

Rettelser

FiXme: Læs venligst dette igennem, og slet denne fixme, hvis du synes dette stykke tekst er fint.	3
FiXme: skriv dette ind i introen	4
FiXme: hvorfor staar der noget om platform under Knowledge saved?	4
FiXme: Indskriv flere features	4
FiXme: er mvc 2 et framework i asp.net frameworket?	5
FiXme: skal vi aendre users, employees og supervisors = clients, staff og admin ?	5
FiXme: skriv hvad statistics skal gore	5
FiXme: slåevents sammen, tilføj de nye events, flere classes se klass diagram, fjern urelevante, kald den general, kontroller at checkmarks er sat rigtigt (department)	10
FiXme: fix saa det stemmer med functions listen, naar den er blevet opdateret	17
FiXme: make it right, when the functions are updated.	18
FiXme: staff must at least have the same functions as client	18
FiXme: admin must at least have the same objects as client	18
FiXme: admin must at least have the same functions as client	19
FiXme: admin must at least have the same functions as client	19
FiXme: Hvorfor er "create problem" markeret som en compute-funktion?	20
FiXme: Nogen der kan huske om vi stadig kigger på tid siden det blev committed?	22
FiXme: dette skal uddybes	22
FiXme: Flere kilder	38
FiXme: Indsæt Microsoft SQL Server versionsnummer	40
FiXme: Ret dette såsnart vi ved hvad vi bruger	40
FiXme: eller hvad?	43
FiXme: Dette er en reference til der hvor beskrivelsen af HelpdeskWindow kommer til at være. Indsæt gerne :)	47
FiXme: Det her kan nok gøres mere udførligt, men ved ikke om det er meningen	48
FiXme: Nødvendigt?	48

FiXme: Skal tag også skrives på eller ligger det implicit	49
FiXme: Lidt kryptisk måske..	49
FiXme: Hvad er det for en klasse?	51
FiXme: Hvad er det for en klasse?	52
FiXme: Hvad er det for en klasse?	52
FiXme: Hvad er det for en klasse?	52
FiXme: Link til MODELLEN sættes ind her.	62
FiXme: Tilføj dette eksempel eller fjern denne linje	69

Title:

Hopla Helpdesk

Theme:

Programming

Project period:

SW3, fall semester 2010

Project group:

S305A

Participants:

Alex Bondo Andersen

Kim Jacobsen

Magnus Stubmann

Kristian Kolding Foged-Ladefoged

Lasse Rørbæk Nielsen

Rasmus Veiergang Prentow

Synopsis:



Advisor:

Nadeem

Page count: 78

Appendices count: 0

Finished: 17/12-2010

The content of this report is open to everyone, but publishing (with citations) is only allowed after agreed upon by the writers.

Contents

I	Analysis	1
1	Task	3
1.1	Introduction	3
1.2	System Definition	4
1.3	Context	6
1.3.1	Problem Domain	6
1.3.2	Application Domain	7
2	Problem Domain	9
2.1	Classes and Events	9
2.1.1	Classes	9
2.2	Structure	10
2.3	Behavioral Pattern	11
2.3.1	Problem	11
2.3.2	Person	12
3	Application Domain	15
3.1	Usage	15
3.1.1	Actors	15
3.1.2	Use Case	16
3.2	Function	19
3.3	User Interfaces	22
3.3.1	Client Interface	22
	Main	22
	Search	22
	Client problem view	23
	Categorize New Problem	23
	Search for similar problems	23
	Create new problem	24
3.3.2	Staff Interface	26
	Main	26

II

	Worklist	26
	Staff problem view	26
	Search for existing solutions	27
	Staff Problem View	27
	Add Solution	27
3.3.3	Admin Interface	29
	Main	29
	Person administration	29
	Department administration	29
	Person view	29
	Category view	29
	Tag view	29

II Design 33

4	Task	35
4.1	Purpose	35
4.2	Quality Goals	35
4.2.1	Prioritizing Quality Goals	35
5	Technical Platform	39
5.1	Equipment	39
5.2	Software	39
5.2.1	DBMS	39
5.2.2	Web Server	40
5.2.3	Operating System	40
5.3	System Interfaces	40
5.4	Design Language	41
5.4.1	Coding Standard	41
6	Architecture	43
6.1	Component Architecture	43
6.1.1	System Component	43
6.1.2	Client-Server	45
7	Components	47
7.1	Structure	47
7.1.1	User Interface Component	47
	Client Interface	47
	Staff Interface	48
	Admin Interface	48
7.1.2	System Interface Component	48
7.1.3	Function Component	48
	Problem Handler	49
	Administrator	49
	Authenticator	49
7.1.4	Model Component	49
	E-R Diagram	50
7.2	Classes	51

III	Implementation	55
8	Development	57
8.1	Development Tools	57
8.1.1	IDE	57
8.1.2	Collaboration	57
8.1.3	Database	57
8.2	Development Method	58
9	Program Presentation	59
9.1	Client Usage	59
9.2	Staff Usage	59
9.3	Admin Usage	59
10	Enviroment	61
10.1	MVC Framework	61
10.1.1	Structure	61
10.2	Database Structure	62
10.2.1	ADO.NET Entity Framework	63
11	Key Points	67
11.1	Problem Search	67
11.1.1	Search for Problems by Tags	68
11.1.2	No Tags to Remove	69
11.1.3	Order by Solved	70
11.2	Balance Workload	70
11.2.1	eta	71
IV	Testing	73
V	Appendix	75
	Bibliography	77

Part I

Analysis

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.

1.1 Introduction

¹ We live in a world where we distribute the problems at hand to the people who are best at solving them. Sometimes this is a easy process, and other times, it can be an immense waste of time trying to find or contact someone who we can ask. Often, we end up solving problems ourselves, which someone else could have solved in, let us say one fifth of the time we used, at a lower cost. Not only did we waste our time, but we could have spent the time working with other things which we are particularly good at, keeping our work efficiency at a maximum.

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 individual 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

¹ **FiXme:** Læs venligst dette igennem, og slet denne firme, hvis du synes dette stykke tekst er fint.

the clients spent time to a minimum.

Productivity goes up, wasted time goes down.

Comprehensive knowledge of the OOA&D method is assumed.²

1.2 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:

- 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 will focus on the ease of use for all users, saving and suggesting solutions to problems before a similar problem is submitted. Furthermore the system should be able to run on all systems without any prior software installation.³
- Statistics
Allow the supervisor to monitor the work process of the employees.⁴

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.

²**FiXme:** *skriv dette ind i introen*

³**FiXme:** *hvorfor staar der noget om platform under Knowledge saved?*

⁴**FiXme:** *Indskriv flere features*

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 framework⁵. 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 staff to communicate about a problem and the following solution.

6

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 Estimated time of completion, comments and what employee is responsible for the problem. This will enable us to create detailed statistic about problem solving efficiency and individual employees.⁷
- 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.

⁵**FiXme:** *er mvc 2 et framework i asp.net frameworket?*

⁶**FiXme:** *skal vi aendre users, employees og supervisors = clients, staff og admin ?*

⁷**FiXme:** *skriv hvad statistics skal gore*

1.3 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.

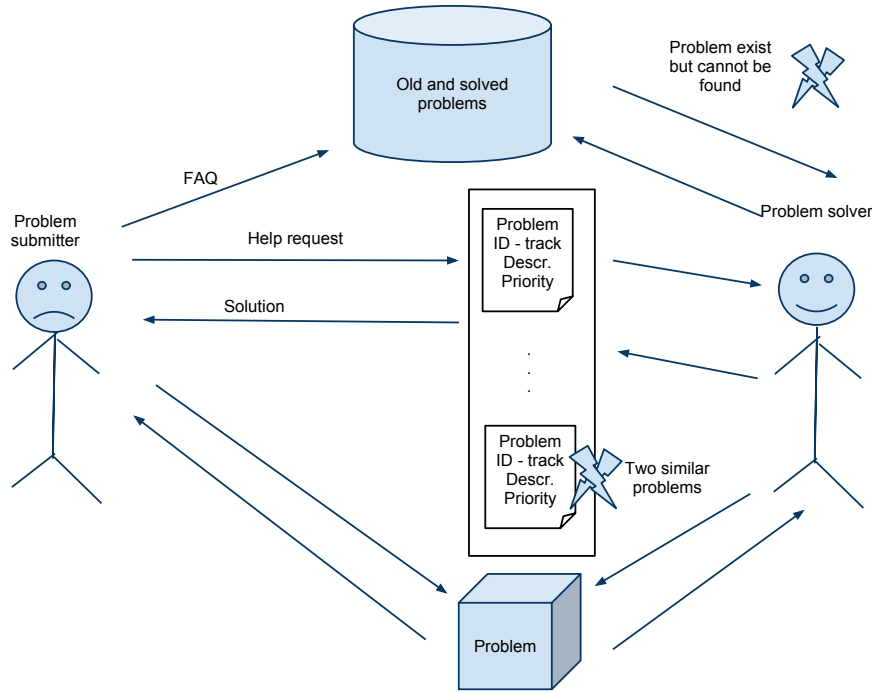


Figure 1.1: *Rich picture of our system*

To fully understand the context we draw a rich picture which can be seen on figure 1.1. 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.

1.3.1 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 2 gives a detailed analysis of the problem domain and will elaborate on the phenomena.

1.3.2 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 3, 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.

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.

2.1 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 2.2 the relations between classes and events are illustrated.

2.1.1 Classes

The following is a list of the classes from the class diagram 2.1.

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.

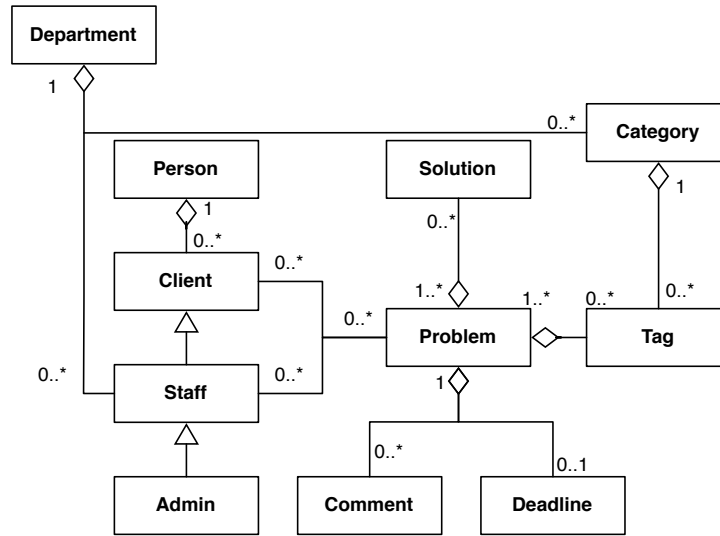


Figure 2.1: Class diagram

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

Tag₁ Tags is used to determine the problems category and thereby department.

2.2 Structure

The class diagram is based on the classes and events, which was described earlier in chapter 2.1. 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

¹**FiXme:** slåevents sammen, tilfæg de nye events, flere classes se klass diagram, fjern urelevante, kald den general, kontroller at checkmarks er sat rigtigt (department)

<i>Events</i>	<i>Classes</i>				
	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 2.2: *Problem-domain analysis event table*

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**.
- **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.

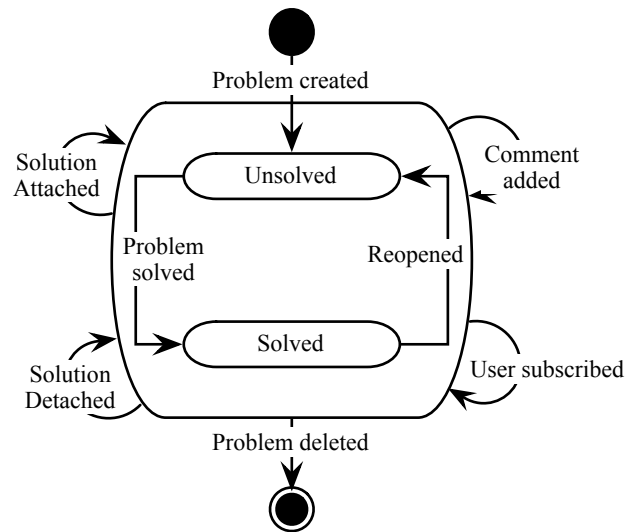
A problem consist of none or many *comments*, none or many *solutions* and, one or many *tags*. A **tag** belongs to a category and a **category** belongs to a **department**. The class diagram is illustrated at figure 2.1

2.3 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.

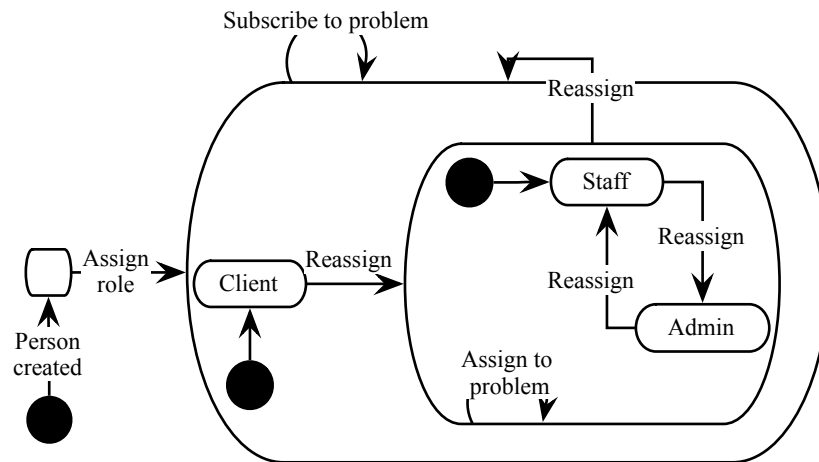
2.3.1 Problem

Figure 2.3 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 2.3: *The statechart of the problem class*

2.3.2 Person

As shown in figure 2.4, 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 2.4: *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.

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.

3.1 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.

<i>Use case</i>	<i>Actor</i>		
	Client	Staff	Admin
Submit problem	✓	✓	✓
My problems	✓	✓	✓
Worklist		✓	✓
Solve problem		✓	✓
Administrate			✓
Statistics			✓

Figure 3.1: *Actor & use case table*

Figure 3.1 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 3.1 are described in subsection 3.1.2.

3.1.1 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 lightbulb in the IT-administrators office is broken and the maintenance officer should fix it.

These two actors are described in details in figure 3.2 and 3.3. 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 3.4.

<i>Client</i>
<p>Goal: A person who has a problem, and his goal is to get his problem(s) solved.</p> <p>Characteristics: The clients are employees or students with different knowledge and experience with similar systems.</p> <p>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.</p>

Figure 3.2: *Description of the actor client.*

<i>Staff</i>
<p>Goal: The staff solves the clients problems and use the system as a taskmanager.</p> <p>Characteristics: The staff are employees and has various levels of technical knowledge.</p> <p>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,</p>

Figure 3.3: *Description of the actor staff.*

3.1.2 Use Case

The use cases from figure 3.1 are described below.

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

Figure 3.4: *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 3.5.

- **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 than 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, category, client, staff.
- **Functions:** Search existing problems, compare problems, create problem, subscribe to problem. ¹

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 3.6.

- **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

¹**FiXme:** *fix saa det stemmer med functions listen, naar den er blevet opdateret*

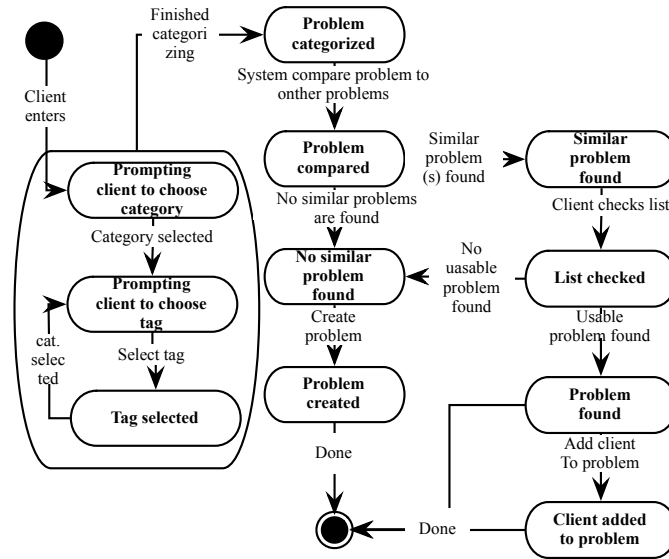


Figure 3.5: A state chart diagram of the use case submit problem.

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, solution, comment, client, and staff. fixmestaff must at least have the same objects as client
- **Functions:** Add comment, reassign, change status, search database, create solution, attach solution and get staff worklist.^{2 3}

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 3.7.

- **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:** Staff, Client, category, tag, department.⁴

²FiXme: make it right, when the functions are updated.

³FiXme: staff must at least have the same functions as client

⁴FiXme: admin must at least have the same objects as client

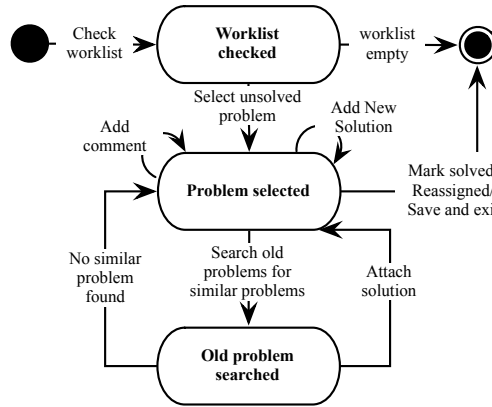


Figure 3.6: A state chart diagram of the use case solve problem.

- **Functions:** Delete person, Add person, create department, set permissions, delete department, set priority, add category, delete category, set tag visibility, set category visibility, create tag, delete tag. ^{5 6}

3.2 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]

Add Comment This function simply adds a comment to an existing problem, making the comment visible for other client and staff members. This function is marked as an *signal*-function because a notification is sent to the assigned staff member.

Reassign Problem The reassign problem function reassigns a specific problem from one assigned staff member to another. It is marked as a *signal*-function as the new assigned staff member received a notification of the newly reassigned problem.

Change Problem Status This function changes the status of a given problem. It is marked as a *signal*-function since the associated clients receives a notification.

Delete Problem To delete a problem, this function is used.

⁵**Fixme:** admin must at least have the same functions as client

⁶**Fixme:** admin must at least have the same functions as client

Get Staff Todo List This function simply fetches the staff members todo list. Hence marked as a *read*-function.

Create Tag creates a tag, which is an element used for categorizing problems, which purpose is to make searching easier, also attaches the tag to a category.

Create Category To create a category, this function is used. It also attaches that category to a department.

Delete Tag If a tag is not attached to any problem it can be deleted, if not it can only be hidden.

Delete Category If any tag is attached to the attempted deleted category it can not be deleted and can only be hidden.

Change Visibility of Category The Change Visibility of Category function changes the visibility of a category. Changes happen and a need of a category may vanish over time, but might be needed again in the future, thus making this function necessary.

Change Visibility of Tag This function hides tag if the admin no longer finds it useful. In cases a delete can not be done, because the *tag* is related to some problems.

Create Department All staff members are associated with a department in order to ease the process of distributing newly created problems to appropriate staff members.

Delete Department As the name dictates, this function simply deletes a department.

Add Person A person can either be a client, staff, or an admin. See figure 3.1 in section 3.1.1. This function creates a person with characteristics to define him/her in the system.

Rename Person This is a trivial function, which simply changes the name of a person.

Remove Person This function removes a person from the system.

Get Statistics This function returns statistics, e.g. average time for specific kinds of problems to be solved, hence “complex”-complexity. It is marked as a *compute*-function since it involves a fair amount of computation on the data in the model.

Subscribe Client To tie a client and a problem together through a subscription, this function is used. It is marked as a *signal*-function since the assigned staff member receives a notification.

Set Priority This function sets the priority of a tag.

Get Priority This functions computes the priority of a problem, based on the priority value of the tags attached to the problem and time since it were committed.⁸

Set role This is a trivial function, which grants a specific persons role.

3.3 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.

3.3.1 Client Interface

The clientinterface is illustrated in figure 3.9. After login, the client is presented with the *main* window. All the window have a back function which enables the client to return to the previous window.

Main

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

- The “Add 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 posting 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
- 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.

Settings⁹ solved problems my problems

⁸**FiXme:** *Nogen der kan huske om vi stadig kigger på tid siden det blev committed?*

⁹**FiXme:** *dette skal uddybes*

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

- A list of categories, when a category is selected the tags connected to it is shown.
- 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. The creator of the problem is not able to unsubscribe.
- A text field with all the previous entered comments if any.
- A field to write new comments in.
- The button “Add 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 “Add problem” which send 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:

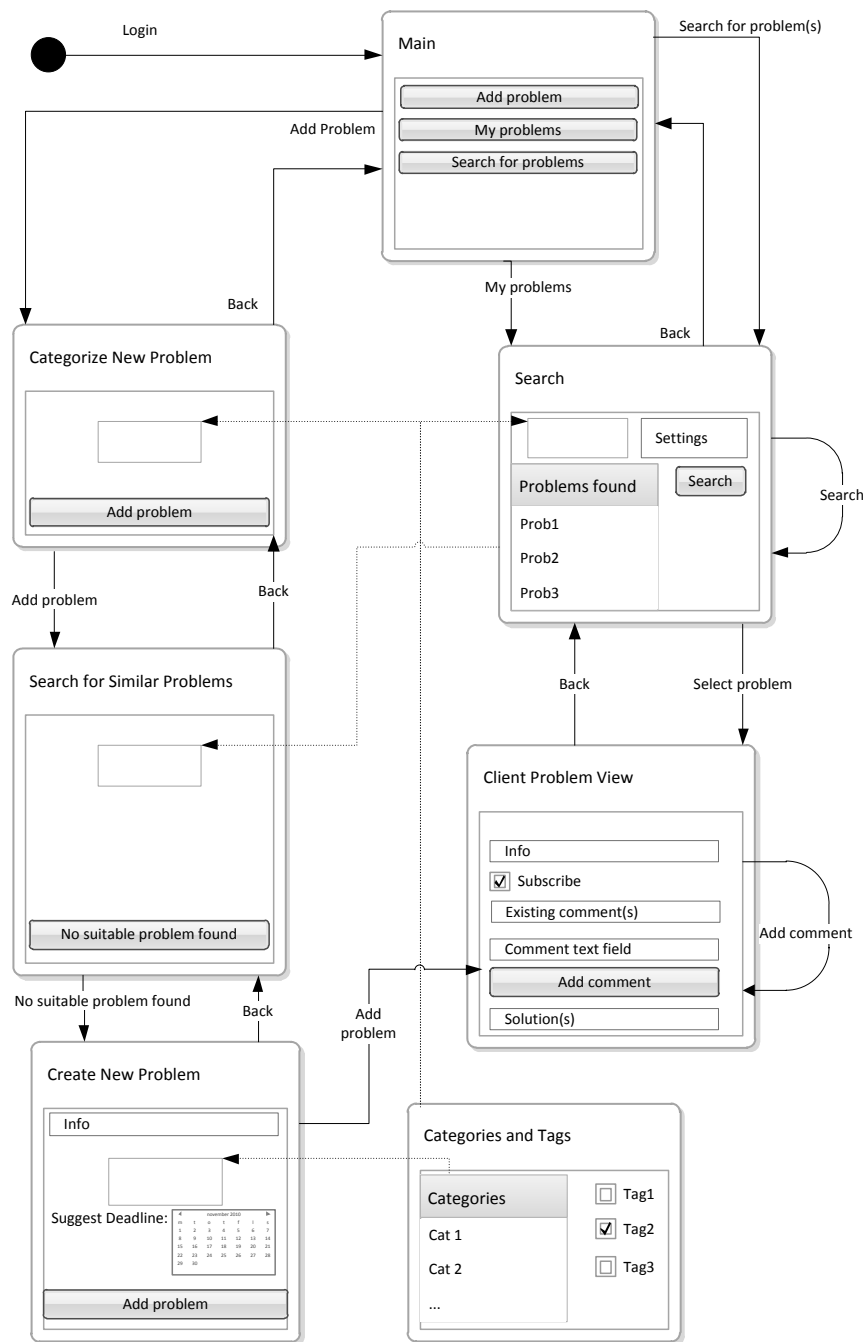
- The window “Search” with all it elements.
- A button called “No suitable problem found” which send loads the window “Create new problem”.

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 client to change and add tags so the problem is best describe.
- A calender where a deadline can be suggested to the staffmember.
- “Add problem” submits the problem and send the user to the “Client problem view”.

Figure 3.9: *Client Interface*

3.3.2 Staff Interface

The staff interface is illustrated in figure 3.10. After login, the staff is presented with the *main* window. All the window have a back function which enables the staff to return to the previous window.

Main

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

- “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 ETA. When a problem is double 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 checkbox where the staffmember can choose to subscribe to 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 “Add comment” to send the entered comment.
- A text field containing none or more solutions.
- The button “Add new Solution” which send the staff to “Search for existing solutions”.
- The “Reassign problem” 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 “Delete” to remove the problem from the system.

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 3.3.1 is reused, and it enables the user to search for problems.
- The button “Write new solution” which sends the staff to the “Add solution” window.

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

Staff Problem View

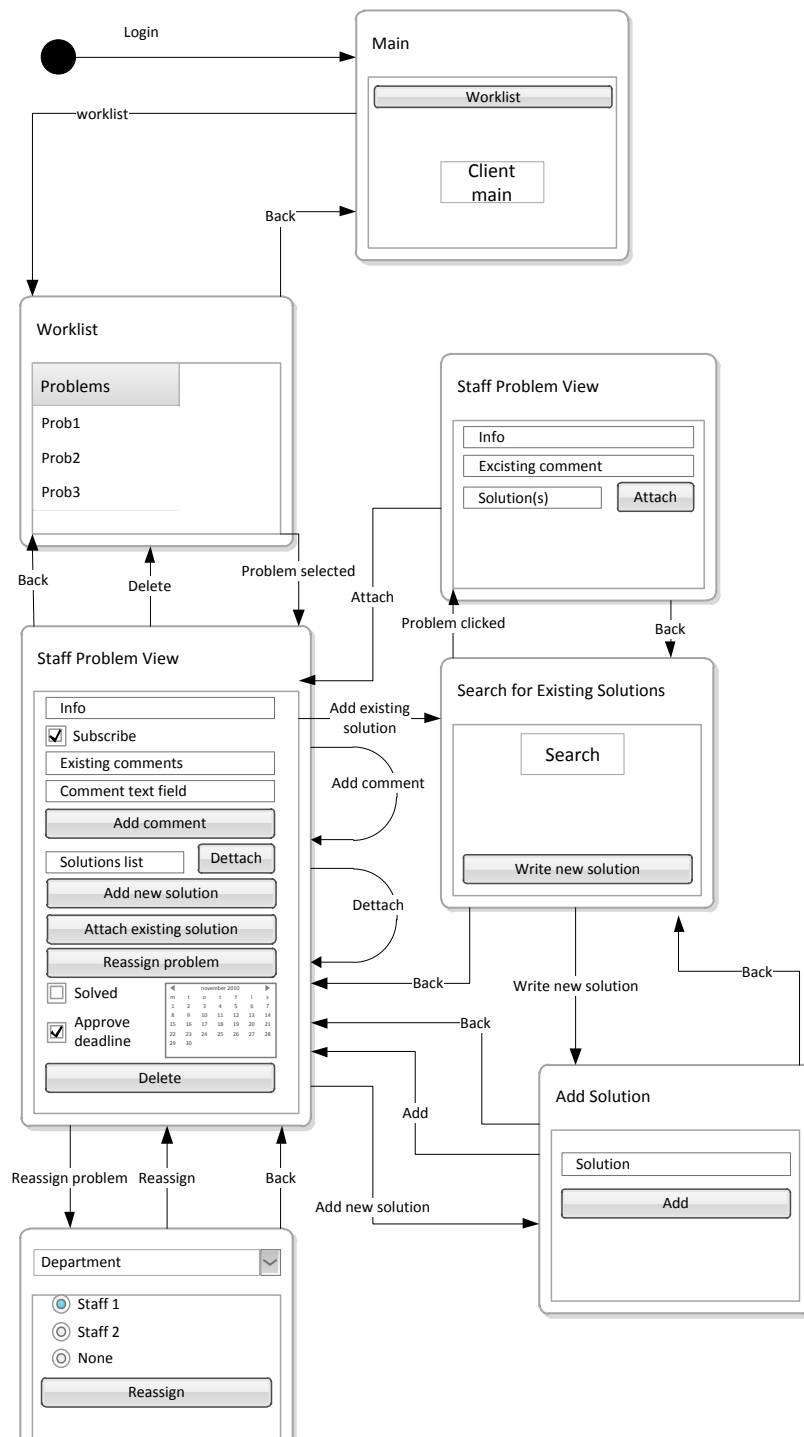
The “Staff Problem View” show properties of the selected problem. This window contains the following:

- 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 staff is send to the “staff Problem View” window.

Figure 3.10: *Staff Interface*

3.3.3 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 “Administrate departments, categories, and tags” which directs the admin to the “Department Administration” window.
- The button “Administrate persons” directs the admin to the “Person Administration” 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 three elements:

- A dropdown menu where departments or “Clients” can be selected.
- A list with all the staffmembers or clients which belongs to the selected group.
- The “New person”

Department administration

This window

Person view

Category view

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

-
- save
- new tag

Tag view

- save

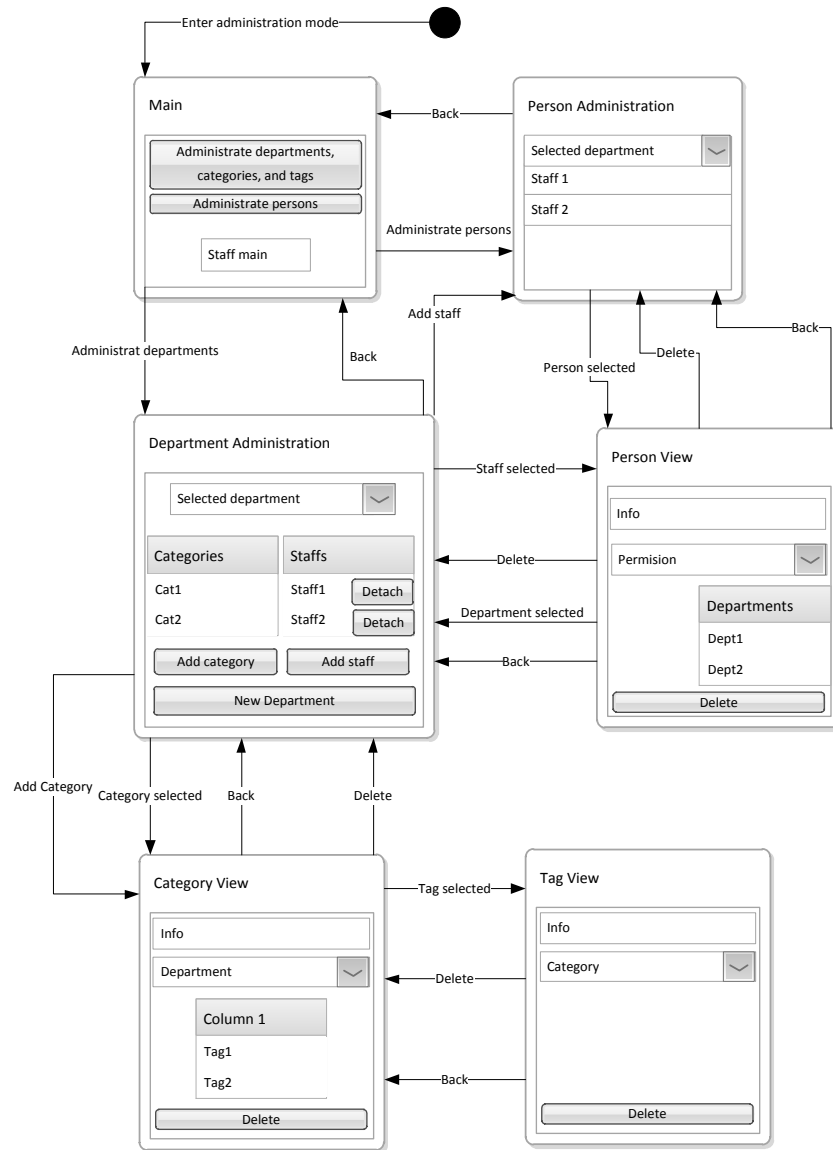


Figure 3.11: Admin Interface

beskrv kontekst af siden/viewed beskriv navigations muligheder! skriv at der skal være et navigations panel. beskriver hvordan man navigere mellem de forskellige vinduer

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.

Get all categories	Simple	Read
Get problem tags	Simple	Read
Search for problems	Complex	Read/Calculate
Get / set problem Deadline	Simple	Read/Update
Get problem matching tags	Medium	Read/Calculate
Get / set problem ETC	Medium	Read/Update/Calculate
Get / set problem Title	Simple	Read/Update
Get / set problem Description	Simple	Read/Update
Create problem	Medium	Update
Get / set problem status	Simple	Read/Update
Get / set problem added date	Simple	Read/Update
Assign / unassign staff to / from problem	Simple	Read/Update/Signal
Get problem assigned staff	Simple	Read
Get / set problem solutions	Simple	Read/Update/Signal
Get problem comments	Simple	Read
Add comment to problem	Simple	Update/Signal
Subscribe / unsubscribe to problem	Simple	Update
Get worklist	Simple	Read
Get / set problem priority	Simple	Read/Update
Approve deadline	Simple	Update
Attach / detach solution to / from problem	Simple	Read/Update
Get list of departments	Simple	Read
Create department	Simple	Update
Create tag	Simple	Update
Hide tag	Simple	Update
Hide category	Simple	Update
Get department staffs	Simple	Read
Get department categories	Simple	Read
Create category	Simple	Update
Get / set person email	Simple	Read/Update
Get / set person department	Simple	Read/Update
Get / set person role	Simple	Read/Update
Delete person	Simple	Update
Get person name	Simple	Read
Get person workload	Simple	Read
Reset person password	Medium	Update/Signal
Balance workload	Complex	Calculate
Get statistics	Medium	Read/Calculate

Figure 3.8: *Function list*

Part II

Design

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.

4.1 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, the ETA, the priority, and the deadline of the problem. 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 problems in categories and tags.

4.2 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 table 4.1. [10, p. 178]

4.2.1 Prioritizing 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 4.1 is prioritized in figure 4.2. Further the figure 4.2 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 1.2. The reasoning for the priority of each criteria in figure 4.2 is shown bellow.

<i>Criterion</i>	<i>Definition</i>
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	The cost for the end user to modify the system after deployment
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 4.1: *Definition of criteria*

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

Figure 4.2: *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 make sure whether it is a client or a staff member who is using the system – 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 be generic – that it can be adapted to any organization without much or any cost. 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]¹

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.

¹**FiXme:** *Flere kilder*

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 1.2 should hold true, and the quality goals which are prioritized in section 4.2.

5.1 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.

5.2 Software

Our system relies on two things:

- Database
- Web server

These two are described in following subsections.

5.2.1 DBMS

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

The differences which we have considered between these two DBMS's are listed in figure 5.1[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 5.1: *DBMS's compared*

Our system requires a DBMS supporting the query language SQL as well as the relational database storage model. We have chosen Microsoft SQL Server¹ 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.

5.2.2 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. For the web server we will be using Microsoft Internet Information Services (IIS). The language of choice will be C# using the ASP.NET MVC(Model-View-Controller) framework.

We could have used other web-frameworks, but chose ASP.NET MVC 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 might as well 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. Also we might as well use it to save time on the creation of trivial things such as how a view is shown.

5.2.3 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 5.4 for more. We are using Microsoft Windows server 2008 R2 Standard. ²

5.3 System Interfaces

The only system interface which our system has is an interface to the SQL server, Microsoft SQL Server. We are using ADO.NET which is part of the .NET-framework, to map our relations to objects.[2] ADO.NET is integrated in Visual Studio 2010, which we are using as IDE(Integrated Development Environment).

¹**FiXme:** *Indsæt Microsoft SQL Server versionsnummer*

²**FiXme:** *Ret dette såsnart vi ved hvad vi bruger*

An alternative would be to make our own component responsible for ORM(Object Relational Mapping). This would however take time from our actual project so we will rather use something which is already working and focus on the central part of our project.

5.4 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(Model-View-Controller) 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.

5.4.1 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 5.1 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 5.1: *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.

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.

6.1 Component Architecture

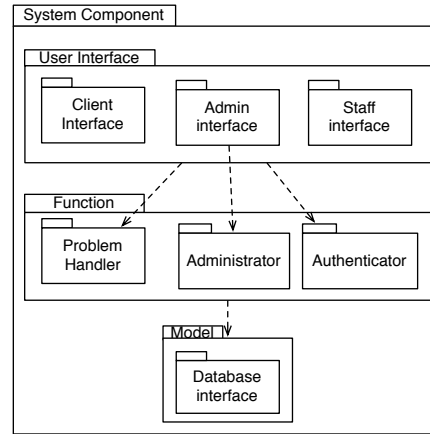
With basis in our system definition in chapter 1.2 and our evaluation of criteria in section 4.2 we found that our main priority is flexibility. To obtain high flexibility we created a closed-strict layers component architecture, we did this to avoid complex dependencies between components and therefore making it easy to change an interface, a function component, or ¹. The architecture has three layers: User- and System-interfaces, Functions, Model as shown on figure 6.1. The figures components will be explained in section 6.1.1.

6.1.1 System Component

The systems three actors are displayed as interfaces. The clients and staffs functions are located in the Problem Handler component, while the admins additional functions are located in the Administrator component. The client only have limited access to the Problem Handler component while the staff and admin have full access to the Problem Handler's functions, in addition the admin uses the administrator component. To determine if a user is a client, staff or admin the component Authenticator is used. The Authenticator asks the Database Interface what the permission the user have. Based on what permissions the user have access are given to the different interfaces. The Problem Handler and the Administrator components sends info to the Database Interface who translate the info and sends it to the database. All the components are described below:

Client Interface The client interface get inputs from the client, through this interface the client submits problems, post comments and search for existing problems. The functions associated with this interface are located in the Problem Handler component.

¹FiXme: eller hvad?

Figure 6.1: *Our system component*

Staff Interface Through this interface the staff members can access all the client functions, submit solutions, post comments to unsolved problems, view worklist and administrate problems. This Interface uses the Problem Handler component.

Admin Interface The interface has access to all the client functions, the staff functions, tags/category management, department management and administration of the client/staff. This is due to the fact that if a person is an admin, then he/she also is a staff and a client. The Admin Interface uses the Problems Handler and the Administrator component.

Problem Handler The problem handler component controls the handling of new problems, the handling of solutions, the searching function, recording and sending statistics, manages comments and sending notifications by using the notifier component. This component is also responsible for sending out signals to relevant actors. e.g. if a user posts a comment the relevant staff member(s) receives a notification.

Administrator Through the admin interface the admin component can be accessed. The functions affiliated with this component are control of departments, management of tags/categories and adding/removing of staffs and clients.

Authenticator To determine which rights a person have, they have to authenticate them selves as either client or staff/admin, this occurs through an authenticator component which communicates with the database interface.

Database Interface The database interface receives all database requests and translate them to the correct database language.

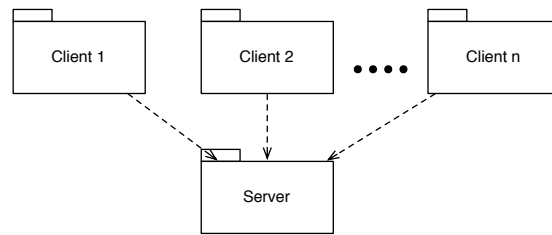


Figure 6.2:

6.1.2 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 6.2. 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.

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.

7.1 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.

7.1.1 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 3.9. Each window represents a class and the navigation arrows indicates an association between the

¹**FiXme:** 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 sub-component. 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 3.11 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.

7.1.2 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 our system is being implemented. There is no internal structure in this component since it only has a single class which holds the responsibility for 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.

7.1.3 Function Component

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

²**FiXme:** *Det her kan nok gÅ, res mere udfÅ,rligt, men ved ikke om det er meningen*

³**FiXme:** *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 7.1.4 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.

7.1.4 Model Component

The classes of the model component are described in section 2.1. However during revision of our class diagram, we decided to insert more classes. Figure 7.1 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:** *Skal tag også skrives på eller ligger det implicit*

⁵**FiXme:** *Lidt kryptisk måske..*

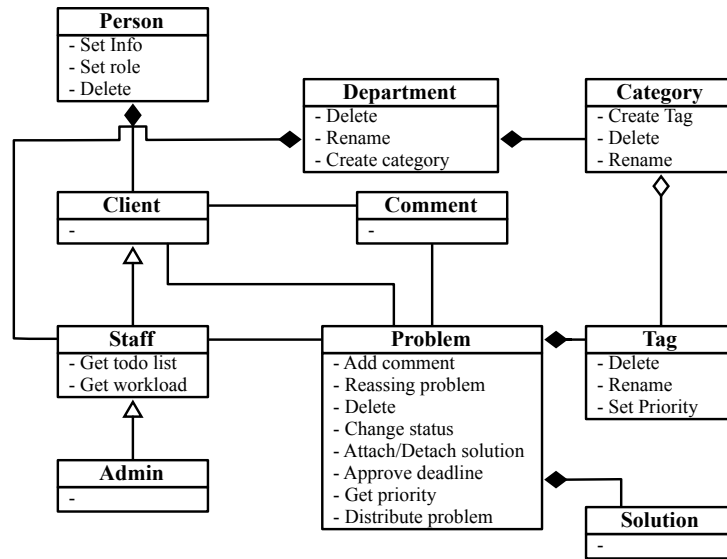


Figure 7.1: The revised model component. Notice that the attributes of each class is omitted, these are however shown in the E-R diagram in figure 7.2

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 7.2. The notation for the E-R diagram is based on the citation used in the book Database System Concepts [11, p. 305]. The E-R diagram is based upon the class diagram in figure 7.1.

Every class is turned into a entity and every relation is turned into a relationship [11, 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

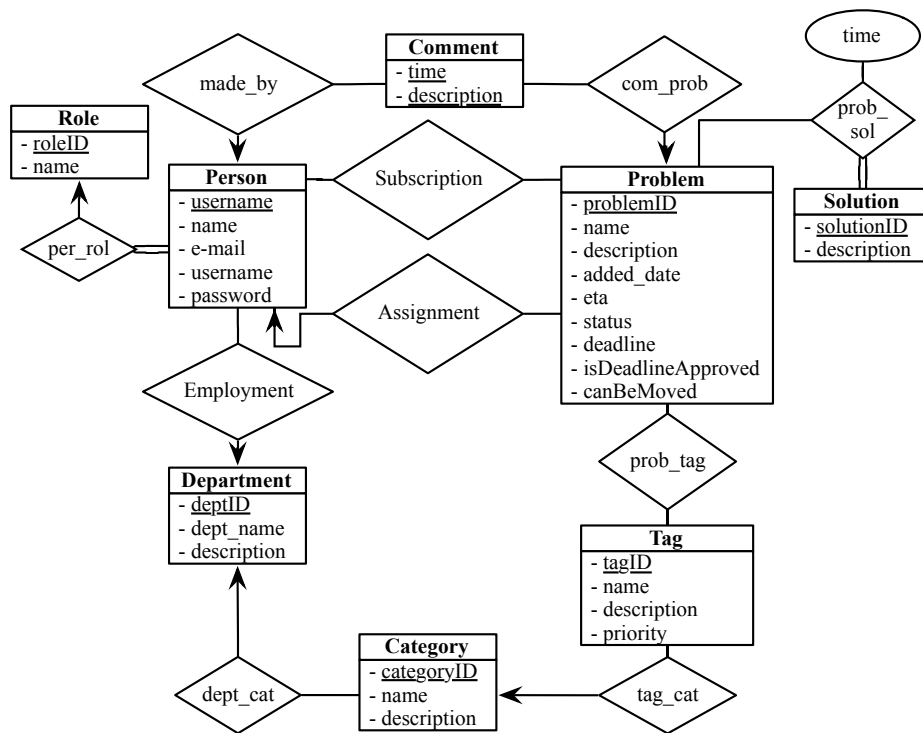


Figure 7.2: The E-R diagram of our model component

also gives a nice modifiability because adding a new role is simply adding a new tuple to the entity.

7.2 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.

Person: This class purpose is to register new users in the helpdesk. When a person register, he/she need to provide the Person class with following attributes:

- name
- mail
- username
- password

Login: ⁶After a user is registered in the helpdesk, he/she can login using their username and password they choose when registering. The login class purpose

⁶**FiXme:** Hvad er det for en klasse?

it to make sure all users is registered before they can submit problems and add comments. The login class need the following attributes: ⁷

- username
- password

HelpdeskWindow: ⁸All windows in the helpdesk inherits from this class. The HelpdeskWindow class is an abstract class, which purpose is to provide a standard layout for all windows. This class use the following attributes: ⁹

- width
- height
- position
- er der mere?

Problem: When a user submit a new problem to the helpdesk, he will be using the Problem class. The Problem class responsibility is to make sure the user provides the required attributes:

- title
- text

Client: This class is the default role, that every user got after registering. The Client class purpose is to provide the user with basic rights, such as submitting a problem and adding a comment. This class doesn't require any attributes.

Staff: This is a employee role, you can only have this role if your a employee of the helpdesk staff. The Staff class give the employee rights to change problem status, assign problems, delete problems/comments, edit problems/comments, attach solution. This class doesn't require any attributes.

Solution: When a client submit a problem, it's the staff's job to find and attach a solution. The Solution class purpose is to let the staff create and attach a solution. This class requires the following attributes:

- file
- text

Assignment: This class purpose is to assign problems to staff members, and let staff members assign problems to other staff members. This class doesn't require

⁷**FiXme:** *Hvad er det for en klasse?*

⁸**FiXme:** *Hvad er det for en klasse?*

⁹**FiXme:** *Hvad er det for en klasse?*

Subscription: Clients can subscribe to a problem, each time the problem is updated with either a comment, solution or status change, the client will be informed by mail.

Comment: This class give the clients the rights to comment on a problem. The class require the following attributes.

- Text

Department: When a problem is submitted, it is assigned to a department based on tags the client provided. The purpose of this class, is to make sure the problems is assigned to the right staff. The Department class require the following attributes:

- name

Category: A department contain one more more categories that partly describe the department. The Category class purpose is to make sure, problems is assigned to the right department. The class require the following attributes:

- Name
- Description

Tag: Before a client can submit a problem, he/she needs to add predefined tags to the problem. Each tag belongs to a category, more tags can have the same category, but one tag can't have more categories. This class purpose is to add tags to a category or a problem. This Tag class require the following attributes:

- Name
- Description

Admin: This is the highest role that a staff member can have. As admin got all rights, the class doesn't require any attributes.

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

Top

8.1 Development Tools

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

8.1.1 IDE

The primary development tool has been Microsoft Visual Studio Ultimate [7], 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 Mono and none has experience with this IDE either. As most group members work at a daily basis on Windows Based pc's and C# is a Microsoft product, Visual Studio is the favorite choice of IDE.

8.1.2 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 [6], but this requires installation and configuration of a Team Foundation Server. We chose AnkhSVN since we already had a running SVN server.

8.1.3 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 editor. Which allows for editing the SQL server from our workstations and not only from the server itself. We considering using PostgreSQL, but using postgresSQL with C# and Visual Studio required a plugin inorder to use the ADO.NET Entity Data Model Designer. As a database vendor either choice

would not give us any advantages on the data layer, since the functionality we require is supported by both systems.

8.2 Development Method

Tail

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 3.

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.

We will start by presentating the Clients usage.

9.1 Client Usage

9.2 Staff Usage

9.3 Admin Usage

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

10

Enviroment


hvad og hvorfor

10.1 MVC Framework


10.1.1 Structure

The Model-View-Controller (MVC) framework consists of three different types of components; Views, controllers and models. Each created with a different purpose:


- Models


The models are the components that handle the data domain of the application. Forexample, the model might be mapped to a database, from which it returns data and writes changes 


- Views

The views display the UI and is typically created with data from the model. These views are bascially ordinary (X)HTML pages with the addition of C# or Basic code, to fetch content from the model. 

- Controllers

The Controllers reacts to user interaction, and select which view should be rendered. They can also send data with the  request, in order to display it in the controller.


Because of the seperation into three independent modules, developers are able to create different parts of the application with different kind  logic, thus enabling them to better manage complexity. It also enables developers to create applications with a high maintainability, because it is easy to alter any of them without effecting the others.

Displaying actual content  Another interesting thing about MVC is that there are no actual HTML files. When entering an URL your actually calling methods in controllers, you can even add parameters if you want. These controllers then redirects you to the correct view. This is possible because everything is displayed inside of a Master page, which is basically a HTML document, with the addition of content containers. These containers can be created anywhere on the Master page, and act as spaces where views can put their HTML output, the entire page is then created as HTML output, and sent to the browser which requested the page via the URL.

Data validation


Testing

10.2 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 10.2.1. 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 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  When we needed to access data from the membership tables we had to use SQL statements and not the object oriented approach we used for the rest. This is limited since our main functionality depends on the model.

¹**Fixme:** *Link til MODELLEN sættes ind her.*

10.2.1 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].

The ADO.NET EF is an object relational mapping framework [5].

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 11.2.1

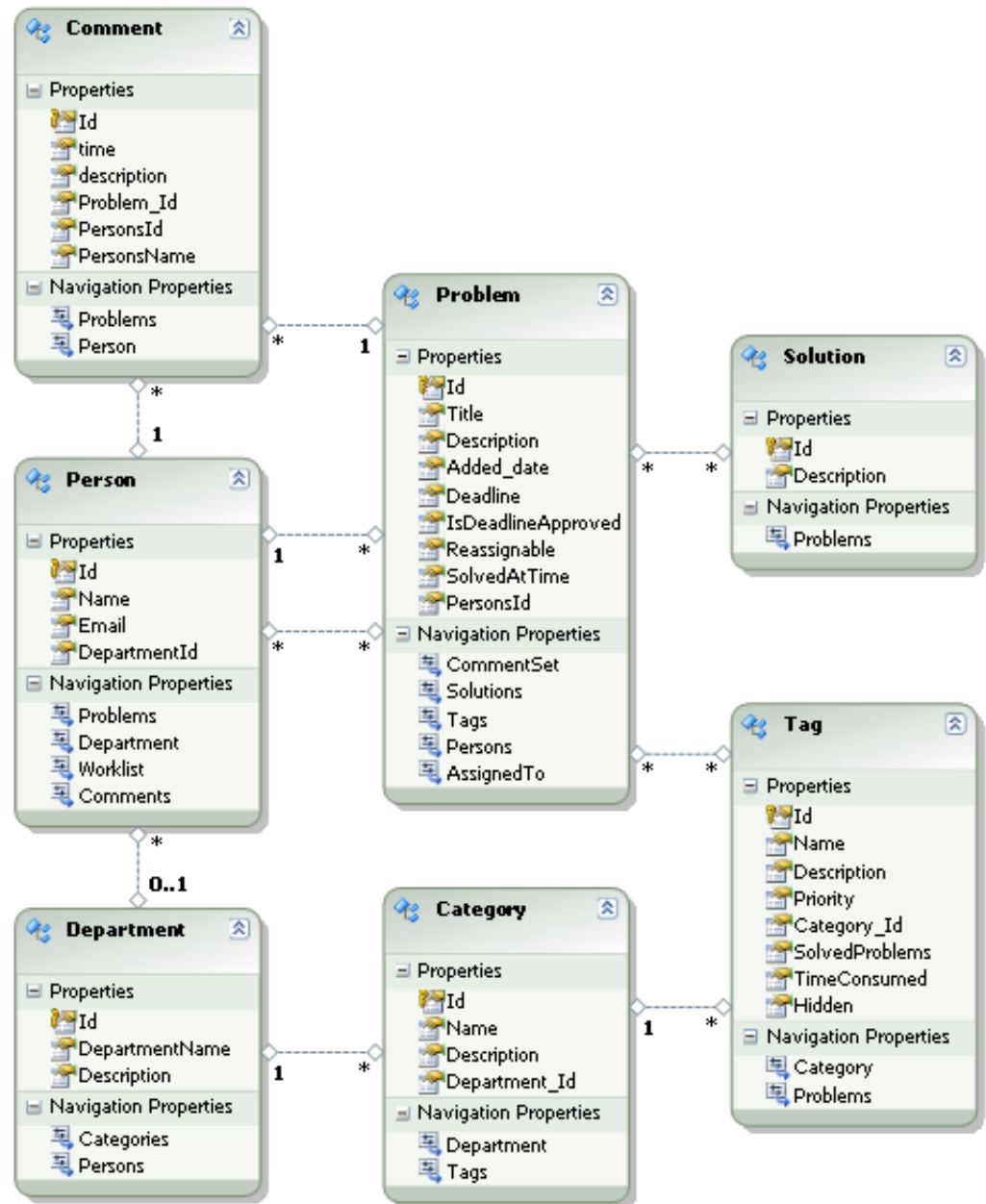


Figure 10.1: Our model as it is seen in the ADO.NET Entity Data Model Designer.

Hvad

11

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 1.2. 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.

11.1 Problem Search

The system needs to be able to search for problems. This search is based on tags. It will 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 then calls an *InternalSearch* function which is private, with the same parameters and a compare function which determines how the problems should be sorted. The *InternalSearch* function is described in subsection 11.1.1 and 11.1.2. Subsection 11.1.3 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 function.

11.1.1 Search for Problems by Tags

```

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 11.1: *The while loop which finds and sorts problems matching the input tags*

The most important part of the *InternalSearch* function is the while loop shown in code snippet 11.1. 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 11.1.2.

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 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 ??¹.

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 searing 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.

11.1.2 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

¹**Fixme:** Tilføj dette eksempel eller fjern denne linje

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 than 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.

11.1.3 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.

11.2 Balance Workload

Whenever a staff is removed from a department or has marked a problem as solved, it becomes necessary to balance the workload of each staff member in the department, since we do not want the staff members to be overloaded with problems.

To balance the workload, each staffs workload must be calculated. The workload of a staff 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 in the department with the minimum workload and the staff with the maximum workload. Then it moves the highest priority problems from the maximum staff to the minimum. It keeps reassigning problems until the minimum staff has a higher or equal workload than the maximum staff. If it is higher the problem is reassigned back. All this is iterated once per staff minus one in the department. E.g. if there are two staffs it is ran once, if there is three it yields two etc. The pseudo code is displayed in code snippet 11.2.

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

We also wanted the algorithm to take the individual priority of the problems into account. This algorithm makes sure that the high priority problems will be distributed. If the priority was irrelevant the algorithm would take the problem with lowest time consume, since this would give the most balanced workload. An example of the algorithm is shown on figure 11.1.

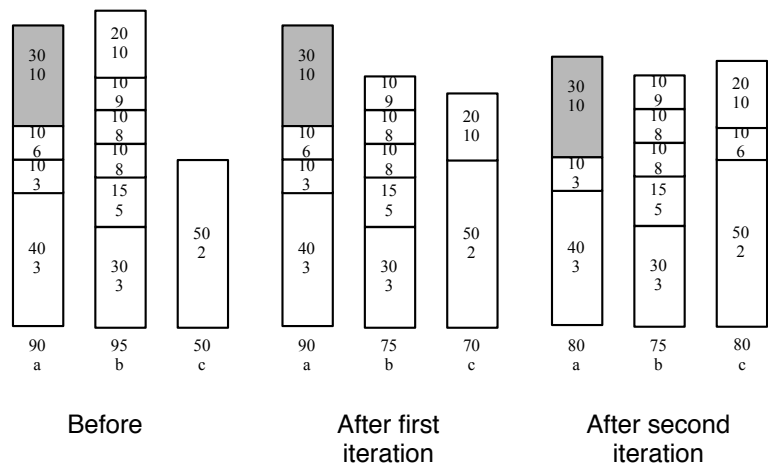


Figure 11.1: A diagram of the balance workload method. Each column represents a staffs workload. Each box is a problem. The upper number is the estimated time consumed of that problem. The lower is the priority. The problem colored dark grey is not reassignable. There are three staffs a, b, and c. The problems that will be moved is colored light grey.

11.2.1 eta

Get workload kalder ETA

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.


```
var max = Persons.FirstOrDefault(y =>
    y.GetWorkload() == Persons.Max(x =>
        x.GetWorkload()));

var min = Persons.FirstOrDefault(y =>
    y.GetWorkload() == Persons.Min(x =>
        x.GetWorkload()));

// sorts the list after priority
max.Worklist.ToList().Sort(Problem.GetComparer());
while(true)
{
    var problemToBeMoved = max.Worklist.FirstOrDefault(y =>
        y.Reassignable == true &&
        y.HasBeen == false &&
        y.SolvedAtTime == null);

    if (problemToBeMoved == null) break;
    problemToBeMoved.HasBeen = true;
    problemToBeMoved.AssignedTo = min;

    if (min.Workload > max.Workload)
    {
        problemToBeMoved.AssignedTo = max;
        break;
    }
    else if (min.Workload == max.Workload)
    {
        break;
    }
}
```

Code snippet 11.2: A code snippet of the balance workload method. The presented code is within a for loop running each for every staff minus one

Part IV

Testing

Part V

Appendix

Bibliography

- [1] Collabnet. Ankhsvn. WWW, 2010. URL <http://ankhsvn.open.collab.net/>. Last viewed: 6/12.
- [2] Microsoft Corporation. Overview of ado.net. WWW, 2010. URL [http://msdn.microsoft.com/en-us/library/h43ks021\(VS.71\).aspx](http://msdn.microsoft.com/en-us/library/h43ks021(VS.71).aspx). Last viewed: 19/11.
- [3] Microsoft Corporation. Design goals for ado.net. WWW, 2010. URL [http://msdn.microsoft.com/en-us/library/7b13c12s\(v=VS.71\).aspx](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](http://msdn.microsoft.com/en-us/library/aa697427(VS.80).aspx). Last viewed: 6/12.
- [6] Microsoft Corporation. Team foundation. WWW, 2005. URL [http://msdn.microsoft.com/en-us/library/ms181232\(VS.80\).aspx](http://msdn.microsoft.com/en-us/library/ms181232(VS.80).aspx). Last viewed: 6/12.
- [7] Microsoft Corporation. Visual studio ultimate. WWW, 2010. URL <http://www.microsoft.com/visualstudio/en-us/products/2010-editions/ultimate>. Last viewed: 6/12.
- [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] Abraham Silberschatz, Henry F. Korth, and S Sudershan. *Database System Concepts*. McGraw-Hill, international edition 2011 edition, 2011. ISBN: 978-007-128959-7.