperating with, therefore it is prioritized less important.

The purpose of our system is defined in this chapter followed by the quality goals which our system should fulfill when it is deployed.

6 Technical Platform

This chapter describes which choices we have made with respect to equipment, system software, system interfaces, and design language. Further, the alternatives which we have considered in the decision process are also shown. Our decisions are primarily based on the fact that our system definition in section 2.2 should hold true, and the quality goals which are prioritized in section 5.2.

6.1 Equipment

The equipment needed to power our software can be any computer with a reasonable amount of processing power, and RAM storage together with at least one NIC(Network Interface Controller) to connect to the network.

We will be using a Dell Optiplex 960 with a Core2 Duo CPU, E8400 @ 3.00GHz, with 4 GB RAM installed. We chose this setup because we have the opportunity to borrow it from our IT department at AAU.

6.2 Software

Our system relies on two things:

- Database
- Web server

These two are described in following subsections.

6.2.1 DBMS

Using an external DBMS allows it to be installed across multiple machines instead on a single computer. This is a good thing as it makes our system more portable, allowing it to be either installed on a single computer, or more computers.

The differences which we have considered between these two DBMS's are listed in figure 6.1[9]

Feature	**PostgreSQL**	**Microsoft SQL Server**
Accessible	BSD Open source	Requires license, but we do have this through the university.
Cooperative with our tools	Hard to incorporate into Visual Studio 2010.	Easy to incorporate into Visual studio 2010.
Experience in our work group	We have had a course based on PostgreSQL.	None, but resembles other SQL DBMS's.

Figure 6.1: *DBMS's compared*

The Hopla Helpdeskrequires a DBMS supporting the query language SQL and relational database model[15]. We have chosen Microsoft SQL Server version 10.50.1600.1 in favor of PostgreSQL, which we have past experience with due to a university course. However PostgreSQL is less integrated with Visual Studio 2010, which we are using for developing our system.

6.2.2 Web Server

We have chosen to implement our system as a webapplication, as it minimizes the amount of software which the users of our system has to install before using our system. The language of choice will be C# using the ASP.NET MVC2(Model-View-Controller) framework, therefore should the webserver support this.

We could have used other web-frameworks, but chose ASP.NET MVC2 because we could use Visual Studio 2010 and C# which we have all used during a course in object oriented programing.

We also discussed not using a framework at all, and simply build everything from ground. We chose not do this because we assessed it is best to use what is already available, further more the framework has been developed for a long time and is therefore likely to be more effective and secure than what we could make within in this limited timeframe.

6.2.3 Operating System

It is a quite natural decision to choose a operating system from Microsoft, as we have chosen a Windows based programming language, C#. See section 6.3 for more. We are using Microsoft Windows server 2008 R2 Enterprise.

6.3 Design Language

As we have chosen to develop a webapplication, we are limited to the programming and markup languages from that domain. Besides the basic markup language HTML, the language of choice will be C# using the ASP.NET MVC framework. When using the Object Oriented Analysis & Designmethod, it comes natural to split our system in parts, called model, view and control.

Combined with our choice of using C#, it becomes a quite natural to choose ASP.NET MVC.

Further more we have all followed a course in object orient programming, where we used C#, so we all have a fundamental understanding of that language compared to other languages, e.g. php, where some of us have a lot of experience and others no experience.

6.3.1 Coding Standard

All functions, properties, and classes will follow big camelcase naming convention, as well as public variables. All private variables names will as seen in code snippet 6.1 be prefixed by an underscore and use small camelcase. Input variables are like private variables except for the absence of the underscore.

```
class SampleClass
{
    // private variables should be written as smallCamelCase
        // with an underscore as prefix
    private int _privateVariable;

    // public variables should be written as BigCamelCase
    // without an underscore as prefix
    public int PublicVariable;

    // properties should be written as BigCamelCase
    public int DummyProperty
    {
        get { return _privateVariable; }
        set { _privateVariable = value; }
    }

    // functions should be written as BigCamelCase
        // input variables should be written as smallCamelCase
    public int DummyFunction(int inputVariable)
    {
        _privateVariable = inputVariable;

        return _privateVariable;
    }
}
```

Code snippet 6.1: *SampleClass.cs*

This chapter describes which choices we have made with respect to equipment, system software, and design language. The choices which we have decided among are all presented as well.

7 Architecture

This chapter shows the layered architecture and the server-client architecture of our system. Both of these pattern architectures are described along with our own use of these patterns in our system.

7.1 Component Architecture

With basis in our system definition in chapter 2.2 and our evaluation of criteria in section 5.2 we found that our main priority is flexibility. As a part of obtaining high flexibility we will use ASP.NET Mvc 2 framework, which is a model view controller design pattern based framework. The database will be access by the system though the ADO.NET data provider. This gives us a component design as seen on figure **??**.

Model The Models responsibility is to represent the objects and their data which is stored in the database, fetched from the database. The result is an interface which enables the controllers and view model] to access the data in the database in a nice and comprehensible way. This component is further described in 8.1.4.

View Model view modelc]s are dependent on the model since it is a container of elements from the model.

View This components' responsibility is to structure the information and layout in the GUI. The interfaces consist of buttons, links, forms and data from a controller, the model, or a view model]. It is dependent on the model, controller and view model]. It is dependent on the controller because the views responsibility is to make buttons and links which have to match the correct controller. It is dependent on the model and view model] because this is the data it needs to present. The dependency on the model could be removed and then all view data has to be parsed though a view model], but we allow this because it is not always necessary to put the model into a view model].

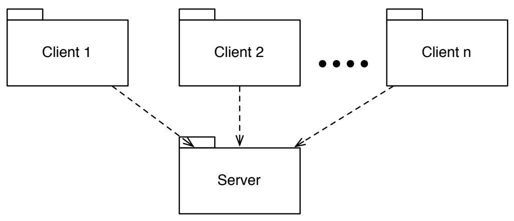


Figure 7.1:

Controller The controllers all have independent responsibilities, but the general responsibility of the controller components is to process the data in the model and return this to a specific view, possibly through a view model]. The controller depends on view, model, view model] and tool. Changing any of these components will most likely result in a change of the controller.

Tools The tool component consist of various help functionality. For example the problem distributer and the search functionality will be placed in this component. The tools will be used by the controller and it relies on the model.

ADO.NET This component will not be build by us, this is an already made component which we will use as a database data provider.

7.1.1 Client-Server

Because we are designing a help desk, we are dealing with users who are not present in a specific location. Therefore we also designed the system using a client-server architecture. On the client side there is a user interface and on the server side the functionality and model are located as seen on figure 7.1. By using Local Presentation [10][p. 200] we enable clients to access the system from anywhere, and still keep the functionality and model on the server and thus keeping the system flexible.

The architecture of our Hopla Helpdesk system is described in this chapter along with standard patterns which we have used.

8 Components

This chapter describes the internal structure of the components which comprise our system and a definition of each class within each component. In short this chapter present the entire system on a class level.

8.1 Structure

There are four overall components: User interface, system interface, function component, and model component. This section describes how these components internal structure is. A subsection for each component is presented below.

8.1.1 User Interface Component

This component consists of three sub-components: Client Interface, Staff Interface, and Admin Interface. Each of these interfaces is described in the sub-subsections bellow.

The sub-components share a login class, which is a window where a user enters his/her user name and password. Depending on the role of the user who is logging in, he/she is directed to the main window of the corresponding interface.

Every window in the sub-components will inherit from a class called "HelpdeskWindow". This class is described in paragraph **??**.[1]

This entire component depends on the function component to gain any functionality. The Authenticator makes sure that a user is directed to the correct Main Window after login, the Problem Handler provides problem search, modifiability, and addition for interfaces, the Admin Interface relies mainly on the Administrator sub-component for administrating person and departments.

Client Interface

The structure of the Client Interface is shown in figure **??**. Each window represents a class and the navigation arrows indicates an association between the

[1] FiXme Dødelige: Dette er en reference til der hvor beskrivelsen af HelpdeskWindow kommer til at være. Indsæt gerne :)

classes. Each object of the different classes will hold a pointer to the object(window) that created it, so it is possible to go back from one window to another.

As mentioned, each window class inherits from the HelpdeskWindow class. Further the Client Problem View inherits from an abstract Problem View class. The reason for this is that there are different problem views – two in the Staff Interface – and making one class that the others inherits from will make the views similar and thereby making the system more comprehensible for the end user.

Staff Interface

The structure of the Staff Interface is visualized in figure **??**. The windows in the figure represents classes of this component, the navigation arrows are translated to associations in UML. Like the Client Interface, the classes in this component

Notice that Staff Main Window aggregates the clients Main Window. Likewise does the Add Solution aggregate the Search Window from the Client subcomponent. If a staff member is an administrator he/she is presented with an administrate button in the main window, which will transfer the user to the Admin Main Window, which is described in the following sub-subsection.

Admin Interface

Figure **??** shows the navigation diagram of the Admin Interface. Every window inherits from the abstract HelpdeskWindow class.

Like the Staff Main Window aggregated the clients Main Window, the Admin Main Window aggregates the staff Main Window, thereby giving the Admin Interface all the possibilities that the two other interfaces has.

8.1.2 System Interface Component

[2] This component is responsible for connecting our system to another systems database, so that our system can use the user names, which already exists in the organization where out system is being implemented. There is no internal structure in this component since it only has a single class which holds the responsibility fr this entire component.

This component is however connected to the Authenticator sub-component in the function component. The connection between these components is that this component is a supplier of data for the Authenticator component. The Authenticator component asks this component to retrieve information about a user in the database and this component provides this information if it is available.

8.1.3 Function Component

The function components main purpose is to provide functionality for the users of the system.[3] This component id divided into three sub-components; Problem

[2]FiXme Dødelige: Det her kan nok gÃ¸res mere udfÃ¸rligt, men ved ikke om det er meningen
[3]FiXme Dødelige: Nødvendigt?

Handler, Administrator, and Authenticator. These are described independently in the following sub-subsections.

Problem Handler

This sub-component is responsible for handling problems – as the name implies. This component consists of only a single class, which yields no internal structure. It is though connected to the Model component, where the systems data resides. The Problem Handler is able to operate on the model, particularly on the Problem class, but also on the Client class, Staff class, and Solution class.

The Problem Handler component can both create, modify, and delete problems as well as assigning problems to staff members. Note however that the Problem Handler does not assign problems by it self, but rather provide the functionality for the Staff Interface. The Solution class is handled in the sense that solutions can be attached and detached to/from problems through this component.

Both the Staff and Client classes are handle through the relations which are between these classes and the Problem class, namely Assignment, Subscription, and Comment. See subsection 8.1.4 for information on Assignment, Subscription, and Comment.[4]

Administrator

In short this sub-component handles everything in the model which the Problem Handler does not. This include adding, modifying, and deleting Person objects, Departments, Categories and Tags. As well as assigning Roles to the Persons.

Like the Problem Handler, this sub-component does not do anything by it self, it simply provides functionality for the Admin Interface.

Authenticator

The Authenticator makes sure that a user is authenticated and can only commit actions appropriate for his or her Role. This sub-component is either connected to an external database through the Login System Interface component or is connected to our systems database through our Model component.

8.1.4 Model Component

The classes of the model component are described in section 3.1. However during revision of our class diagram, we decided to insert more classes. Figure 8.1 shows our revised Model component.

The new classes are: **Comment**, **Tag**, **Category**, and **Admin**. Comment simply holds a comment for a given problem, and remembers the poster of the **Comment**. The Tag class objects can be tied to a problem in order to "Categorize" it. The Tags also belongs to a Category which again belongs to a single Department. This way a Problem with some given Tags, can easily be sent to a Staff member of the Department which the problem is categorized to be in.[5]

[4]FiXme Dødelige: Skal tag også skrives på eller ligger det implicit
[5]FiXme Dødelige: Lidt kryptisk måske..

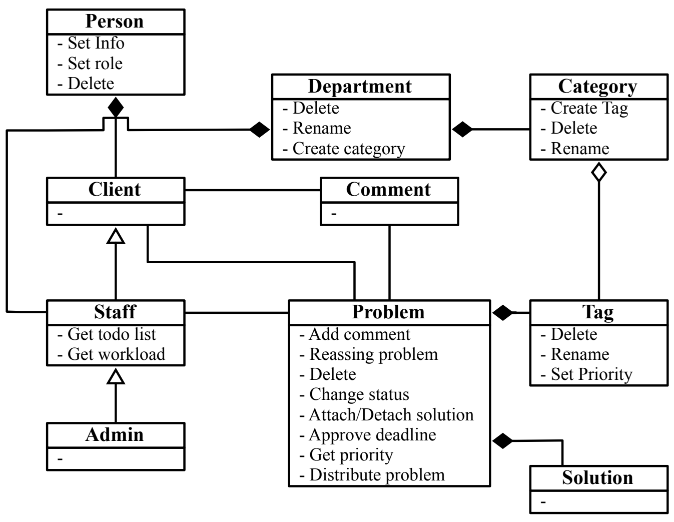


Figure 8.1: *The revised model component. Notice that the attributes of each class is omitted, these are however shown in the E-R diagram in figure 8.2*

Since we want to save our model in a database is is relevant to model our model in an E-R diagram. This is done in the following sub-subsection.

E-R Diagram

In order to get an overview of how to structure the database E-R diagrams gives a good foundation. The E-R diagram can be seen on figure 8.2. The notation for the E-R diagram is based on the citation used in the book Database System Concepts [15, p. 305]. The E-R diagram is based upon the class diagram in figure 8.1.

Every class is turned into a entity and every relation is turned into a relationship [15, p. 259 - 321]. The relationships contain the primary keys from the two entities it is a relation between. Only the relationship prob_sol has an attribute beside the primary keys. This is called time and represent the time for when the solution were attach to the problem, this is used to determine when the problem is solved. The last attached solution to a solved problem must indicate the end of the solving phase.

Any other relation does not need attributes beside the primary keys. It would only be necessary if the same entity could have more than one relation with the same entity e.g. if a system administrates a zoo, then the feeding of the tortoises could be done more than once by the same zoo keeper, and a time stamp would be necessary to avoid duplicates. In our system this problem does not occur at any point e.g. a client can not be subscribed twice to the same problem. This applies to all relations.

The three classes who inherits from each other, client, staff, and admin are combined into one entity named roles. This is due to the fact that a person can only have one role, and therefore this representation is more efficient. This also gives a nice modifiability because adding a new role is simply adding a new

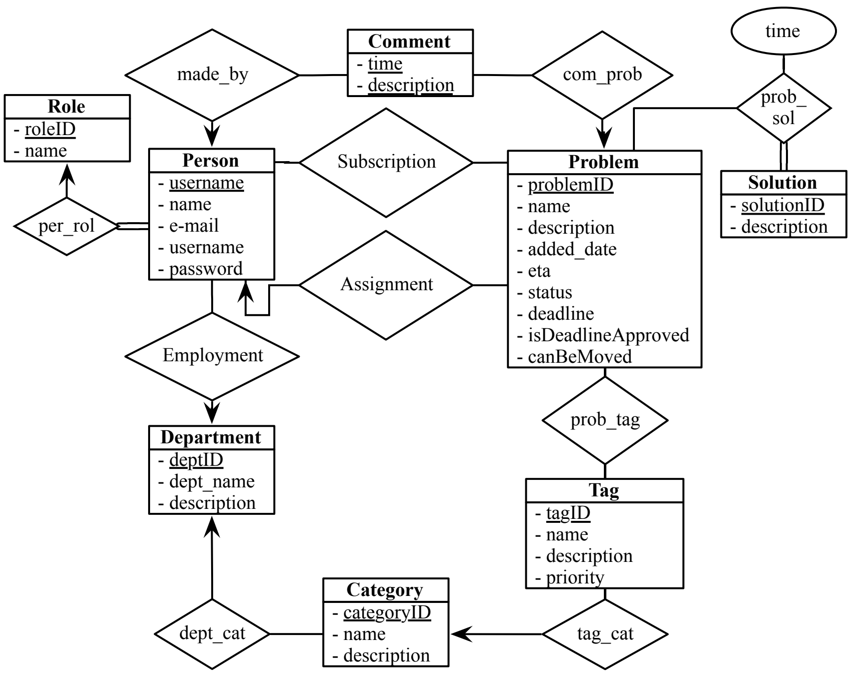


Figure 8.2: *The E-R diagram of our model component*

tuple to the entity.

8.2 Classes

In this section we will give a brief description and list of attributes for each class, and an operating specification of each complex operation.

Account: Actions: Logon Logoff Register Parameters: Username Password Email

Client: Action: View Problem, Search Problems, Parameters: id

Staff: Action: AttachSolution, ListSolutions, DetachSolution, AddSolution, Edit Parameters: Problem Id, Solution Id,

Admin: [6]

Department: [7] Action: Create Department, Edit department Parameters: Id

[6]FiXme Dødelige: Har vi en klasse der hedder admin? og hvis vi har, hvad gør den?
[7]FiXme Dødelige: Virker vores delete?

Category: Action: Create category, Edit category, Delete category, HideUnhide, TagHideUnhide Parameters: Id

Tag: Action: Create, Edit, Delete Parameters: Id

Problem: Action: Categorize new problem, Similar problem, Create, Subscribe, Unsubscribe, Parameters: Id

Home: Action: If there are no users in the system, the home class will create a root user aswell as client staff and admin role. Parameters: dunno

Person: Action: Create, Change Department, Edit person, Choose department, Delete person, Add user to role, Remove user from role, Mail Parameters: id

Reassign: Action: Assign problem Parameters id

Statistics: Action: Show statistics for each department

The internal structure of the three main components – model, function, and interfaces – and their subcomponents is presented in this chapter. This chapter also defines each of classes in our system.

Part III

Implementation

9 Development

Top

9.1 Development Tools

In the creation of our web application we use some development tools which will be described in the following section.

9.1.1 IDE

The primary development tool has been Microsoft Visual Studio Ultimate [6], which include a large variety of inbuilt tools for unit testing, SQL and data modeling. None group members had previous experience with development using Visual Studio Ultimate. The alternative were MonoDevelop [12], which is a open-source cross platform .NET IDE, but since none had experience with this tool either it were not preferred. As most group members work at a daily basis on Windows Based pc's Visual Studio is the favorite choice of IDE.

9.1.2 Collaboration

For collaboration we used Subversion(SVN) and for the code sharing we used AnkhSVN [1] together with SVN. AnkhSVN is a source control provider for Microsoft Visual Studio. Alternatively we could have used Team Foundation Server [2], but this requires installation and configuration of a Team Foundation Server. We chose AnkhSVN since we already had a running SVN server.

9.1.3 Database

As our main data storage we use a Microsoft SQL Server. We choose this data storage vendor since it is compatible with ADO.NET Entity Data Model Designer and Visual Studio Ultimate comes with a inbuilt Microsoft SQL Server manager. Which allows for editing the SQL server from our workstations and not only from the server itself. We considering using postgreSQL, but using postgreSQL with C# and Visual Studio required a plugin in order to use the ADO.NET Entity Data Model Designer. As a database vendor neither choice would not give us any advantages on the data layer, since the required functionality is supported by both systems.

9.2 Development Method

In this section we will describe a few different Software delvelopmentmoethods that we considered using.

9.2.1 Waterfall

The waterfall method, first published by Dr. Winston W. Royse in 1970, has derived its name directly from the concept of the method. If following this method in it's traditional form, development will go through several completely separated steps, from which it is never possible to go back (see 9.1). These steps will not necessarily be done by the same development teams which means that everything has to be documented thoroughly before starting the next step.

Royce originally ment for this method to be developed into a fully iterative model, however, most software developers adapted to non-iterative form. [14]

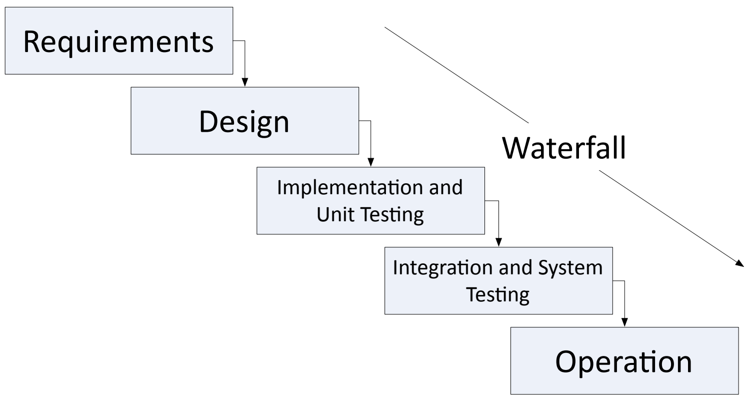

Requirements
Design
Implementation and Unit Testing
Integration and System Testing
Operation
Waterfall

Figure 9.1: *This illustrates a traditional waterfall in which the "water" can only proceed downwards untill reaching the end (process completetion)*

9.2.2 Agile Methods

Tail

10 Program Presentation

This chapter outlines and present the process of using our system from the three points of perspective, based on the three user roles. This is done to show that our systems usage is consistent with the use cases we presented in the Application Domain Analysis, chapter 4.

As described there are three roles a user of our system can have:

- Client
- Staff
- Admin

As with all users of the system, the first thing which the user is greeted with, is the welcome page, followed by the login screen. After that, the path is split up according to the role the user has. All the pages of our system shares the same master file, particularly the menu which holds the functionality of the logged in user. The shared master gives a top of every page, which is seen on figure 10.1. The points in the menu changes depending on which privileges the current user has.

Below the most common usages of our system is described. These are based on the use cases from our analysis in section 4.1.2. We will start by presenting the clients usage.

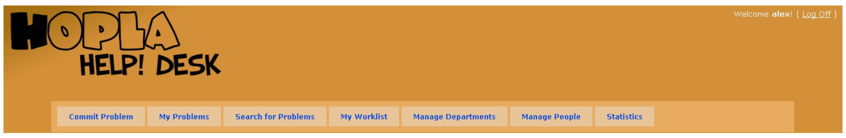


Figure 10.1: *The menu which the master file is responsible for making. The Commit Problem, My Problems, and Search for Problems are menu points which are allow for clients. The My Worklist menu point is a menu point for the staff users. The Mange Department, Mange People, and Statistics menu points are allowed for admins only*

10.1 Client Usage

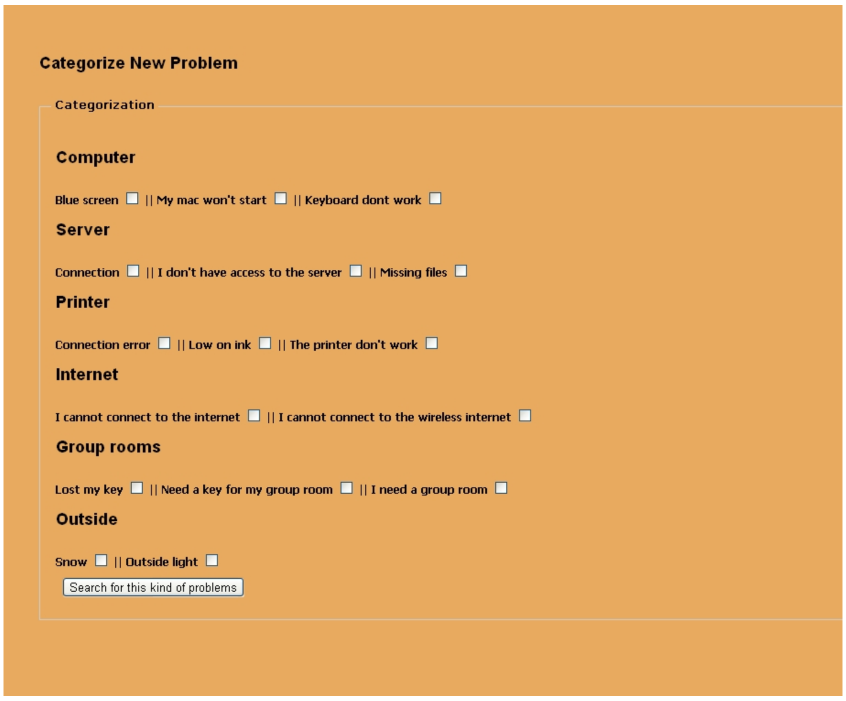
The client can generally do three things; commit a problem, see status of his/her problems, or search the database for problems. The first two are based on their corresponding use cases in section 4.1.2. The search function was added as a functionality for client because we assume that it would primarily be persons with a new problem who might want to search for others problems.

The following subsections describes the three usage of the system which the client has access to. The process of searching for problems and seeing status of the clients problems have switch places in the subsections below because in order for a client to check the status of his/her problems a search will be initiated.

10.1.1 Commit a Problem

The process of committing a problem consists of three steps: Categorizing the problem, considering existing problems, and if no problems suffice the to match the new problem; a creation step.

Categorizing the Problem


Categorize New Problem

Categorization

Computer

Blue screen ☐ || My mac won't start ☐ || Keyboard dont work ☐

Server

Connection ☐ || I don't have access to the server ☐ || Missing files ☐

Printer

Connection error ☐ || Low on ink ☐ || The printer don't work ☐

Internet

I cannot connect to the internet ☐ || I cannot connect to the wireless internet ☐

Group rooms

Lost my key ☐ || Need a key for my group room ☐ || I need a group room ☐

Outside

Snow ☐ || Outside light ☐

Figure 10.2: *The categorization step in committing a problem, note that the page has been cropped*

The process of committing a problem starts with a click on the Commit Problem menu point, which can be seen in figure 10.1. When Commit Problem is clicked the user is asked to categorize the problem in the window showed in figure 10.2. As the figure shows the tags, the check boxes, are ordered under headlines. These headlines are categories.

Consider Existing Problems

When the problem is categorized the system searches for problems matching the specified tags, the search function is described in 12.2. The problems found in the search are listed in a table with columns showing number of matching tags, deadline, estimated time of completion, and title and description. The last column also shows whether the given problem is solved or not.

Create New Problem

The client can choose a problem(or even more problems) to subscribe to in order to receive notifications when the problem is updated, e.g. a solution is attached to it. The client can also choose to write a new problem if none of the existing problems suffice to match his/her problem. The screen showed when creating a new problem is seen in figure 10.3 The categorization part of this page remembers what was entered in the categorization step, but allows the client to change it if he/she wants to reconsider the chosen tags. The tags which were marked in the categorization step are remembered and are also marked when this step is started.

When the problem has been described, given a title it can be created. Optionally the client can give the problem a deadline for the staff member assigned to the problem to consider. When the problem is created the client is presented with a page showing whether or not the problem was successfully added to the system. To see the problem later, the client can either search for it, as described in subsection 10.1.2, or go to the My Problems menu point, which is described in subsection 10.1.3.

10.1.2 Search for Problems

To search for problems, the client must select the menu point called Search for Problems, which can be seen in figure 10.1. The page which is then shown can be divided into two fields; search field and result field. The search field is similar to the categorization step of the commit new problem usage, which can be seen in figure 10.2 except that a few more options, namely: Only my problems, only unsolved problems, only solved problems, and minimum number of problems to find. The three first are represented as check boxes and the last is a text field, which accepts a number above 0.

If the only my problems check box is checked, the result will only show problems which the client is subscribed to. The only solved problems and only unsolved problems check boxes does as their names indicate. If both check boxes are check, no problems will be found of course.

The minimum number of problems is used as an input to the search function, which specifies how many problems should be found before the search function stops running. The reason for this is that the search is done stepwise, lowering the bound for similarity at each step. In order to make the search be fast it then need a number which indicates that the size of the result is satisfactory. The search function is described more thoroughly in section 12.2.

The result field displays the problems found by the search function. This list is initially empty, because the search button must be clicked in run a search. The way the problems are displayed resembles the table in the Consider Existing Problems step in section 10.1.1 except that it does not have a column with the number of matching tags but instead has a column with the time it was solved – if the given problem is solved. The problems in the list can be clicked to enter the details of the given problem, this detail view is described in the following sub-subsection.

Problem Details

The properties of a problem can be seen in the problem details view in figure 10.4. This view also allows the client to add comments to the problem and subscribe or unsubscribe to the problem. If a client is subscribing to a problem, he/she will receive an e-mail every time a solution is added to the problem or when a comment is added.

The details which can be seen in the problem details are: The title, description, added date, assigned to staff member, ETC, deadline approval, solved time, categories and tags, and if the deadline is approved; the actual deadline. If there is any solutions attached to the problem, these are shown in this view. The same applies to comments.

10.1.3 See Own Problems

For a client to see his/her problems, he/she has the My Problems menu point, which can be seen in figure 10.1. This actually initiates the search for problems process as described in subsection 10.1.2. The difference is the headline and the fact that the only my problems and only unsolved problems check boxes are checked. Also a search is initiated when the menu point is selected. It is still possible to specify your search further and change the specification of the search, i.e. it works the same way as the search usage described in subsection 10.1.2 from this point on.

10.2 Staff Usage

The staff members main usage of the is to solve problems. This is corresponding to the solve problem use case in section 4.1.2. This use case is how ever quite large, so it is divided into three steps; choosing a problem to solve, communicate with the subscriber(s), solve the problem. The communication with the subscriber(s) step might not be applied during the solve usage of the client. The three steps are described in the following subsections.

10.2.1 Choosing a Problem to Solve

When a staff member wants to solve a problem he/she clicks the menu point called My Worklist which can be seen in figure 10.1. This shows the worklist of current logged in staff user. An example of a worklist is shown in figure 10.5. The staff member can see the title, description, deadline, priority, and

ETC. He/she can click on a problem in order to get a more detailed view of that specific problem. The staff members are responsible for choosing the right problems in the right order them selves, but they can use the priority as a guideline for which problems should be solved first. This is made easier by the system because the list of problems is ordered by the priority of the problems.

It is from problem details view that the staff member can communicate with the subscribers, accept the deadline, lock the problem to him/her self, attach solutions, and finally declare the problem solved. Even though this view share name with the problem details that a client can access, the staff members do have more options, as described above. The staff members problem details view can be seen in figure 10.6. To accept the deadline a staff member simply checks a check box and submit the change. When a staff member has chosen problem to start working on, he/she can make it non-reassignable by unchecking the reassignable check box. This will stop the balance workload function, which is described in section 12.3, from removing that given problem from the staff member while he/she is working on it. If the staff member does not do this, the problem might be reassigned to another staff member and thereby reducing the effectiveness of the recourses because to staff members will be work on the same problem.

The staff members can reassign problems which have been assigned to them. When doing so, he/she can either reassign directly to another staff member or to a department. In either case the problem will be marked non-reassignable, in order to stop the balance workload function from moving it to another staff member or back to the staff member who had it in the first place.

10.2.2 Communicate With Subscribers

To communicate with the subscribers, the staff member can write comments to a problem, which the subscribers can see and again respond to. This communication is important because the staff member might need more details about a problem in order to solve it.

10.2.3 Solve the Problem

Through the staff members problem details it is possible to attach an existing solution or write a new one. Once a problem has been given a solution the subscribers are notified and they can check if the solution actually solved their problem. They can then write a comment back telling the assigned staff member whether or not the solution solved the problem for them. If the subscribers are content with the solution(s) the staff member can mark the problem as solved and enter the time he spend on the problem, so that future problems similar to the given can be given an estimation.

If a staff member wants to use an already existing solution he will have to search the database for it. Here is used a search view very similar to the one which the clients have, which is described in subsection 10.1.2. The difference is that the staff does not have the opportunity to choose any of the following options: Only my problems, only solved problems, only unsolved problems, and minimum number of problems. The reason for this is to make it simpler and faster to find the solution. When the staff member finds a problem with the right solution, he/she can attach this to the problem currently being solved.

To write a new solution the staff member clicks the corresponding link in the staffs problem details view. He/she is then presented with a text area and a create button. When the button is clicked, a new solution is created with the given text is attached to the problem being solved. The problem is not marked as solved as soon as a solution is attached to it, because the solution might not actually have solved the problem or the staff member might want to attach more than one solution before he/she wants to mark it solved.

10.3 Admin Usage

The use cases administrate and statistics, which are described in section 4.1.2, are the two main usages of the system which the admin will use. The administrate use case is divided into two usages; manage people and manage departments. These three usages are described in the following subsections.

10.3.1 Manage People

For an admin to manage the people in the system – or rather the peoples users – he/she can click the menu point called Manage People which can be seen in figure 10.1. The screen which the admin is then presented with a table of all the users in the system. The table includes: Actions on the user, id number, e-mail, department name, workload, and the roles of the user. The department name and workload is only available for staff users, because they are the only ones with a department and a workload. Other users simply gets "N/A" in these cells of the table. The actions on the users are: Edit, delete, reset password. This is done by clicking the corresponding link next to the user which the action is to be invoked on.

The edit link takes the admin to the edit view of the corresponding user. From here he/she can see the id and user name of the user, and can edit the e-mail, department, and the roles of the user. The department can only be edited if the user is a staff member. If a non-staff user is given the staff role, the admin is asked to give the user a department, because we do not want to staff members without a department.

The deletion of a user will result in permanent removal of the user from the system. This can only happen if the user is not subscribed to any problems and is not assigned to any problems.

The reset password action initiates a random password generator to set as the new password for the given user. After that an e-mail is sent to the user containing the newly generated password.

10.3.2 Manage Departments

The menu point Manage Departments which is seen in figure 10.1 allows the admins to administrate the departments, categories, and tags in the system. The view shown when this menu point is clicked, the admin is presented with a list of every department in the system along with a link to create a new department.

When creating a new department, the admin creating it must provide the name of the department and a description. A newly created department is initially empty, i.e. it contains no categories and no staff members are part of it. To add categories and staff members an admin can select the department in the list of the Manage Department view.

When a department is selected, the admin is presented with a view similar to figure 10.7. The view contains to lists; one with the staff members associated with the department and one with the categories associated with the department. Links to add a new category and to attach staff member to the department are also provided in this view. These two parts of the department view are described in the following sub-subsections.

Manage Staff Members

When a staff member is selected, the admin is redirected to the person editor of that staff member, which is described in subsection 10.3.1. To add a staff member to the selected department, the admin can click the Add staff link in the department view. This renders a view with a table of every person in the system who has the staff role. The table resembles the table described in subsection 10.3.1 except that the id for the staff members are not shown.

Manage Categories

To create a category, an admin clicks on the corresponding link and provides a name and description for the category. Like a newly created department, the new category is empty, thus containing no tags. To change the new category – or another one – an admin can click on it in the department view for the department which the category belongs to. The view which is then presented shows the id, name, description, department, and whether or not the category is hidden. Further more the category can be deleted, hidden, and edited from this view. Hiding a category will hide all the tags associated to that category. If the category is hidden it can be unhidden(revealed), which will unhide all the tags associated to the category. To delete a category, every tag must be removed beforehand. It is possible to create new tags from the category view, these tags will initially be attach to the category from which they were created.

During creation of a tag, the admin most provide a name for tag, a description, and a priority. The priority of a tag is a number which indicates how important a problem with the tag is relatively to problems with other tags. There is no inbuilt scale, if the organization using out system wants their tags to vary in priority from 0 to 10 they can do that, the only constraint is that the priority is a 16 bit integer, resulting in a range from 2^{15} to $2^{15} - 1$.

In the category view, the tags can also be individually be hidden, edited, viewed in details, and deleted. Hiding a tag means that it will not be shown in e.g. the problem search usage of the clients. When editing a tag, the admincan set the same properties of the tag as when a tag is created. The the detailed view of a tag shows the properties of the given tag, namely: Id, name, description, priority, category id, number of solved problems, time consumed, average time spend, and whether or not it is hidden. the time consumed indicates the time, in minutes, for how long every problem with this tag has taken to solve. It is used along with number of problems solved to calculate the average time spend to solve problems with that tag. This is again used to calculate the estimated time of completion of unsolved problems.

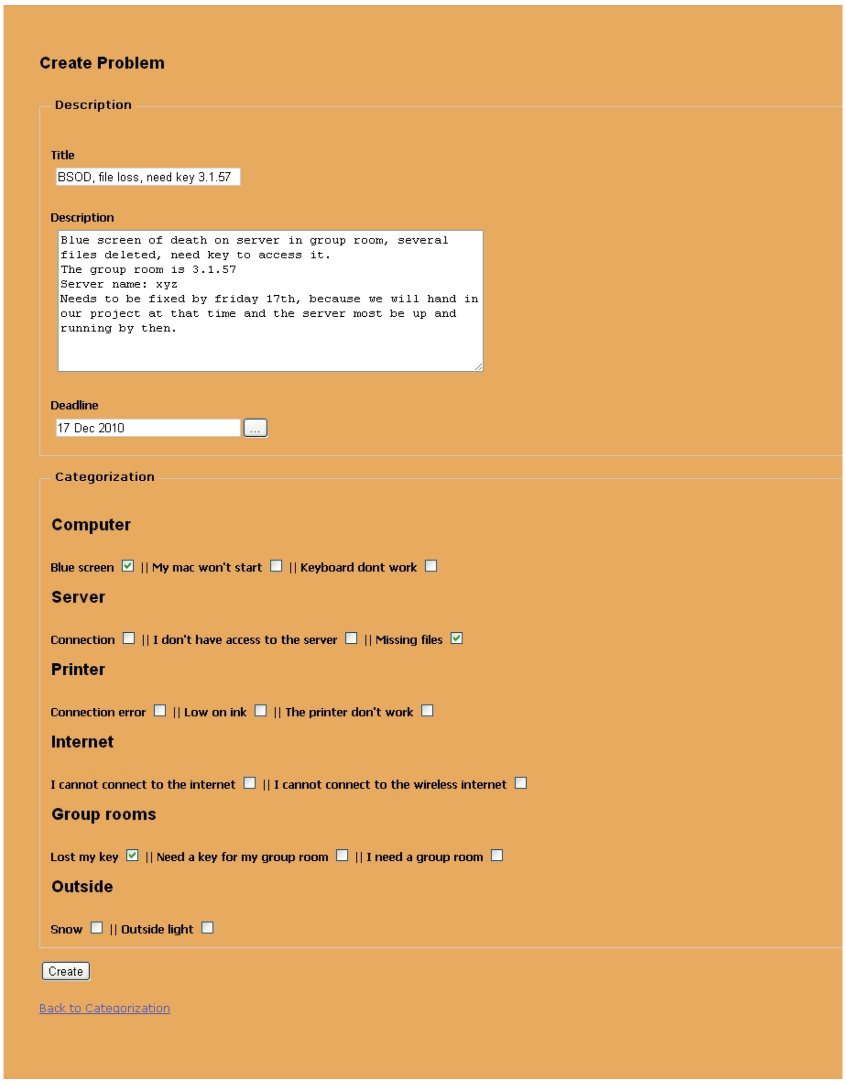
To delete a tag, no problem can be associated with it. The deletion of tags is only supposed to be used if a tag is created incorrectly, e.g. another tag already exists or it is created in the wrong department.

10.3.3 Statistics

An admin can get statistics about how long problems in average remain unsolved. To do this he/she must select the Statics menu point as seen in figure 10.1. He/she is then presented with a view which shows the average time for a problem to remain unsolved for each particular staff member, each department, and for the system over all. It also shows the average time for problems to be solved for the last week for all of the three groups above.

These statistics can be used in any way the admin wants, e.g. to see if a particular department needs more staff members if that given department is generally slower than the others. It is only admins who are allowed to see this page because clients might use the statistics to categorize their problem, so that it will end up in the department which is generally fastest to solve problems, but this will actually make the problem be solved slower if the staff member has to figure out where to reassign it. The staff members can neither see this page in order to avoid internal competition for fastest solve time, because it could lead to a lack in quality of the solutions.

This chapter outlines and present the process of using our system from the three points of perspective, based on the three user roles.



Create Problem

Description

Title
BSOD, file loss, need key 3.1.57

Description
```
Blue screen of death on server in group room, several
files deleted, need key to access it.
The group room is 3.1.57
Server name: xyz
Needs to be fixed by friday 17th, because we will hand in
our project at that time and the server most be up and
running by then.
```

Deadline
17 Dec 2010

Categorization

Computer

Blue screen ✓ || My mac won't start ☐ || Keyboard dont work ☐

Server

Connection ☐ || I don't have access to the server ☐ || Missing files ✓

Printer

Connection error ☐ || Low on ink ☐ || The printer don't work ☐

Internet

I cannot connect to the internet ☐ || I cannot connect to the wireless internet ☐

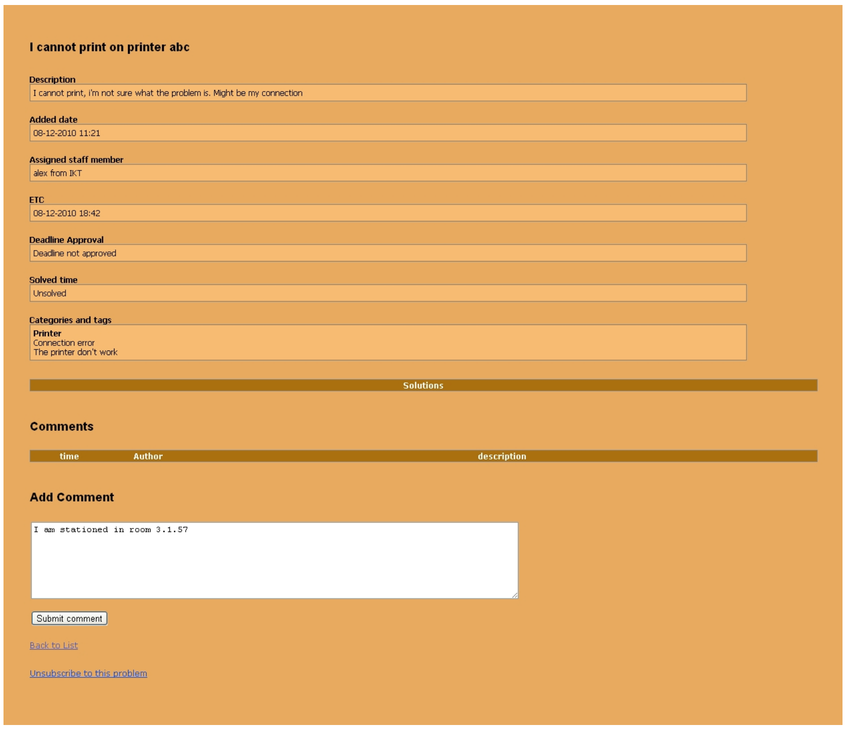
Group rooms

Lost my key ✓ || Need a key for my group room ☐ || I need a group room ☐

Outside

Snow ☐ || Outside light ☐

Figure 10.3: *The create step of committing a new problem, note that the page has been cropped*



I cannot print on printer abc

Description	I cannot print, i'm not sure what the problem is. Might be my connection
Added date	08-12-2010 11:21
Assigned staff member	alex from IKT
ETC	08-12-2010 18:42
Deadline Approval	Deadline not approved
Solved time	Unsolved
Categories and tags	**Printer** Connection error The printer don't work

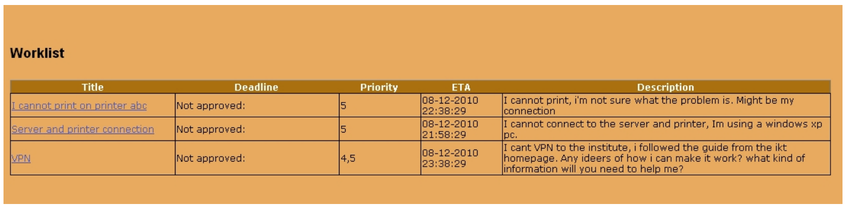
Comments

time	Author	description

Add Comment

```
I am stationed in room 3.1.57
```

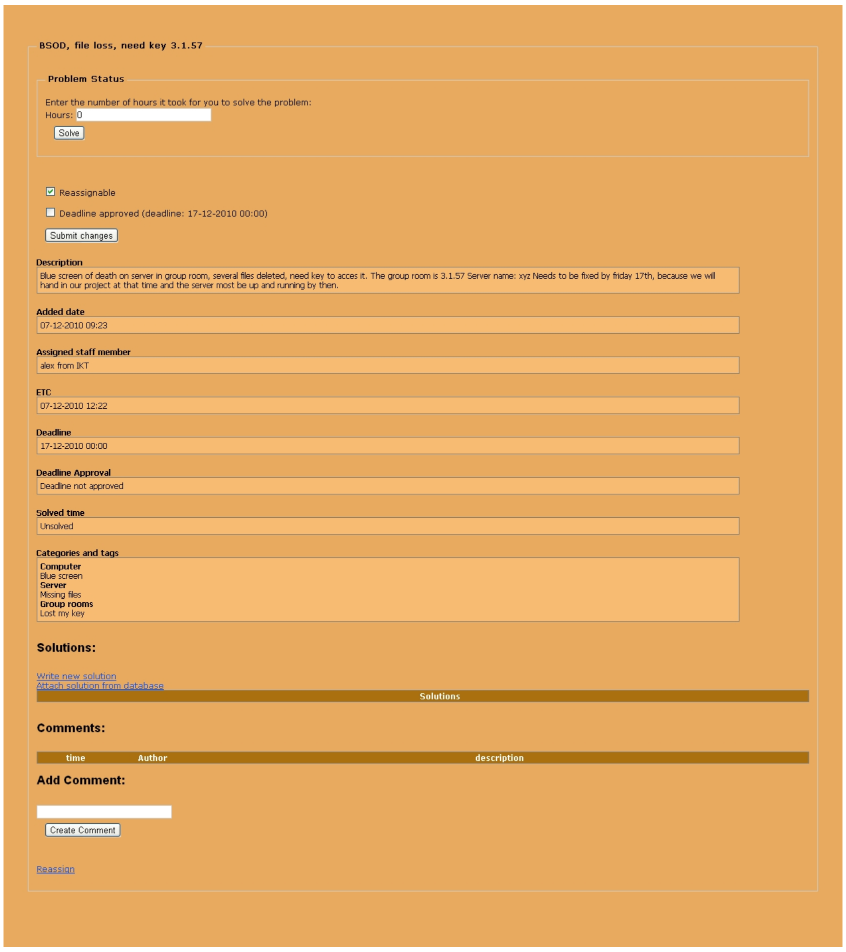
Figure 10.4: *The details of a problem*



Worklist

Title	Deadline	Priority	ETA	Description
I cannot print on printer abc	Not approved:	5	08-12-2010 22:38:29	I cannot print, i'm not sure what the problem is. Might be my connection
Server and printer connection	Not approved:	5	08-12-2010 21:58:29	I cannot connect to the server and printer, Im using a windows xp pc.
VPN	Not approved:	4,5	08-12-2010 23:38:29	I cant VPN to the institute, i followed the guide from the ikt homepage. Any ideers of how i can make it work? what kind of information will you need to help me?

Figure 10.5: *A staff members worklist*



BSOD, file loss, need key 3.1.57

Problem Status

Enter the number of hours it took for you to solve the problem:
Hours: 0

☑ Reassignable

☐ Deadline approved (deadline: 17-12-2010 00:00)

Description
Blue screen of death on server in group room, several files deleted, need key to acces it. The group room is 3.1.57 Server name: xyz Needs to be fixed by friday 17th, because we will hand in our project at that time and the server most be up and running by then.

Added date
07-12-2010 09:23

Assigned staff member
alex from IKT

ETC
07-12-2010 12:22

Deadline
17-12-2010 00:00

Deadline Approval
Deadline not approved

Solved time
Unsolved

Categories and tags
Computer
Blue screen
Server
Missing files
Group rooms
Lost my key

Solutions:

Write new solution
Attach solution from database

Solutions

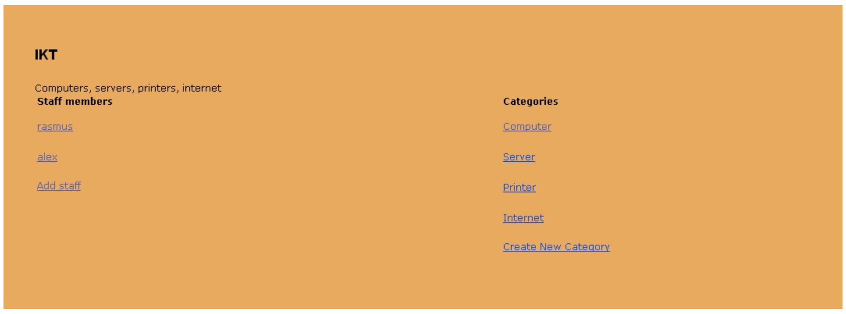
Comments:

time	Author	description

Add Comment:

Reassign

Figure 10.6: *A staff members problem details view*



IKT

Computers, servers, printers, internet

Staff members	**Categories**
rasmus	Computer
alex	Server
Add staff	Printer
	Internet
	Create New Category

Figure 10.7: *The department view, note that the screen shot has been cropped*

11 Enviroment

hvad og hvorfor

11.1 MVC Framework

The Model-View-Controller (MVC) was originally designed by Trygve M. H. Reenskaug in 1979, who was at that time working for Xerox PARC. The goal was to create an environment in which the users (developer, customer ect.) could preserve the original perception of the data structure, while being able to view and edit portions of it.

11.1.1 Components

The (MVC) framework consists of three different types of components; Views, controllers and models. Each created with a different purpose, see figure 11.1

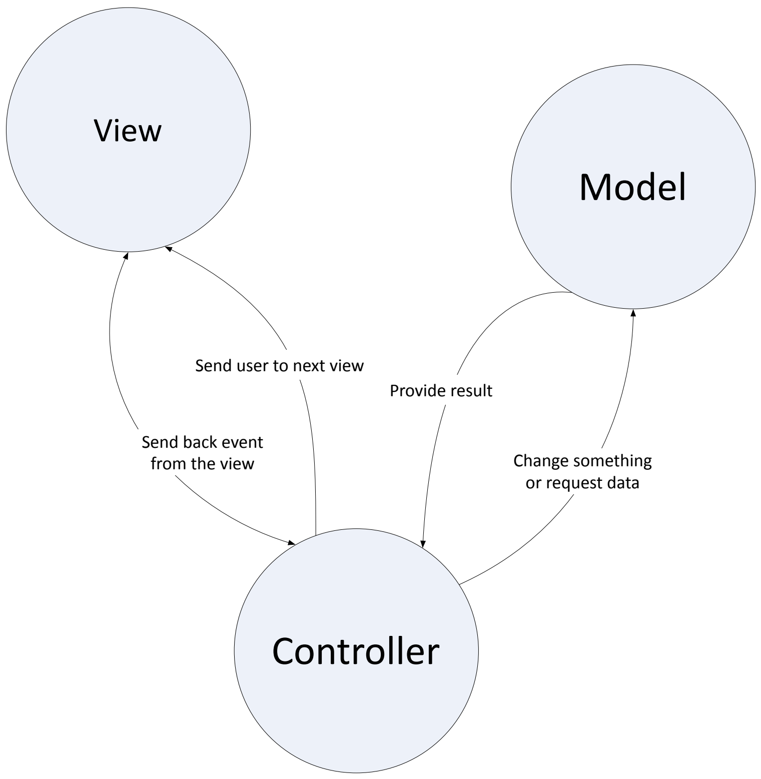

View
Model
Send user to next view
Provide result
Send back event from the view
Change something or request data
Controller

Figure 11.1: *These are the three main components in the MVC*

- Models

The models are the components that handle the data domain of the application, it is typically mapped to a database from which it reads and writes data. This way developers can handle the data, without worrying about the actual data connection.

- Views

The views display the data and UI. They are typically created with data provided from the model. These views are full-featured (X)HTML, with the addition of C# or Basic code, to fetch content from the model.

- Controllers

The Controllers reacts to user requests the users provides through the UI. Based on this they decide which view should be rendered. If the data the view should display does not exist in the model (input, ect.) the controller can also provide this data to the view to display.

Master-page

Master-pages is not a part of MVC as a design-idea, it is a component implemented by Microsoft. The master-page is a page that is always loaded when displaying a view. Like the view, it consists of (X)HTML and C#/Basic, however, it also has content-containers. These containers can be created anywhere on the web page, and are filled with content by views.

11.1.2 Structure

An interesting thing about MVC is that there are no actual HTML files, all content is generated generically as the users interacts with the different components within the web page. When entering an URL you actually call methods in controllers, you can even add parameters to the calls. The controllers then redirects you to the correct view based on the input and the method called. The view then executes, which results in a plain web pages that can be displayed in a browser.

Because of the separation into three independent modules, developers are able to create different parts of the application with different kind of approaches, thus enabling them to better manage complexity, create applications with a high maintainability, and create UI that does not have to show the actual data structure, which can often be confusing for an ordinary user of the webpage.

Data validation

Client side validation Server Side validation

Testing

When creating web applications with MVC you have the possibility of creating unit tests to thoroughly test your code. These tests can be automatically generated from within Visual Studio, and run as a seperate function, in order to allow the developer to test for all possible inputs. For a more detailed description of Unit-testing in MVC, look in the section **??**

[**?**] [13] [**?**]

11.2 Database Structure

We distinguish two parts of the database, the login part, and the model part. The model part is created from our Model [1] using the ADO.NET Entity Framework(EF) which is described in subsection 11.2.1. We choose to use ADO.NET EF because we do not have to worry about converting our tuples to objects and to make sure that changed properties are mapped to the database correctly. To administrate users we use the inbuilt ASP.NET membership provider which provides a login system with support for role authorization. We use this provider since it saves time instead of building our own login system. Security is not a big concern in this project and therefore we do not want to spend time creating a secure login system. The major disadvantage is that including the membership providers database scheme with the model is not supported and will not

[1] FiXme Dødelige: Link til MODELLEN sættes ind her.

work proper. Therefore we had to add a person entity and change the register functionality to also save the registered person in our person table. This gives some redundant data, but only the username and email. When we needed to access data from the membership tables we had to use SQL statements and not the object oriented approached we used for the rest. This is limited since our main functionality depends on the model.

11.2.1 ADO.NET Entity Framework

The ADO.NET is designed by Microsoft with the purpose of making a disconnected database architecture to use with the .NET framework [3]. This architecture is an improvement from the older database architectures where a connection were made and held open in the entire runtime. With a disconnected database the connection is only open when data is needed [**?**].

The ADO.NET EF is an object relational mapping framework [5] build upon ADO.NET. Together with the ADO.NET Entity Data Model Designer and a SQL Server ADO.NET EF gives a strong tool for mapping a database and using the data in a object oriented matter, without dealing with the transformation from entity to class and from tuple to object. The Designer is built into Visual Studio Ultimate.

Using the ADO.NET and the Entity Data Model Designer can be done in two ways. The model can be created from an already existing database or by creating the model and generate a database from this [4]. We use the last approach and creates the database from the model. In this way we do not have to worry about setting the right foreign keys and relational tables. Instead we get a fully functional model linked to a database.

Our model as it is in the ADO.NET Entity Model Designer is shown on figure **??**

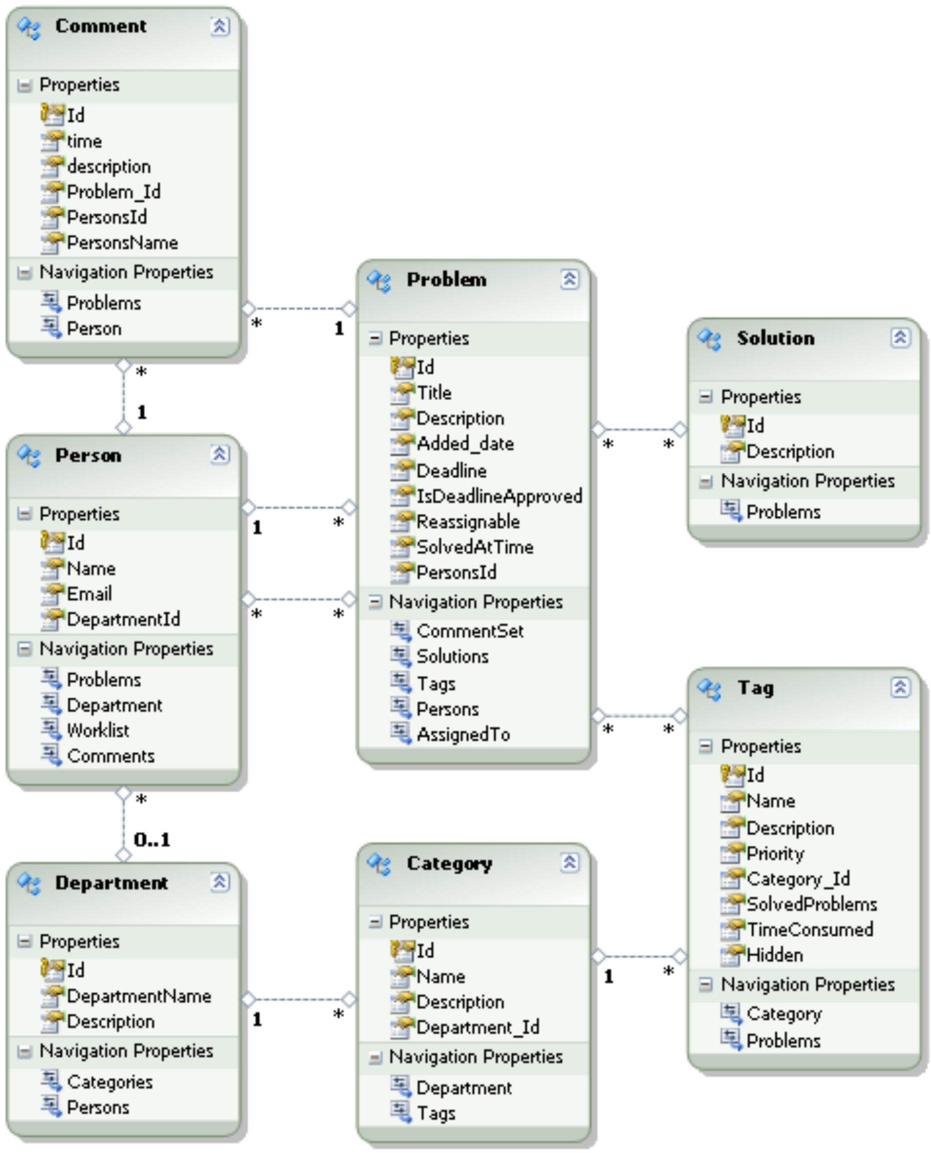


Figure 11.2: *Our model as it is seen in the ADO.NET Entity Data Model Designer.*

Hvad

12 Key Points

This chapter describes the key points in our program. The key points which are included are problem searching and workload balancing. These are chosen because they both are central functions to the entire system. We want to search for problems to avoid clients committing similar problems and to search the database for a specific problem according to the system definition in section 2.2. We have also chosen that our system should balance the workload between staff members in order to provide more efficient solving as we state in our system definition.

12.1 Problem Prioritization

When assigning multiple problems to a staff member, they will appear as a list. Due to the human nature, we might pick a specific order of solving the problems in, which does not necessarily take priority, deadlines and such into account. Therefore, the order of which the problems appear in in the staff members work list, is important.

We have defined two elements which define a problems importance, and therefore its placement in the list of problems. These two elements are:

- Whether or not a deadline is approved
- The priority

We acknowledge that problems which has a deadline should be at least reviewed by the assigned staff member earlier than problems without a deadline.

The list is ordered by priority of the problem, however problems with approved deadlines will always appear on top of the list regardless of their or other problems priority. This splits the list into two parts. Above, priority-sorted problems with approved deadlines, and below priority-sorted problems with or without not approved deadlines.

If a problem with an approved deadline is overdue, then the priority will go up to the maximum value 10, which in turn will make the problem appear on the top of the list.

An example can be seen in figure [1]

[1] FiXme Dødelige: Indsæt figur fra “Program presentation” section der viser det.

12.2 Problem Search

The system needs to be able to search for problems. This search is based on tags. It should find an amount of problems which match the specified tags and order them by number of tags matching. The amount of problems this function will find is depending on a specified number know as “Minimum number of problems”, which determines when the function should stop searching for more problems.

The input for this function is:

- Selected tags
- Problems to search among
- All tags
- Minimum number of problems to find

The *Search* function is called with the parameters above. It calls an *InternalSearch* function which is private, with the same parameters and a compare delegate which determines how the problems should be sorted. The *InternalSearch* function is described in subsection 12.2.1 and 12.2.2. Subsection 12.2.3 describes the *SearchSolvedFirst* function, which takes into account whether or not a problem is solved, when ordering the list of problems to return. It does so by calling the *InternalSearch* function with another compare delegate.

12.2.1 Search for Problems by Tags

The most important part of the *InternalSearch* function is the while loop shown in code snippet 12.1. Generally, this loop finds problems which matches the tags specified in the input to the function and orders them with the problems with the most amount of matching tags in the beginning of the result list. The while loop beginning in line 4 will continue to run as long as there has not been found enough problems to suffice the minimum number of problems input and there still is at least one tag to search for. If there still is not enough problems another part of the search function will take care of this. This part is described in subsection 12.2.2.

The function will increase the number of tags to remove from the tag list which was input to the function every time an iteration ends in the outer while loop; lines 4-32. In the first iteration no tags are removed, this means that the function will find every problem which has every tag which is being search for and put these in the beginning of the result list. Further more the function sort the problems each step by the least amount of tags. The reason for this is that the less tags a problem has which are not searched for, the more likely it is that the given problem matches the search. For example if a search is run for the tags “Computer” and “Harddisk”, the problems only containing these tags will be listed first and if a problem contains the tags “Computer”, “Harddisk”, “Database”, and “Connection” it will be listed further down on the result list because it has unrelated tags attached to it. See a more detailed example of a run of the search function in appendix **??**[2].

[2] FiXme Dødelige: Tilføj dette eksempel eller fjern denne linje

```
 1 .
 2 .
 3 .
 4 while (result.Count < listMinSize && noOfTagsToRemove < tags.Count)
 5 {
 6     tempResult = new List<Problem>();
 7     tagsToRemove = new List<int>();
 8     for (int i = 0; i < noOfTagsToRemove; i++)
 9     {
10         tagsToRemove.Add(i);
11     }
12     try
13     {
14         List<Tag> currentSearch = tags.RemoveCurrent(tagsToRemove);
15         while (true)
16         {
17             temp = allProblems.ToList();
18             foreach (Tag tag in currentSearch)
19             {
20                 temp = temp.Where(x => x.Tags.Contains(allTags.
                       FirstOrDefault(y => y.Id == tag.Id))).ToList();
21             }
22             tempResult.AddRangeNoDuplicates(temp.ToList());
23             currentSearch = tags.RemoveNext(ref tagsToRemove);
24         }
25     }
26     catch (NotSupportedException)
27     {
28         noOfTagsToRemove++;
29         tempResult.Sort(compare);
30         result.AddRangeNoDuplicates(tempResult.ToList());
31     }
32 }
33 .
34 .
35 .
```

Code snippet 12.1: *The while loop which finds and sorts problems matching the input tags*

The inner while loop spanning the lines 15-24 iterates over the tags to remove, this does not have any effect when no tags are to be removed. However if the function gets the tags “Computer” and “Harddisk” as input, we first want to find every problem with both tags, then find every problem with the “Computer” tag, and finally find every tag with the “Harddisk” tag. The order of the last two is not important because they are sorted in a single list, which is then inserted into the result list. The *RemoveNext* function called in line 23 is responsible removing the tags which are not to be search for in the in the next iteration of the inner while loop in lines 15-24. It will throw a **NotSupportedException** when it has removed every combination of tags, which will break the inner while loop and add the problems found in the current search to the result list, which will be returned to the call site later. The function *AddRangeNoDuplicates*, is used instead of the in-build *AddRange* function, because otherwise one problem could be added several times, which is not wanted. One problem should only appear one time in the list returned from this function, because it would simply not make sense in relation to the minimum number of problems, since a single problem would be counted several times towards finding enough problems. Further more the client seaching for problems should not have the same problem appear on his/her list more than once. Therefore at some point in our code we would have to filter out the duplicates, we chose to do it here, because it is the earliest step in finding problems in our database. If we were to filter the duplicates out another place, the search function could potentially return a list containing a single problem several times, which then – when filtered – only yields a single problem and thereby rendering the minimum number of problems nearly useless.

The for-each loop in the lines 18-21 finds all the problems match the current search. The current search is the tags being input to the function without the tags to be removed. The for-each loop removes every problem not containing a specific tag in each iteration, until every tag in the current search is covered.

The initialization of the tags to remove is done in the for loop in the lines 8-11. It sets the first x tags to be removed where x is the current number tags to remove. This means that if e.g. three tags should be removed it will initially be the first, second and third tag, which are removed.

12.2.2 No Tags to Remove

If there has not been found enough problems to suffice the minimum number of problems during the search in tags the function will start to look for problems with no tags at all, then problems with one tag etc. This part of the function will start by finding every problem with no tags and add them to the result list, then every problem with a single tag is found and added. Here the problems are also added using the *AddRangeNoDuplicates* applying the same reasoning as above. This part of the function will continue to run until enough problems are found or it is about to search for more tags then there is in the "All tags" input. This means that it can actually return less problems than minimum number of problems if it cannot find any more, but at this point it has searched every problem, this means that it will actually return every problem in the "Problems to search among" sorted.

12.2.3 Order by Solved

In some cases we want to order the problems by whether or not the problems are solved. E.g. we want to show the solved problems first to clients who are categorizing a problem, which might already exist. The reason for this is that the client should be presented with problems with a solution first, in hope that the client can use a solution and does not need to subscribe to a problem or add a new one.

This search function makes use of the *InternalSearch* function but with a different compare input to the *Sort* function. This compare function sorts first by whether or not a problem is solved, then by the least number of tags.

12.3 Balance Workload

Whenever a staff member is removed from a department or has marked a problem as solved, it becomes necessary to balance the workload of each staff member

Figure 12.1: *A diagram of the balance workload method. Each collumn represents a staff members workload. Each box is a problem. The height represents the estimated time consumption. The problem colored dark grey is not reassignable. There are tree staff members a, b, and c. The problems that will be moved is colored light grey.*

in the department, since we do not want the staff members to be overloaded with problems.

To balance the workload, each staff members workload must be calculated. The workload of a staff member is defined by the amount of time estimated that each problem on his workload takes to be solved. The workload is calculated by the *GetWorkload* method.

The time a problem takes to be solved is estimated by the average time consumption of the tags connected to the problem. This is calculated by the *CalculateTimeConsumption* method.

The *BalanceWorkload* method works by finding the staff member in the department with the minimum workload and the staff member with the maximum workload. Then it moves the problem with the lowest estimated time consumption from the maximum staff member to the minimum. It keeps reassigning problems until the minimum staff member has a higher or equal workload than the maximum staff member. If the minimum staff member has a higher workload, then the algorithm checks if the balance can be balanced even further by calculating if the last moved problem should be moved back or not, and moves it accordingly. See lines 25 to 42 in code snippet 12.2.

All this is iterated once per staff member minus one in the department. E.g. if there are two staff members it is ran once, if there is three it yields two etc. The pseudo code is displayed in code snippet 12.2.

The primary concern of the algorithm is to distribute the problems so each staff member has as balanced workload as possible.

An example of the algorithm is shown on figure 12.1.

The key points presented in this chapter are problem searching and workload balancing. The problem search describes how our system is able to search and sort problems. The balance workload method is used to distribute problems among staff members.

```
bool couldStillMove = true;
do
{
    // Finde the reassignable problem with the highest priority which has not been moved yet.
    var problemToBeMoved = maxWorklist.FirstOrDefault(y =>
        y.Reassignable == true &&
        y.HasBeen == false &&
        y.SolvedAtTime == null);

    // If none can be moved leave the while loop
    if (problemToBeMoved == null)
    {
        couldStillMove = false;
    }
    else
    {
        // Mark as has been moved
        problemToBeMoved.HasBeen = true;

        // Reassign the highest priority problem to staff member called min.
        problemToBeMoved.AssignedTo = min;

        if (min.Workload >= max.Workload)
        {
            // Initialize variables for checking whether or not to move the last problem back
            double beforeMoveBack = 0.0;
            double afterMoveBack = 0.0;

            // Calculate difference before moving
            beforeMoveBack = Math.Abs(max.Workload - min.Workload);

            // Move it back
            problemToBeMoved.AssignedTo = max;

            // Calculate difference after moving
            afterMoveBack = Math.Abs(max.Workload - min.Workload);

            // Compare
            if (beforeMoveBack < afterMoveBack)
            {
                problemToBeMoved.AssignedTo = min;
            }
            couldStillMove = false;
        }
        else if (min.Workload == max.Workload)
        {
            // Don't move back if they are equal
            couldStillMove = false;
        }
    }
} while (couldStillMove);
```

Code snippet 12.2: *A code snippet of the balance workload method. The presented code is within a for loop running for each staff member minus one. "min" and "max" are the person objects of which the algorithm are currently moving problems between.* ***maxWorklistist*** *a sorted worklist. It is sorted in non-deacreasing order according to the estimated time consumption.*

Part IV

Testing

13 Black box testing

This chapter outlines the black box testing we did during and after the development of our system, as well as an example. We have utilized testing as correctness of the key methods are crucial to our system.

Throughout the development, we have used Visual Studios built in Team Test, which is an integrated unit-testing framework. [11] The idea behind unit testing is to check an individual method by executing it with appropriate input, and afterwards check that its output corresponds to the expected.

All major functions have been tested with Team Test. This mean that we have not made any unit testing on controllers. We considered controller unit testing as inefficient because it is easier to test controllers by running the program and see that they behave as expected. If any controller has very complex code a method will be made instead and this will be unit tested. This is done with for example the search and the problem distribution methods. Controller unit testing requires a great deal of independency injection and mocking since most controllers uses a database entity which we have to mock in order to make proper testing.

A unit test consist of three parts, an arrange phase, an act phase, and a assert phase. The arrange phase sets up the input and requirements for the test. The act is the run of the method and the assert is the actual testing.

In section 13.1 a black box test of the balance workload method is presented and in section 13.2 a unit test with independency injection is explained.

13.1 Balance workload unit test

The department class has a method called balance workload, which balances the workload between all staff members in the department. The method is described fully in section 12.3. To arrange the test a department object is initialized and the required properties is set. This means the **persons** property is set to a list of **persons** and each person is assigned a number of problem with tags. The arrange can be seen in code snippet 13.1.

Now the expected result needs to be calculated. Mike has three problems and this gives him a total workload of $20/1 + 20/2 + 20/2 = 40$. John has 1 problem which gives him a total workload of $20/2 = 10$. This implies that Mike's

```
var tag1 =  new Tag(){ TimeConsumed = 20, SolvedProblems = 1 ,
    Priority = 1  };
var tag2 =  new Tag(){ TimeConsumed = 10, SolvedProblems = 1 ,
    Priority = 2  };
var tag3 =  new Tag(){ TimeConsumed = 10, SolvedProblems = 1 ,
    Priority = 3  };
var tag4 =  new Tag(){ TimeConsumed = 10, SolvedProblems = 1 ,
    Priority = 4  };

var prob1 = new Problem() { Tags = new EntityCollection<Tag> { tag1
     }, Reassignable = true };
var prob2 = new Problem() { Tags = new EntityCollection<Tag> { tag2
     }, Reassignable = true };
var prob3 = new Problem() { Tags = new EntityCollection<Tag> { tag3
     }, Reassignable = true };
var prob4 = new Problem() { Tags = new EntityCollection<Tag> { tag4
     }, Reassignable = true };

var mike = new Person() { Name="mike", Worklist = new
    EntityCollection<Problem>() { prob1, prob2, prob3 } }; //
    Workload = 40
var john = new Person() { Name= "John", Worklist = new
    EntityCollection<Problem>() { prob4 } };               // = 10

Department target = new Department()
{
    Persons = new EntityCollection<Person>()
    {
        mike, john
    }
};
```

Code snippet 13.1: *The arrange phase of the unit test of balance workload*

workload is overbalanced. The algorithm is expected to reassign the problems so the workload is balanced. The most balanced possible with the given problems is when one has 30(10+20) workload and the other has 20(10+10). This can be expressed with a boolean expression which can be tested with the assert method *IsTrue*. This assert can be seen on code snippet 13.2.

It is necessary to test various scenarios e.g. test cases where problems are solved, not reassignable, problems with extreme estimated time consumption, staff members with no problems, an empty department, various estimate time consumptions, and different priorities.

13.2 Independency injection

For some methods independency injection must be made in order to execute a proper test. In the example with *BalanceWorkload*, independency injection could have been made on the workload property of **person**. This was not made because the workload method were already tested at the time *BalanceWorkload* were tested.

To properly make a independency injection the method in action should be programmed to an interface. This makes it easy to swap the implemented class with another class. We did this to test the method *GetStaff* in the **Prob-**

```
.
.
.
target.BalanceWorkload();
Assert.IsTrue(
        (john.Worklist.Contains(john.Workload == 30 && mike.
            Workload == 20) ||Â
        (mike.Workload == 30 && john.Workload == 20)
    ));
.
.
.
```

Code snippet 13.2: *An example unit test which tests a specific instance of the balanceWorkload method.*

lemDistributer class. The method is described in section **??**.

GetStaff were designed to be more testable and therefore it acts on the **IPerson** interface. The method depends on the **Workload** property of **person** and at the time of the unit test for this method the **workload** were not fully implemented and neither were the **isStaff** property. Therefore we made a **TestPerson** class which implemented the **IPerson** interface. The **TestStaff** implementation of the *isStaff* method returns true unless the **name** is "john". The *workload* property just counts the number of items in the **person** class's **worklist**.

In this way it is possible to make a unit test work.

This chapter outlines the testing we did during and after the development of our system, as well as an example.

Part V

Epilogue

14 Improvements

This chapter outlines and discusses possible future enhancements, which we for various reasons did not implement in our system.

14.1 BalanceWorkload

The original *BalanceWorkload* method distributes the problems by looking at the staff member which has the highest workload as well as the one with the lowest, and then shifts the reassignable problems until the workload between the two are as equal as possible. This approach doesn't take the priority or whether or not the deadline is approved into account.

The main idea behind the *FutureImplementationBalanceWorkload* method – which is seen in code snippet 14.1 – is that the algorithm only has to balance the workload by some extension, which is by always choosing the person with the minimum workload, whenever a problem is about to be distributed. The argument behind this idea is that whenever a staff member solves a problem, then the algorithm will be executed again, which then autimatically will assign him/her new problems.

This simple approach of distributing problems allows us to choose which problems to distribute first. From here, all we have to do is to sort the list of problems to distribute in the manner we want. In the *FutureImplementation-BalanceWorkload* method, we've chosen to have the problems with an approved deadline at the top of a temporary list, sorted by highest priority at top. Below that, all other problems – problems without an approved deadline.

In short, the *FutureImplementationBalanceWorkload* method does the above explained by first unassigning all the reassignable problems from all members of the specific department, followed by sorting them in the fashion explained above, and at last continually assigns the problem from the top of the temporary list, until the temporary list is empty.

14.2 Staff Calendar

A nice feature to add is a personal calendar for the staff members, where his working hours is included and the deadline of the problems assigned to him is

```
public void FutureImplementationBalanceWorkload()
{
    Person dummyPerson = new Person();
    List<Person> staffMembers = Persons.ToList();
    List<Problem> problemList = new List<Problem>();

    foreach (var member in staffMembers)
    {
        foreach (var problem in member.Worklist)
        {
            problemList.Add(problem);
        }
    }

    problemList = problemList.Where(x => x.Reassignable == true &&
        x.SolvedAtTime == null).ToList();

    foreach (var problem in problemList)
    {
        problem.AssignedTo = dummyPerson;
    }

    while (problemList.Count > 0)
    {
        // find the person with the lowest workload
        Person min = Persons.FirstOrDefault(y => y.GetWorkload() ==
             Persons.Min(x => x.GetWorkload()));

        // assign the most important problem to the person
        problemList[0].AssignedTo = min;
        problemList.RemoveAt(0);
    }
}
```

Code snippet 14.1: *A possible future implementation of the BalanceWorkload algorithm.*

presented.

This will allow for the staff to easily see if any deadline is approaching. It will also enable the system to more efficiently distribute problems, since it will take into account when a staff member is off duty. The calculation of expected time of completion of solving a problem can be done with higher accuracy when the staff members working hours is taking into account.

14.3 Agile roles

If the system should be implemented at different work environments the role system should be more flexible. It is not given that all workplaces can make use of the three predefined roles. In fact it is not given at all. Therefore a more flexible role system would be more efficient and make the system more agile. As it is now each controller has a predefined role which is allowed to access the controllers functionality. This method is not very agile and it is impossible for the users to change which roles can access what. This can be improved by introducing a double layered role system. This work by making a role for each controller or even the controllers action methods. Then adding a high level role set which can access the low level roles. This allow for agile changing the

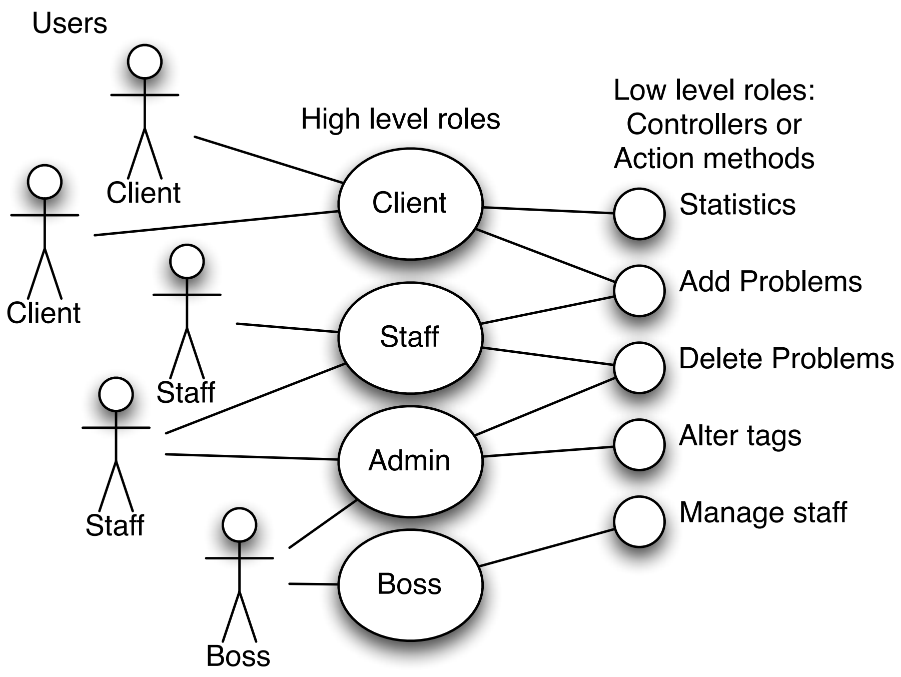


Figure 14.1: Example of how the roles can be structured with an improved role system

privileges for an entire group of users. E.g. if the user of the system wants to allow clients to see statistics this can easily be changed with this improvement. An example of this alternative role system can be seen on figure 14.1.

14.4 Text based search

If a client has a problem which cannot be described sufficently by the tags provided by the admin. His search for a similar problem can be improved by adding text based search. This feature can be extended by using it as statistics. If a word is often searched for, a tag with that word might be needed. The admin will have a view allowing him for seeing often searched keywords and allowing him for creating tags with these keywords. This feature will, ease the admin's process of creating the categories and tags, and improve the categorization process of adding a new problem and finding similar problems.

This chapter outlines and discusses possible future enhancements.

15 Conclusion

top

---

conclusion goes here

---

tail

16 Perspective

top

---

perspective goes here

---

tail

Part VI

Appendix

Bibliography

[1] Collabnet. Ankhsvn. WWW, 2010. URL `http://ankhsvn.open.collab.net/`. Last viewed: 6/12.

[2] Microsoft Corporation. Team foundation. WWW, 2005. URL `http://msdn.microsoft.com/en-us/library/ms181232(VS.80).aspx`. Last viewed: 6/12.

[3] Microsoft Corporation. Design goals for ado.net. WWW, 2010. URL `http://msdn.microsoft.com/en-us/library/7b13c12s(v=VS.71).aspx`. Last viewed: 6/12.

[4] Microsoft Corporation. Ado.net entity data model designer. WWW, 2010. URL `http://msdn.microsoft.com/en-us/library/cc716685.aspx`. Last viewed: 6/12.

[5] Microsoft Corporation. The ado.net entity framework overview. WWW, 2010. URL `http://msdn.microsoft.com/en-us/library/aa697427(VS.80).aspx`. Last viewed: 6/12.

[6] Microsoft Corporation. Visual studio ultimate. WWW, 2010. URL `http://www.microsoft.com/visualstudio/en-us/products/2010-editions/ultimate`. Last viewed: 6/12.

[7] Firefox. Install or update firefox: different platforms. Website.

[8] Google. Install or update google chrome: System requirements. WWW, 2010. URL `http://www.google.com/support/chrome/bin/answer.py?answer=95411`. Last viewed: 17/10.

[9] Leo Hsu and Regina Obe. Cross compare of sql server, mysql, and postgresql. WWW, May 2008. URL `http://www.postgresonline.com/journal/index.php?/archives/51-Cross-Compare-of-SQL-Server,-MySQL,-and-PostgreSQL.html`. Last viewed: 19/11.

[10] Lars Mathiassen, Andreas Munk-Madsen, Peter Axel Nielsen, and Jan Stage. *Object Oriented Analysis & Design*. The Johns Hopkins University Press, 1 edition, 2000. ISBN: 87-7751-150-6.

[11] Mark Michaelis. A unit testing walkthrough with visual studio team test, March 2005. URL `http://msdn.microsoft.com/en-us/library/ms379625(VS.80).aspx`.

[12] Mono-project. Mono, 2010. URL `http://www.mono-project.com/Main_Page`. Last viewed: 7/12.

[13] Trygve M. H. Reenskaug. Mvc xerox parc 1978-79. www. URL `http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html`. Last viewed: December 2010.

[14] Winston W. Royce. Managing the development of large software systems.

[15] Abraham Silberschatz, Henry F. Korth, and S Sudershan. *Database System Concepts*. McGraw-Hill, internation edition 2011 edition, 2011. ISBN: 978-007-128959-7.