NTNU
Innovation and Creativity

Scenario testing in a real environment

Key card Administration System at the University Hospital in North Norway

Yngve Halmø
Geir-Arne Jenssen

Master of Science in Computer Science
Submission date: June 2006
Supervisor: Tor Stålhane, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

The purpose of the project is to conduct research in software engineering, assessing the usefulness of the scenario testing method through a software test with its real users. The results will be used for further implementation in the tested software system.

Assignment given: 16. January 2006
Supervisor: Tor Stålhane, IDI

Scenario testing in a real environment

Key card Administration System
at the University Hospital in North Norway

TDT4900 Computer Science, Master's thesis

Yngve Halmø
halmo@stud.ntnu.no

Geir-Arne Jenssen
geirarj@stud.ntnu.no

June 12, 2006
Supervisor: Tor Stålhane

NORWEGIAN UNIVERSITY OF TECHNOLOGY AND SCIENCE,
DEPARTMENT OF COMPUTER AND INFORMATION SCIENCE

Abstract

Software is gradually replacing paper based administration systems. The migration to electronic systems is supposed to make life easier for the users. If this is to be the case then these software systems must be created in such a way that the end users are able to use them effectively. To achieve usable systems, software testing must be utilized. There are many ways to test a program, with or without involving real users. Scenario testing is a somewhat poorly documented discipline in software testing, with ambiguous definitions. It does however seem to be well suited in combination with users to test external parts of a software system in a late state of development.

This project is based on the work done in the software engineering depth study [12]. There we conducted empirical work and internal testing of the software system KAS, and laid the foundation for this Master's thesis. In this report we have continued the work with this software and concentrated on its external characteristics and user testing. We have analyzed scenario testing further through a software test of this system involving its future users.

The users have been given tasks to complete through stories that explain what to do but not how to do it. We have observed the test subjects closely throughout the tests, and collected important data. The results have been evaluated in order to assess their usefulness, which further points to the quality of scenario testing as a testing method. The results have also spawned functional requirements which have been implemented into the KAS. Through this project we have gained experience that can be useful to others conducting scenario tests or doing research in software testing in the future.

Preface

This report is our Master's thesis which is the final part of our Master's degree at the Norwegian University of Science and Technology in Trondheim (NTNU). This work lasted from January to June 2006 and was based on the work done in the depth study [12] in the previous term. The work has primarily been done at NTNU, in addition to a trip to the University hospital in Tromsø (UNN) to conduct user testing on a previously developed software system.

We would like to thank our supervisor Tor Stålhane for his guidance throughout the project. We also would like to thank our contacts at UNN, Thomas Lyngmo and Jonny Svendsen, for giving our work priority, and the IT department at UNN for their assistance. We finally want to thank the employees at UNN who participated in our scenario test for their valuable feedback.

Yngve Halmø *Geir-Arne Jenssen*

Contents

List of Figures

List of Tables

Part I

Background

Chapter 1

Introduction

> *Just remember: you're not a "dummy," no matter what those computer books claim. The real dummies are the people who, though technically expert, couldn't design hardware and software that's usable by normal consumers if their lives depended upon it.*
> Walter Mossberg

This chapter contains an introduction to this Master's thesis. It starts with our motivation, followed by an elaboration of the problem at hand. The last section of this chapter gives an outline of the contents of the report.

1.1 Motivation

The foundation for this thesis was laid in the in-depth study project in the previous term [12]. Both the in-depth study project and this Master's thesis are based on a software system developed by us as a student project in the year 2004. This software system, which is described in more detail later in this report (chapter 2), is a key card administration system for the University Hospital in North Norway (UNN)[1].

The in-depth study project during the previous term [12] was an empirical study of the software system (which from now on will be referred to as KAS[2]) where we conducted interviews and questionnaires for the purpose of analyzing different definitions and importance of quality attributes. Understanding the users' definitions of these attributes would make it easier to deliver a system that lives up to their expectations. Internal testing[3] based on the results of these empirical studies was performed to confirm that the system lived up to the requested level of quality. We did this to ensure that the KAS was as usable as possible before the external testing, and eliminate errors that would have had to be fixed anyway.

The general feedback received from the previous study [12] was utterly motivating as people welcomed the new electronic system with comments like "Do it as fast as possible!", "Very nice" and "Good luck to you guys!". Positive user feedback has increased our incentive to make the system as good as possible. It has also been a motivation factor that

[1] Universitetssykehuset i Nord-Norge, http://www.unn.no/

[2] Key card Administration System

[3] The ISO-9126 standard [1] defines the capabilities of a software system as internal attributes that can be measured through internal metrics. This is how we define *internal testing*. Furthermore, the standard states that software quality characteristics can be measured externally by the extent to which the capability is provided by the system containing the software. This is how we define *external testing*.

UNN believes in us and wants to use our system if we can complete it.

We have been excited about receiving response from an external test phase, to see if the users actually are satisfied with the system we have developed. We have also been anxious to see if the system provides the required quality, functionality and intuitiveness in order to be useful in the day-to-day work of the users. If it does satisfy all UNN's needs and is taken into use, it should save UNN a lot of time, money and paperwork, and give us the satisfaction of having created something others find useful. We have therefore continued the development of this system, while doing some research on user testing that could provide more insight into the users' goals and opinions, until the system is operational.

1.2 Project context

This Master's thesis was carried out at the Department of Computer and Information Science at the Norwegian University of Science and Technology in Trondheim. The work in this Master's thesis lasted from January to June 2006. Our teaching supervisor was professor Tor Stålhane.

This project was also carried out in cooperation with UNN, which is to be the owner of the system when it is completed. Our contact with UNN has primarily been through our contacts Thomas Lyngmo and Jonny Svendsen, who will later be the administrators of the KAS. We have previously worked with them on other projects, including the initial student project in Tromsø. We have also worked with the joint IT-department of UNN and UiT[4], which will later host the system on their servers.

1.3 Problem definition

When a software system has been implemented according to the functional requirements specification, work still remains before it is ready to be used in its intended environment. The end-users should evaluate it to make sure it is usable, and it should be tested in its real technical environment. Most software systems are, however, shipped without any user testing at all because of the short-term costs involved and time-to-marked concerns.

To make a software system usable one should, according to our experience and sources like [1] and [16], perform both internal and external testing. Internal testing should uncover technical errors while external testing, preferably involving users, should uncover practical problems and high-level errors. If the system works from a technical point of view, and it does what the users need it to do, it should intuitively be usable. We want to analyze the user testing process further on existing software, namely the KAS.

The main goal of this project is to achieve the highest possible quality of our software and getting it operational. To achieve this, the perspectives of all the stakeholders[5] should be considered and extensive user testing should be done.

Picking up the thread from the previous project's internal testing efforts [12], we want to

[4] University of Tromsø

[5] A stakeholder is any person with an interest in the software system, be it a user, investor, corporate head or developer

concentrate on the external aspects of the system through user testing. User tests require more planning and more resources than testing that does not involve users (i.e. tests which are done by the developers or dedicated testers), because of the practical issues that must be considered. Examples include people needing to take a break from normal work to participate in the tests, the tests must be well planned to make good use of the allotted time, people may not show up as agreed, and so forth. Scenario testing is a testing method that seems to lack good documentation, but the method is a viable option for external testing with users involved. We want to study user testing closely, and see how well scenario testing suits our software system given its current stage in the development cycle and the system's characteristics[6]. We want to find out if this testing method, given its high initial costs and resource requirements, still is cost efficient to the results it can provide. Through a qualitative approach we hope to spawn many experiences that can be useful to others conducting scenario testing or doing research in software testing in the future.

1.4 Report outline

This section outlines the rest of the report. The report is structured so that the chapters should be read in chronological order, since some chapters provides background for the subsequent chapters.

Chapter 2, *About the KAS*, is a brief introduction of the system we have developed. Here the history of the system is described, along with an overview of what the system can do. A description of the future users of the system is also provided here in order to get a better understanding of the system's requirements.

Chapter 3, *Testing*, provides an overview of software testing and different testing methods that is of relevance to this project.

Chapter 4, *Scenario testing*, describes scenario testing as a testing method in more detail. This includes a detailed description of what scenario testing is, how to conduct a scenario test, and an extension of the concept *scenario testing*.

Chapter 5, *Research*, gives a better understanding of the focus of this project and provides an overview over data collection methods.

Chapter 6, *Planning*, goes into details concerning the planning of the project and provides the ground work for the testing phases.

Chapter 7, *The scenario test*, describes the scenario test session and provides qualitative results obtained from it along with some discussion of these results.

Chapter 8, *The field test*, describes the planned field test phase.

Chapter 9, *Discussion*, contains a discussion and evaluation of all the results obtained during this project. A discussion of scenario testing as testing method can also be found here.

[6]The system and its characteristics are further described in chapter 2

Chapter 10, *Conclusion*, highlights our conclusion based on the discussion and details any further work that needs to be done with both the system and scenario testing.

Appendix, The appendix contains the scenarios created for the scenario test, the results from the scenario test, a to-do-list containing functional requirements discovered during the scenario test and a glossary with some of the most used concepts throughout this report.

Chapter 2

About the KAS

> *That's the thing about people who think they hate computers. What they really hate is lousy programmers.*
> Larry Niven

This chapter describes the software system we have developed, which has been tested during this project. The chapter begins with the background for creating this system in chapter 2.1, followed by a summary of the system's history in chapter 2.2. Chapter 2.3 contains a brief description of the system, and the technology used to create it is described in chapter 2.4. In chapter 2.5, details about the GUI of the KAS can be found. Finally, the future users of the system are introduced in chapter 2.6.

2.1 Initial project background

UNN has many employees, departments and doors with access control. An elaborate access control system is in place to control people's access to the different parts and departments of the hospital. Each door has an electronic lock that detects radio signals from approaching key cards, these signals are then used to authenticate the person trying to open the door. A central computer controls which employees have access to which door. It is important that no one gets access to a door or department they are not supposed to. This is especially important regarding medicine storage rooms. Some employees are hired on a long time contract and some are hired temporarily to work part time. All of these people must have their own key card to get access to their respective departments. Some of these cards are lost, others need updating, some must be activated immediately after being ordered while others are ordered a long time beforehand. This must be handled by the security personnel.

Every year the security personnel at the hospital handles approximately 800 orders for ID cards and 2000 orders for temporary key cards. These two types of cards are described in more detail later in this section. Today these orders are administrated and handled manually through paper forms. Every order must be filled in by hand by selected people at every department who have been given authorization to do so. The form must then be delivered to the security department, where they log the order by hand and note the day[1] it should be entered into the security computer[2] in a calendar. This system for registering

[1] The day the employee starts working, or changes department, or any other circumstance when his/her security privileges need to be updated

[2] The computer controlling the elaborate access control system described above, which is a separate system from the one we are developing

and administrating key cards is cumbersome, and it is easy to make mistakes. A department name may be written wrong, it may be interpreted wrongly because someone has bad handwriting, or the order form may never arrive at the security department at all.

The order forms go through several phases with up to four persons involved at different times before a card or card update is issued. The "control function" that checks if the cards are returned on time when they expire is relatively complicated and takes a long time to go through. All of this leads to much paperwork which leads to relatively much paper laying around to keep track of the history.

An employee has to deliver orders in person to the security personnel but there exists no mechanism for the department heads to control that an order has actually been delivered, i.e. it could be forgotten, or the orders could simply disappear among other papers. This could lead to no card being made, and if this happens, people who were supposed to start their work will not get access to the rooms and departments they were supposed to. Thus, such disappearance should be prevented, and the department heads need a better tool to trace their orders.

There exists two kinds of key cards and therefore two kinds of orders forms, one for employees working full time (ID cards) and one for people working part time (temporary cards). The temporary cards are only issued to people for a short period of time. Employees working full time get their own ID card with their picture on it. When a job is done or a persons quits his/her job, all key cards have to be delivered back to the security personnel. The security personnel must at all times know how many cards that are in use, who has these cards, when these cards should be turned in, and last but not least, be sure that only authorized personnel can order key cards to other people.

To alleviate the work load and make the administration more efficient, the technical department at UNN, contacted the Tromsø University College in order to get some students to make a computer system for them that could replace the existing paper-based system and better handle their administrative challenges.

2.2 History

As students at Tromsø University College we chose this assignment for our spring 2004 bachelor thesis. Our contacts at UNN recognized, as we did, that this system was too large and extensive for a student project, and they had therefore cunningly divided the system into parts that could be build upon at a later stage, and our initial assignment was to implement the system of ordering time-limited cards and administering these. It quickly became clear that this system would require a rather complex database as a core, and most of the initial work was concentrated around the database. But as the first, apparently simple part of the system was being implemented, the initial quality requirements specification grew drastically as our contacts discovered features they had not thought of, but had to be implemented if the system were to be usable. Several problems arose with the initial design that could not have been foreseen given the information available at the time, and large parts of the system had to be redesigned. A deadline had to be set for new features, and when the implementation of the system was finished according to the first specification, it contained a beta version of the core functionality it supports today. It could, however, not be put into use at that time as it had to be tested

for bugs and errors and to ensure that it had all the functions the users needed to use it effectively. This was outside of the scope of the initial project. At this point, the login and ordering functions of the system looked like the figures 2.1 and 2.2:

Figure 2.1: An early version of the login screen

The system was then improved upon during a summer internship at UNN. Much more functionality was added and most known bugs were fixed. The system was seen by future users for the first time when a few selected security personnel got to voice their opinions. There was, however, still not enough time to test the system thoroughly or write good documentation.

During fall 2005 we did extensive internal testing of the system [12] as a first stage of this master thesis at NTNU, and reviewed its architecture (see [5]) to see if it had to be changed to meet the identified business goals and quality requirements of our contacts. A few enhancements and changes were made and errors fixed.

Spring 2006 we have tested the external parts of the system, and done the work required to move from the implementation phase to the final, installed and working phase. This is described throughout this report. In its near final state, the same functions shown in figures 2.1 and 2.2 now looks like the figures 2.3 and 2.4.

2.3 Outline of the KAS

Technical details will not be presented in this report, as security in this system is paramount and such information is therefore restricted. Such details are in any case not important regarding the focus of this project. We will, however, give an overview of what the system is and what it can do in the following sections. For more screenshots and details about the system see the user manual for security personnel in the appendix F.

The system is basically an administration system for handling orders for key cards. It

Figure 2.2: An early version of the time limited card order screen

Figure 2.3: The login screen

Figure 2.4: The order time limited card screen

will run on the hospitals intranet and can be reached from all in-house computers. It has a common login page for all users (see fig. 2.3). If the user is authenticated he/she will be given the appropriate access to the system. There are five types of users, which will be explained in more detail in chapter 2.6. In short, all the system's users can order key cards, and the security personnel can and must additionally decide whether these orders are to be approved.

2.4 Platform/framework/technology

The KAS is implemented as a web interface for running on a webserver on the hospital's intranet. The interface is reachable with an Internet browser from any PC directly connected to the hospitals network. In such a large and complex environment with hundreds of computers, maintaining updated software on the clients would be nearly impossible, and would without a doubt be the source of many errors. The server-side scripting approach[3] makes it unnecessary to install any software on the clients to use the system, as all the code is kept on the server. The relatively small amount of estimated users on this system compared to the Internet with its millions of users, which the technology platform is able to handle, combined with the high-speed internal intranet will give the rather powerful web server no challenge at all, and the system's response time has been tested to be as fast as an installed application. The intranet is not directly connected to the internet, greatly reducing the possibilities of hacking from the outside.

The framework used to develop this system is ASP.NET [4]. ASP.NET uses the .NET Framework Class Library with all its services, and is in our opinion useful and easy to work with as long as you are writing applications solely for MS Windows servers. ASP.NET code is parsed and compiled on the web server. When a client sends a request for a web page with ASP code, the server processes the page and sends the output back to the client as an HTML[5] page. ASP.NET-files consists of both HTML and ASP-code in some programming language. In our case Visual Basic.NET was used because this was the hospital's standard programming language.

The KAS is connected to a database that stores all dynamic information and data. This database is a MS SQL database[6]. Detailed knowledge of the database design is considered a security risk, and is therefore not described here. The database is large and complex and any modification of this database would be difficult and would require extensive knowledge of the entire system.

2.5 The graphical user interface

The graphical user interface (GUI) of the KAS has been developed from scratch and differs from the normal MS Windows standard in many ways. First, it does not use drop-down menus. All choices are displayed directly where they intuitively belong, and no choices are displayed where they are not needed. What the user sees is highly context-sensitive. Second, it does not use windows and pop-ups where this can be avoided, all

[3] See for example www.php.net or www.asp.net

[4] ASP.NET is a scripting technology for web developed by Microsoft. ASP is short for Active Server Pages, see http://www.asp.net. Similar non-proprietary technologies include notably php and perl

[5] HyperText Markup Language, the standard for making web pages. Utilizes special "tags" to describe the formatting of a web page. For more info see http://www.w3.org

[6] Microsoft's implementation of the SQL database standard

information is displayed in the active window to avoid confusion. Task-switching was initially agreed upon with the stakeholders as not user-friendly for people with low computer skill. Third, the system tries to scale all information to fit the active page, to prevent that the user must use scrollbars to locate information. This reduces the number of elements on screen that the user must relate to. Finally, by utilizing CSS[7] the GUI can also be easily changed to the users preferences.

2.6 The future users

An overview of the future users of the system is presented here. There are five different user levels, all with different privileges in the system. A short description of their responsibilities and perspective on the system is provided.

2.6.1 Employees

These users can not use the system themselves and are therefore not considered users of the system. They are included here because they are still a part of it as they need to be registered in the system in order to have a key card. All employees that do not fall under any other category is a normal employee.

2.6.2 Time-limited card orderer

This is the bulk of the systems users, and they have only a few privileges in the system. Users with this level of access are able to order temporary key cards to other employees in their own department, and see status of these orders. An order can be queued, accepted or rejected by the security personnel. Time-limited card orderers are able to send remarks along with the order to the security personnel, and they are able to see answers to these messages. They also have the ability to change their own password. These users will not necessarily receive any training because someone may need to perform this function on a very short notice without any former experience with the system. This part of the system must therefore be extremely intuitive and easy to use, and leave no margin for error. This is also the least complicated part of the system.

2.6.3 Department head

One or a few users in each department has this privilege, which basically gives them the same access as the time-limited card orderers. In addition, they can grant other users the time-limited card orderer access level for their own department, for practical reasons.

2.6.4 ID card orderer

If a person is hired on a permanent basis, he or she will be eligible for a permanent ID card. These cards can be ordered for any department. It is mainly the personnel department that has this user level. An ID card has a picture of the owner and a description of his/hers position. The procedure for ordering these cards is similar to ordering time-limited cards, and a separate list of previously ordered ID cards is available to these users. They can also order changes to a permanent employees name or access rights. These users will use the system regularly which make them important resources for feedback

[7] Cascading Style Sheets, technology that separates the layout of a HTML page from its content. See http://www.w3.org/style/css for more information

to the developers during the test phases. They can provide advanced feedback as experienced users.

2.6.5 Security personnel

This user level is significantly different from the previous ones. These users will receive all submitted orders and either accept/reject them at the appropriate time, activate key cards[8] if an order is approved and deactivate key cards that have expired. They will also verify and implement changes to personalia as ordered by the personnel department. These users have, as a benefit of a relational database[9], several new options compared to the old paper based system. They have for example the ability to inspect all logs and sort them by any criteria, see which cards should be returned soon, which should have been returned already and who is in possession of ID cards. In addition, this user group has access to all functions of the lower user levels, like submitting an order, changing passwords etc. There are only a handful of these users, but they are also the most critical group for maintaining normal day-to-day operations in the hospital. Their daily routines must be supported by the system. This is a difficult task as not all of them are skilled in computer use as discovered in the previous project [12], but their daily responsibilities are still complex. This part of the system is also the most technically complex and prone to errors, mostly because it must handle a large amount of possible issues and problems and combinations of them. Most of the work has gone into designing this part and its success is probably the most important factor when UNN decides whether they will finally accept the system or not. The start page for security personnel is shown in figure 2.5.

[8] This is done on the security computer described earlier

[9] A popular type of database, usually implemented through the SQL standard.

Figure 2.5: The start page for security personnel

2.6.6 Administrators

Administrators have all access rights and can use all functions in the system. They also have some additional tools, including the ability to update several tables in the database easily through the system's GUI, i.e. delete a department, insert a new department, delete and insert access levels that is used on key cards, and decide which employees that should have access to the higher user levels. Only a few people will have administrator rights. Although they are an important part of the system, they are technically skilled and should be able to use the system without trouble. They have also had their opinions voiced to a large extent as they have essentially defined the functional requirements specification for the system in the previous projects.

Part II

Prestudy

Chapter 3

Testing

> *Considering the current sad state of our computer programs, software development is clearly still a black art, and cannot yet be called an engineering discipline.*
> Bill Clinton

In this chapter we will explore various testing methods that are commonly used in modern software engineering. For each method we will explain why we have/have not used the method. We will also look at what makes a test good.

Testing is an essential part of any software development. Every piece of software in a system must be tested to see of it does what it is supposed to do. A function must accept the correct in-data and return the right value, an algorithm must be effective enough and a database query must collect the right data. Furthermore, even if a program does what it is intended to do, errors may still exist.

Modern software testing have in many respects come a long way since people started writing software. New, advanced testing tools have been created that make testing easier. Pre-tested subroutines exist that can be used right out of the box. Development environments and compilers identify a part of the errors, but still the percentage of resources required to properly test a software system is about 50% of its total development cost, as it has been since the early 80's [16]. Increasingly powerful computers, many new operating systems and programming languages, and a plethora of possible hardware combinations have complicated the matter. Computer programs are also continuously growing in size, making the task daunting [12].

The prestudy section of the previous project [12] contains more information on software testing.

3.1 What is a good test?

Tests are "good" in different ways, for example good at identifying some types of errors and bad at finding other types. No test is good in every way [7]. Tests are used for many purposes, including everything from finding ways to work around known bugs to reducing the risk of safety-related lawsuits against a company [7]. The quality of a test is therefore highly relative to its purpose. However, if we narrow their intention down to exposing defects in the code and getting these bugs fixed we can say more about what makes a test good. According to [7], there are within this objective still many different ways a test could be considered good;

- **It is credible**, in the sense that it tests scenarios that the stakeholders agree to be realistic. Certain errors will occur more frequently than others because of the way a system normally is used, and exposing these errors first could be wise.

- **It is persuasive**, in the sense that its result will make a stakeholder with influence[1], for example a developer, supervisor or customer protest if the problem is not fixed. Setting up tests to reflect actual usage statistics could intuitively be very persuasive to a customer that has ordered a software system.

- **It is powerful**, in the sense that it has a high probability of revealing many bugs. However as mentioned before, no test will be better at finding any type of error.

- **It is easy to evaluate**, i.e. it is easy to determine if the system passed or failed the test.

The above definitions will be used in the following sections. It is also important that the test costs less than its potential benefit, otherwise it may not be worth doing and will in practice probably not be approved by the stakeholders with influence.

3.2 Testing methods

To prove to the customer that the software has reached the desired level of quality, all applicable aspects of the program needs to be tested. However, testing all possible permutations of a program would take too long, and would not be economically feasible [16]. Using only one type of test will probably not be sufficient to achieve the requested level of quality, mainly because some tests are good at exposing certain types of error, but bad at others. There are many types of tests to choose from, all with their distinctive strengths and weaknesses. When the system seems to work from the developers point of view, one should investigate if this is true from the users point of view. If a program has flawless code, but is unable to perform the tasks the users expect from it, or hinder the users in their work, it is still a bad program. It is also worth noting that "Testing is the process of executing a program with the intent of finding errors." as opposed to "The purpose of testing is to show that a program performs its intended functions correctly." [16]. The perspective of the developers may not be sufficient to expose enough errors because they are biased by the fact that the software is their own creation. A successful test should uncover new errors, not "prove" that there are no errors, as the former adds more value to the program and thus makes the test a good investment [16].

A thoroughly tested program has thus undergone testing from different perspectives with different testing methods. A quick overview of the most important state of the art testing methods is given below. In our project, we have already completed the internal black box[2]/white box[3] testing phase (see [12]), and we are moving on to the external testing phase, i.e. testing the parts of the system that are visible to the users.

[1] "A stakeholder is a person who is affected by the product. A stakeholder with influence is someone whose preference or opinion might result in change to the product." [7]

[2] Testing the software as if it is a box you do not know the contents of, but you know which input it accepts and what it supposed to output as a result according to the functional specification of the program [16], [20].

[3] You know the contents of the box, and can use this to your advantage when creating tests to test the code more directly. If the program has a structural specification, white box testing is used to check that the program conforms to it [16], [22].

3.2.1 Module testing

"Module testing (or unit testing) is a process of testing the individual subprograms, subroutines, or procedures in a program" [16]. It is largely a white-box activity, supplemented by Black-box techniques [16].

First analyze the module's logic using white-box techniques, then supplement the tests with black-box methods [16]. Simple functions are tested by giving them normal midrange input and see if they return the expected output. The program only fails such a test if it has an erroneous algorithm, erroneous code or logical errors [7][4].

These tests are the first you would try when you test a program, they are used continuously during development and are cheap and quick to complete. They are highly credible, easy to evaluate but not very powerful [7]. Intuitively, you should mostly find errors you anticipate because you already assume that the part of the program you are testing is supposed to be there. These type of tests will therefore not uncover many higher order errors.

This type of test is normally performed during development, and we completed the necessary module testing at an earlier stage in the development process.

3.2.2 Function testing

The purpose of a function test is to attempt to find discrepancies between the program and the external specification. The external specification precisely describes the system from the point of view of the end-user [16].

Function testing is largely a black-box activity, relying on the previous module testing to test the structure of the program. The specification is analyzed to produce a set of test cases[5] which are then used to test the program [16].

Function testing has been previously conducted, and must be done again if more functionality is added to the system.

3.2.3 Domain testing

Domain testing is testing all extremes of "legal" values the variables can have, specifically borderline, maximum, minimum, and beyond max/min values. Boundary/extreme value errors are common [7], and in our own experience programs can react to such values in somewhat unpredictable ways.

These tests are not necessarily very credible, as they utilize many values that normally no-one would enter, and are therefore unlikely to be a problem.

This type of test is normally performed during development, and we completed the necessary domain testing at an earlier stage in the development process.

[4] [7] says this about function testing, however his definition of function testing fits under [16]'s definition of module testing and is therefore included here.

[5] Making tasks the developers or testers solve themselves, a quicker and cheaper testing method than scenario tests, which require more planning and often involve real users. But test cases could be more expensive on a longer term

3.2.4 Specification-based testing

Specification-based testing tests the system directly against the requirements specification, point by point [7]. The software development community is not in agreement about the importance of such tests, because software quality can be defined in several different ways. The importance of the requirements specification is not clearly defined in these definitions. Some examples are provided in the following:

- Quality is the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs. [1]
- Software quality is divided into software quality factors[6] which can not be measured normally because of their vague description but can be quantified as non-functional requirements through the use of metrics, or measures [21], [5].
- Quality is defined differently across nationalities, in Japan it means excellence, in France luxurious and unnecessary and in America that something actually works. In the product-based sense we normally define quality today as a products abilities compared to the users' needs, demands and expectations. Quality is a characteristic of a product that may well be measured in a way that more/less is better. In a business sense it is ultimately the end-user who determines a product's level of quality [3].

Quality is thus something that may or may not be measurable, and may or may not be defined as functional requirements. If the software's quality is defined by the specification[7] and its level of compliance with these standards can be measured, this type of test is very important and a good indicator of the system's overall quality. If, however, quality is defined as something that is not part of the specification and is not directly measurable, this test can at most be useful to ensure that the contract with the customer has been fulfilled. It will in any case not be able to tell if the software is able to perform a function it intuitively should unless it is a part of the specification. It may be more constructive to test the requirements document for faults, ambiguities and contradictions, and create a specification that is better suited to the customers needs.

In previous projects the specification has spawned a large to-do-list that has been updated as functions have been added and bugs fixed. The specification has also been somewhat incomplete and we have focused our attention on adding quality to the system by revising it. Such a test has therefore been deemed unnecessary.

3.2.5 Risk-based testing

A potential failure in a program can be called a risk. Imagine a scenario where the system might fail, which is at least somewhat credible, and test if the system actually fails [7].

If the scenario is not likely to occur the test may be dismissed by the stakeholders as being of little value, but if you tie it to a realistic scenario, especially market-wise in a bigger company, the test can be highly credible and highly motivating. It should then be given priority by the stakeholder.

[6] "A software quality factor is a non-functional requirement for a software program which is not called up by the customer's contract, but nevertheless is a desirable requirement which enhances the quality of the software program" [21]. Examples are usability, performance, portability etc.

[7] Functional requirements specification, the document this test will check the software against

Since you are trying to imagine ways the system will fail, you will learn a lot, whether it actually fails or not. You may for example think of new risks worth testing.

Risk-based tests were performed to some extent in previous projects.

3.2.6 Stress testing

Stress tests can be defined in somewhat different ways:

- The most common definition is to hit the system with a peak burst of activity and see if it will fail [7].
- "Testing conducted to evaluate a system or component at or beyond the limits of its specified requirements with the goal of causing the system to fail" [2].
- "Feed the system a peak volume of data, or activity, encountered over a short span of time." [16] Under this definition, testing the systems or activities where time is not a factor is defined as **Volume testing**.
- Test the system to find out *how* it fails. The fact that it will or may eventually fail may be less interesting than finding out what happens when it does. For example, if any parts are exposed to intrusion, if the whole system or only parts of it becomes unavailable, does data loss occur etc [7].

You can check if the system can handle the load it is supposed to, or if it can handle an even bigger load. Errors that may occur when the system is running many tasks that in themselves are not too demanding, can be exposed this way [7].

Such a scenario can be dismissed as unlikely to occur because no one would use the system this way (unless it is an Internet application). When security is a concern, especially if it is an Internet application, stress testing is important due to the danger of DoS[8] hacker attacks or due to the fact that Internet applications can become very popular overnight, and suddenly have an extreme increase in server load. Although this is a well known fact, it can still be very unpredictable and therefore worth exploring.

Stress testing has not been done and has not been planned for this project as the software is not an Internet application (although it uses web technology and could have been used on the Internet) and it is designed to handle a lot more stress than it needs to. It runs on a standard platform that could normally handle thousands of users a day, but will only need to handle a few.

3.2.7 Regression testing

Regression tests are re-usable, for the purpose of testing the same aspect with small changes in the code each time. A good regression test should be likely to fail if the code covered by the test is changed in such a way that it causes errors in the program [7].

There has not been many practical ways to use such a test in this project and it has therefore been omitted.

[8]Denial of service, a lot of computers on the Internet flood the server with requests, handshakes or data, until the server crashes. May cause protected data to be exposed

3.2.8 User testing

"User testing is done by users. Not by testers pretending to be users. Not by secretaries or executives pretending to be testers pretending to be users. By users. People who will make use of the finished product." [7]

Simple and straightforward tests with a standardized system for reporting if the system passed or failed can be done by end users without too much trouble, although such a test would obviously not be very powerful, and probably not expose errors the developers didn't foresee while designing the tests. However, to find out what the users are having difficulties with, or where they fail to complete the tasks the software is supposed to help them do, you must put more resources into it. Beta testing is a way to do this, but may be expensive to administrate in practice and may not yield too much information [7]. Another option is doing simulated tests where the developers can watch the users, or full fledged tests "out in the field" where the users can write reports and suggestions, the system can log how it is being used, and the developers can observe the users to record information.

In this project we have used a simulated user test as defined above and made preparations for a full fledged field test.

3.2.9 Scenario testing

A test that is created as a scenario has a story that is believable and details a complex task. The tasks describes what you have to do to complete it, but not how to do it. The system is tested to see if it can cope with this hypothetical situation. According to [9], scenarios come from, and are instantiations of use cases. Variations of scenario tests include soap operas, which presumably got their name because of their similarities with television soap operas. They are based on hypothetical, fictional and overly dramatic stories. Killer soaps are more extreme versions of soap operas, where rare, but still plausible sequences of data are fed to the system [7]. An example could be a string containing illegal characters, which could happen if a user hit a wrong key while typing.

These tests have a high error detection power and gives insight into advanced use of the product according to [7]. We wanted to focus our attention on this type of test in this project and it is therefore described in more detail in chapter 4.

3.2.10 Exploratory testing

Exploratory testing is "any testing to the extent that the tester actively controls the design of the tests as those tests are performed and uses information gained while testing to design new and better tests" [4].

According to [4], all tests fall between a purely scripted one and one that is not scripted at all, i.e. the tester develops the test continuously. The idea of exploratory testing is for the testers to "learn about the software they're testing, the market for the product, the various ways in which the product could fail, the weaknesses of the product (including where problems have been found in the application historically and which developers tend to make which kinds of errors), and the best ways to test the software." [18]. While they test the software they apply their knowledge to create better tests.

According to [7], exploratory testing can utilize any testing paradigm (domain, specification-based, stress, risk-based etc). Following this, we have in our project largely used exploratory testing when designing tests, and little scripted testing. In our experience highly scripted tests are time-consuming and somewhat ineffective, while a more dynamic approach discovers more errors with less effort.

3.2.11 State-model based testing

The visible behaviour of the system is modeled as a state machine[9], and the system is then tested for discrepancies from the predicted behaviour [7].

Automated techniques are normally used to check the software which makes the test easy to evaluate. The test is also credible and motivating according to [7]. Simplifications are, however, often needed because of the sheer number of states, and trade-offs must be made regarding the level of simplicity and detail [7]. State-models can be useful as a guide in creating exploratory tests instead of being used as an actual test [7].

We have previously done the latter to some extent, as some of the tests we have done have been based on state variables and have been highly exploratory.

[9] A state machine is a programming technique that enables a program to have two or more states. The program remembers its state and keeps it until it is explicitly changed

Chapter 4

Scenario Testing

> *Beware of bugs in the above code; I have only proved it correct, not tried it*
> Donald E. Knuth

According to [10], of all types of tests it is the scenario test that is best suited for providing detailed empirical information on those attributes that make up the usability quality characteristic which is defined by the ISO-9126 standard [1]. In our system, KAS, usability is an important quality attribute along with security and scenario testing could therefore prove to be useful. For more information on usability and other quality attributes in our system, look at our work in the previous in-depth study project [12].

4.1 What is scenario testing?

According to [15], there has hardly been any attempts to document and define the exact nature of scenario tests. Our teaching supervisor, professor Tor Stålhane, confirms this statement saying that this testing method lacks good documentation. A precise definition is therefore required. Our definition of scenario testing is as following:

Scenario testing is a software testing method where the tests are defined through tasks described by fictional stories. The stories describe what the tester should do, but not how to accomplish it. The stories should be credible, and encourage the tester to be creative and explore the system's functionality

A scenario test is based on a scenario. A scenario is a hypothetical story, used to help a person think through a complex problem or system [14]. Scenarios aim at describing realistic situations with tasks that reflect real use. This means that a scenario is not only something that could happen, but also something that probably will happen during daily work. It is important to make the tasks as credible as possible.

[10] implicitly states that users are an integral part of a scenario test. We do not include users in our definition, as we believe dedicated testers and developers could benefit from scenario testing as well, as an alternative to test cases. We claim that the essence of scenario testing is that it is based on credible stories which encourage exploration and learning in a software system. However, we acknowledge that scenario tests may have their greatest strength when users are involved and the methods described subsequently implicitly require users to be included.

Scenario testing in software engineering is about the *what* rather than the *how*. The testers

are provided with tasks they are supposed to solve using the system. This way they will know what they are supposed to do, but not how they should accomplish it. They have to explore the system to find out how to solve the tasks. The tests will show how intuitive and usable the system is, and if users are involved they can provide unique perspectives and feedback to the developers. By experience, people learn by doing, not by following checklists or manuals. Learning by doing should make scenario testing more motivating for the tester.

There should be clear indicators that can be used to decide whether a scenario test has passed or failed. If it is not easy to evaluate, the test will loose some of its credibility. The credibility of a test is important to its usefulness. For example, if a scenario test exposes a task that can not be done with the system, this result must be credible enough to persuade a stakeholder with influence to protest, thus forcing the error to be fixed.

Scenario testing involving users is a relatively expensive method in a short term perspective as it depends on a lot of external factors. Future users willing and able to participate must be found, and a suitable time and location must be set. Much planning is also required to make the test realistic enough. However, the long term effects of this effort will be beneficial, because of the unique results a scenario test involving users can provide. Results include feedback on normal use of the system, identification of errors and omissions in its functional requirements specification as well as money saved on training the users later.

With users involved, scenario testing can be compared to black-box testing. The users have no knowledge of the system or its structure, they just provide inputs and observe the outputs of the system.

Scenario testing normally goes through five phases as listed below and described in [10]:

1. *Planning.* Costs, location, who is participating and what to test (system or module) are defined.
2. *Preparation.* Requirements are studied and the test procedures are defined. This is also the phase where instruments (interviews, questionnaire, observation, checklists etc.) are chosen, a test plan is defined and tasks for the scenario test are created.
3. *Testing.* It is important to keep track of time and make sure the test plan is followed in order to stick to the schedule and prevent tests from interfering with one another. It is also important to distribute tasks to users and observe and collect data. Some intervention from the observers could be necessary (like reacting when the testers experience problems). It is also important to document all deviation from the test plan.
4. *Analyzing.* All the collected information has to be analyzed, the less important information should be neglected and the important data should be taken care of. For the important information, statistics could be calculated to represent data in an easy way.
5. *Documentation.* All decisions made during the session should be documented, along with any deviations from the test plan.

To summarize, scenario testing has five key characteristics as mentioned by Kaner [14] and shown in this section. It is (1) a story that is (2) motivating, (3) credible, (4) complex

and (5) easy to evaluate.

4.2 Why use scenario tests?

A tester creates a scenario based on functionality a system should possesses and how the program works or doesn't work, and shows the result to a stakeholder. The stakeholder is thus encouraged to assess the importance of the scenario. By doing this the tester may influence further development and force errors to be fixed. Therefore one could say that scenario tests provide an early warning system for requirement problems that would otherwise haunt the project later.

Kaner lists a number of reasons for using scenario tests in his article [14]. The first thing he mentions is that bug reporting becomes more motivating. The stakeholders will be more motivated to report a bug in the software system to the developers, if they see that the bugs they report are being fixed.

He also discusses that learning the product is beneficial, and as we already have mentioned, people learn by doing tasks that require them to investigate for themselves, not by being lectured or reading large manuals. He says it is less motivating to go through a test following a checklist that someone made for you. While learning a product, the users are also able to compare the system to their needs to see if the system could be used on a daily basis.

Different stakeholders have different needs, and therefore different requirements to the system. As a scenario test session develops, different requirements-related issues could come to the surface. For example a user, who has never before seen the system, discovers while testing the software that it lacks some functionality that he or she often uses. The developers and the other stakeholders do not know about this lack of functionality. This could involve reopening old requirements discussions with new data and inputs or maybe not-yet-identified requirements. As a result of this, the evaluators acquire a lot of interesting information on both the users' behaviour and the users' opinions of the system.

4.3 Methods of scenario tests

There exists two methods to perform scenario tests; field tests and laboratory tests, as reported in [15] and [8]. Both of these methods will be described in the following section.

4.3.1 Field tests

Field testing is testing conducted in the normal environment of the system and the users. The users behaviour is observed by one or more evaluators taking notes, registering time etc. This method should be an easy method for the users to deal with, since they are in their usual environment and do not have to get used to a new one. Elements in a new environment could influence the testing, either positively or negatively, and therefore it has its advantages to keep the users in a surrounding they are used to. The tasks given to the users during the field test should be standardised, which guarantees that every user will encounter the same kind of problems and will have to perform similar operations to succeed.

However, it could be difficult to plan a field test, because of the many variables that are impossible to control beforehand, for example people's movements or the lack of control over the environment around them. It could also be a challenge to separate the important from the not-so-important user behaviour during the testing if observation is the only data collection method used. Furthermore, it is a general rule that the data analysis phase becomes more difficult if not performed directly after the testing phase. Time could blur the situational context in which results were achieved and thus increase the probability of inadequate data, especially in the absence of recording devices. However in a field test the context may be extremely complex and therefore hard to describe to a sufficient degree.

There are four methods often used by the evaluators in field tests, as can be seen in [15] and [19]. These methods are also elaborated further in chapter 5.2:

- Observation, where the users behaviour and reactions are studied. This can be supplemented with checklists.
- Interviewing the users before and after the testing. This could be supplemented with questionnaires.
- Think-alouds, where participants are thinking aloud as they perform their actions. They describe their thinking, feeling and doing as they go about their task while observers objectively take notes without interrupting. This could be supplemented with audio or video recording.
- Log file recording, recording all keystrokes during the user interaction in a separate file.

4.3.2 Laboratory tests

Laboratory testing is testing where the users/participants are moved away from their normal environment, and put into a laboratory environment where they are supposed to solve tasks independent of one another. Here the users are normally observed in a more indirect way, and depending on the laboratory's technical instruments, this could be done either via one-way mirror, video or audio recording or logging programs. Users have to be highly motivated in a laboratory setting in order to deliver useful results. They have to get used to the new environment and the demands the test puts on them. Some travel is usually required for the users, and therefore less users normally participate in a laboratory test than in a field test.

The planning phase in a laboratory environment is not as complex as in a field environment, since the test does not have to fit into a daily working routine. It does not require all the resources and knowledge beforehand from the creators of the test like the field test, and the tasks could be based on somewhat fictitious data. However, like field tests, the tasks in laboratory testing should be standardised to allow comparison of user behaviour. Karat [15] states that, since the variability in the definition of tasks are much greater, laboratory tests are particularly useful if the system under testing is not fully operable. Many software companies conduct laboratory tests at different stages of the software life-cycle instead of using more effort to plan and design the field tests.

	Field test	**Laboratory test**
Environment	Normal working place (least but still slightly obtrusive, same physical/social environment factors)	Controlled, new working environment, integration of developers into tests possible
Test task	Representative integrated task (fits into every-day routine)	Individual tasks (possible to test specific modules only)
Test system required	Operable system or beta version	Prototypes or operable systems
Users	More users/budget	Less users/budget
Instruments	Direct observation (think-aloud, checklists, interviews, logging programs)	Indirect observation (one-way mirrors, video recording, audio recording, think-aloud, logging programs)
Test planning	Complex	Straightforward
Test preparation	Time-intensive	Less time-intensive
Testing	Very difficult	Not so difficult
Data analysis	Brief	Exhaustive

Table 4.1: Important differences between field and lab tests

"The end user's social and physical environment is not replicated in the laboratory environment, and these factors influence the way an end user works with an application" [15]. Whether the task is representative for the daily-work or simply has testing character, also influences the user's behaviour during the test. This is something that one should keep in mind when analysing and interpreting the results of the tests. The analysis process is a bit different for laboratory than for field tests. Everything must not necessary be evaluated at once. In laboratory settings one can have video, audio and logs one can return to in order to get additional data, but it can be an exhaustive process to go through all the material to check whether all relevant information is collected. In addition, one can have checklists or questionnaires after the tests to ensure that as much relevant information as possible has been collected.

4.3.3 Summary

The table 4.1 summarize the most important differences between field and laboratory tests. A more detailed version of this table can be found in [10].

4.4 Expanding the scenario expression

One way to increase a scenario's power is to exaggerate slightly while building it on real experience. When someone in a story does something that sets a variable's value, that value could be made a bit more extreme to get a more "on-the-edge" result. Sequences of events could be made more complicated and for example more people or documents could be added to the scenario.

Hans Buwalda [6] calls the types of scenario tests where you exaggerate for soap operas. The reason for this is the similarities to real soap operas on television. Television soap operas describe life in a way viewers relate to, but the situations shown are typically exaggerated. More things will probably happen to the characters in a television soap opera than most of us will experience in a lifetime. Television soap operas does this to make a better result, i.e. to capture more viewers in the story. In software testing Buwalda's recipe is to make the tests more fun and aggressive by substituting each variable with a more extreme value, and if a scenario can include repeating elements, repeat it lots of times. Buwalda also makes the environment less hospitable to the software during the test with for example increasing/decreasing memory, resolution etc. Hans distinguishes between normal soap operas and killer soap operas. Normal soap operas combine many issues based on user requirements typically derived from meetings with the user community and probably don't exaggerate beyond normal use. Killer soap operas combine and exaggerate the most extreme values to produce extreme cases. Killer soaps aim at finding hidden problems, typically those that would be discovered after a while by stress testing or some other kind of extreme use of the system. The "specialists" are typically asked for input to these tests. These are the ones with most knowledge of the scenario at hand, that be developers, leaders or other stakeholders. Killer soaps are often run when every other tests has passed in order to really test the system and the formula is to exaggerate each aspect of the test so that one would get a more extreme result. A killer soap could be seen as a domain test that is based on a scenario.

Part III

Focus

Chapter 5

Research

> *Every program starts off with bugs. Many programs end up with bugs as well. There are two corollaries to this: first, you must test all your programs straight away. And second, there's no point in losing your temper every time they don't work*
> Z80 Users Manual

This chapter describes the research focus and the data collection methods used in this project.

5.1 Focus

The KAS had been implemented according to the specified requirements, but was still not ready to be put into active use. In order to reach this goal the users would have to test the system to see if it was usable and if it lived up to their expectations. And most importantly, they had to try it to see if it could be used to do their day-to-day work.

If the users didn't approve of the system, the migration from todays paper-based system to this software system would be problematic at best. Testing was needed to facilitate this migration as painlessly as possible, and for this project it seemed prudent to utilize scenario testing combined with user testing.

- Scenario testing because at this point in the development process we needed a high-level software test that could reveal errors and emissions that a specification-based test (see 3.2.4) could not. We needed a test that could expose requirements-related issues that may not have been identified earlier.
- User were involved because the system had not previously been tested on normal users and only they could ultimately assess whether the system was practically usable or not.

By combining the two types of tests we could get user feedback and high-level software testing at the same time. The scenario test is, in addition to the points above and according to [14], suitable for learning to use the system. This would greatly increase the users' ability to provide meaningful feedback on the software. We also wanted to see how scenario testing could be effectively used in a software system at a late stage of development. We were interested in finding out if scenario testing could provide the feedback needed to complete the development of the system, and how the users would experience this type of test.

In the in-depth study project last year we focused on quality attributes in software architecture[1] and what they meant to end-users. We decided to divide the software testing process into internal and external testing according to the ISO-9126 standard [1]. The internal testing phase tested the system for quality from the developers and the administrators' point of view, while the external phase was part of the current project and involved users in the testing process.

5.2 Methods of data collection

According to [11], the methods of data collection during software testing can be divided into two categories; namely manual and automatic data collection. In this section we will elaborate commonly used methods before we describe which methods that specifically suits our project.

5.2.1 Manual collection

Manual data collection is defined as gathering data by hand. There are several ways to do this, and common alternatives will be described in the following subsections.

Questionnaires

Questionnaires are often used in software testing, both in the development phase and the evaluation phase. Questionnaires can be used to collect both quantitative and qualitative data. Usually a sample which is representative for the population to be studied are given the questionnaire, and the results are first analyzed and then generalized to the population from which the sample was taken [23]. The questionnaire could be either a form to be filled in or an on-line system, both of which have their strengths and weaknesses. Based on experience, a form to be distributed is quick and cheap to make. An online form can be dynamic, it can be updated during the course of the collection phase and the results are instantly available. A more detailed discussion of these issues exists in the report from the previous in-depth study [12].

One general problem with questionnaires, is that the choice of questions will limit the data one will receive. They way the questions are asked could implicitly suggest the "correct" answer [11].

In this project, we wanted to use a questionnaire as a supplement to observation after the scenario test phase. The reason for this was to capture information that would not be captured by observation only. The questionnaire could not be too extensive as the test subjects were supposed to fill it out at the end of each test phase, and they were not supposed to spend more than five minutes completing it.

Checklists

According to [19], the aim of using checklists in general software testing is "to obtain a concise and coherent description of the system in terms of objects, attributes, functions, relations between objects as well as between objects and functions, dialogue state, selections and estimated usability".

[1] More information on software architecture and quality attributes can be found in [5] and [1]

Checklists are often rated, either with simple yes/no (or true/false) values or with a more comprehensive score, ex. a scale from 1-5. Checklists should aim at capturing the essence of the data you need in a way that is practical to register. The contents of checklists should depend both on the user and the system specification. For this reason checklists are often project or system specific. It is easy to fill out inspection checklists, but effective use of this technique in scenario testing is difficult according to [19]. This is mostly due to the fact that the observer has to do two things at the same time, i.e. both observing and filling out checklists. If you concentrate on observing the users actions it is difficult to follow the structure of the checklist as the user may not follow this structure at all. If you, on the other hand, concentrate on registering a single piece of data for the checklist, you will probably miss other useful information the user could have provided. Checklists could intuitively be useful in scenario tests for collecting quantitative data while observing users if such data are considered important enough to take priority.

Checklists was not planned to be used for this project. This was mostly because of the "strict" layout of a checklist as described above. We wanted to base the data collection on a more open approach as all the test subjects probably wouldn't work in a predictable way.

Interviews

Interviews are an important part of most scenario tests. In an interview one or more subjects are asked questions by one or more interviewer(s). During the planning phase of an interview, the time and place for the interview is important to decide. Concerning the point in time at which interviews are typically performed, one may distinguish between pre- and post-testing interviews [8]. Pre-testing interviews are performed in order to elicit the subjects' personal backgrounds, opinions and expectations concerning the system that is going to be tested. Post-testing interviews are an important part of each scenario test. They are performed after the observational data, i.e. video tapes, checklists, notes etc., are analyzed. Each aspect that needs further clarification is taken up in the post-testing interview.

A combination of pre- and post-testing interview is possible and could provide useful information, since it allows the assessment of the change of mind of subjects during the testing exercise. The major advantage of interviewing lies in the fact that, unlike e.g. questionnaires and observations you capture exactly the data you need through conversation. Interviews can be recorded on tape or video. One major disadvantage of interviews is that the lack of anonymity in personal interviews may drive interviewees to suppress important or add exaggerated information. In general, the success of an interview is largely dependent on the interviewers skill.

Performed in combination with other instruments, as observations and questionnaires, interviews are an important part of any data collection during tests involving users.

Interviews were not to be used in this project due to the time it takes to perform a good interview, and the limited time available during each testing phase. We did however plan to have a conversation with each user after the testing to elicit more valuable data from them. This conversation could be seen as a post-testing interview, but with an open structure as we did not plan to prepare any questions, and the time spent on this conversation

would have to be minimal. We would have to base each conversation/interview on the data collected during the testing, and some test subjects could receive more questions than others based on this data.

Observations

"Observations are the most important instrument in any kind of software test involving users. Observations can deliver results on all user-related quality characteristics" [11].

There are two types of observations, direct and indirect. Direct observations are done by one ore more evaluators sitting close to the subject, watching and taking notes. It could be difficult to observe users without intruding or interrupting, and this could alter the results of the tests. This is something that one must have in mind as observer. Indirect observation is conducted differently. Video recording, audio recording or one-way mirrors are usually used. Video is not as intruding as direct observation, but provides the same overview of user behaviour. The major advantage of indirect observations, is that the data could be reviewed as many times as necessary to get the relevant information in the data analysis phase. However, reviewing the material many times is a time-consuming job.

Observations could be supplemented with interviews and/or checklists to be able to verify the information noted and to get even more information from the users.

In this project we planned on using direct observation as the main data collection method. We wanted to observe each user's behaviour and progress, while they tested the system based on scenario tasks. We would have to take notes while observing in order to remember the important observations. The collected data would be subjective as it would be solely up to us which information would be important and worth noting. To secure the collection of all the important information we wanted to supplement observation with a questionnaire at the end of each test. This way we, as developers with full knowledge of the system, could observe and elicit information we knew to be important at the observed time, and the users would be given the opportunity to provide additional thoughts and feedback afterwards.

Think-alouds

"The motivation behind using think-aloud protocols is to collect information on the users' own reasons for their behaviour. The collected information needs to be evaluated carefully, since thinking aloud presupposes that users are able to describe their actions, which is only true for users trained to verbalize their thoughts" according to Vainio-Larsson [19].

She describes several problems with the think-aloud approach, and suggest that they are used as a complement to other data collection methods to ease interpretation, rather than being used alone.

We did not plan to use think-alouds in this project. The reason for this was that we didn't see this as a valuable method in our case, as this method is based on the users being able to grasp their own behaviour and verbalize all their thoughts, and we had no way of knowing whether our test subjects would be trained in doing so.

5.2.2 Automatic collection

Automatic data collection methods are concerned with programs or instruments that are used to perform tests on other software systems with the purpose of gathering data. Most automatic test instruments are developed for testing specific types of software. They can not be used to test other types of software. Automatic collection instruments elicit both qualitative and quantitative data and are useful supplements to manual test instruments in user-oriented software testing. They collect large amounts of data and do not intrude on the user's thoughts or activities [8].

Logging and playback programs are general data collection programs that are external to the software under testing, i.e. no changes are necessary from one application to another. It can be used with actual product code or prototypes of the user interface of a product under development. "The application of toolkits for logging programs which are available on the PC market generally requires intimate knowledge of the systems under testing; since the interfaces between the application and the logging program need to be specified" [8].

Since automatic collection largely depends on test programs, and these test programs usually are made for a specific task, doing automatic collection in this project would not have been possible as our software is unique and does not have any standardized interface for such a tool. We did not have the time or resources to make such instruments to test the KAS, and we did not find any existing programs that could test our software this way.

Chapter 6

Planning

> *The major difference between a thing that might go wrong and a thing that cannot possibly go wrong is that when a thing that cannot possibly go wrong goes wrong it usually turns out to be impossible to get at and repair*
> Douglas Adams

This chapter outlines how this project and the main parts of it were planned. The first section gives an overview of the project as a whole. The following sections will go into more detail on the planning of the two test phases. The last section of this chapter shows the overall project plan.

6.1 The project

In order to complete the KAS it had to be tested, both to be accepted by the users and to see if the system could run in its intended environment. At the start of this Master's thesis we created an overall project plan, see chapter 6.4. The project was planned so that the KAS would go through two major test phases. In the first phase, the system was to be tested by the users to see if they understood the system and if it could prove useful for them. This could be done via a laboratory test during the course of one day. In the second phase, the system was to be tested in the hospital's environment in order to see if it would run as intended and if it could replace todays paper based system. The second phase would have to last for a few weeks, in order to see how the system would behave over a longer period of time. After these two test phases, the system would be of sufficient quality to be put into active use at UNN, or it would at least be clear what the system lacked to achieve this.

Based on the theory in chapter 4, the first test phase was to be a scenario test where the users would be given standardized tasks to solve. There are four reasons why we chose to perform this kind of test. First, we wanted to increase the quality of the system. With this test we could remove potential bugs and obvious function errors that would have occurred at a later stage, that probably could have disrupted the second test phase in its real environment. Second, we wanted to explore scenario testing as a testing method and conduct scenario testing in order to see how well it suited our project. Third, we wanted to see if the users appreciated our system. Finally, some form of simulated test before a real test was important because door access control is a crucial part of the hospital's day-to-day operation and minimizing the risk of its failure is important. If a real test had revealed critical errors it could have created severe problems.

The scenario test was planned as a combination of field and laboratory tests as seen in chapter 4.3. It would be a field test because it would be located in the hospital's environment and the users would be observed by us, taking notes at the same time. It would also be a laboratory test because the system would not run at the hospital's server in its usual environment. It would take too much time to organize the system on the hospital server, and we wouldn't have the possibility to correct and edit the database during the test session. It was apparent that such a test would require a lot from us as evaluators, as it would be solely up to us which information should be considered important.

We did not plan to use scenario testing for the second test phase. We wanted the users to perform the testing during their normal day-to-day work, to check if the system covered all requirements, in order to replace the existing paper-based system. The system would be tested with "real" tasks as a larger number of users performed their daily work. This testing would be a full fledged field test since the system would be operating in its real environment on the hospital's server. The test would not require anything from us as evaluators during the data collection phase, as direct observation of the users would be impossible because of the long test period and large area to cover. The data would have to be collected through a feedback function implemented in the system, where all entries are stored in a database.

6.2 The scenario test

A detailed plan for the scenario test is presented in this section.

For the scenario test, the plan was to observe users and take notes of their behaviour while they, two at a time, solved different scenario tasks in the system. After each pair of users finished their scenarios, we planned to give them a small questionnaire/feedback schema in order to capture data we didn't capture during the testing. They would then be able to explore the system while they filled out the questionnaire. Every user would be given a set of predefined tasks to solve with the system. The various types of users have different responsibilities and tasks in real life, and different access levels in the system. Therefore the system would have to contain some predefined data for each scenario.

Of the five phases of scenario testing, as seen in chapter 4.1, we combined the first two phases, planning and preparation, so that we had to go through four phases during the scenario test. The location, test procedure and test plan were defined simultaneously. This phase is described further in the following sections. The testing phase is described further in chapter 7. The analysis and documentation phases were partly combined as we began to document the test at the same time we began the analysis of the results. The analysis phase is covered through the discussion in chapters 7 and 9.

6.2.1 Laboratory arrangement

For the test we needed a laboratory, in the sense that it would be a place under our control outside the context of the test subjects' normal working environment. After some e-mail correspondence with our contacts at UNN[1], we decided that a conference room at the hospital was a suitable location as it would minimize the time the test subjects would need to leave their normal work, and other resources needed would be nearby.

[1]Our contacts are mentioned in chapter 1.2

Time	Type of user
09:00-09:30	Time-limited card orderer
09.35-10:05	Time-limited card orderer
10:10-10:40	Time-limited card orderer
10:45-11:15	Time-limited card orderer
11:20-11:50	Personnel department
12:00-12:30	Lunch
12:30-13:00	Personnel department
13:05-13:50	Administrators
13:55-14:50	Security personnel
15:00-15:45	Security personnel

Table 6.1: Time schedule for the scenario testing

We would need at least two computers for the test subjects, as we were two developers and could then simultaneously observe one person each. We would also need access to the University's main server or an additional local pc set up as a server. Although using the University's server would enable us to test the system in its real working environment, potential problems would interfere with the user tests and degrade their quality. We would have little time to fix such issues once the system was online. We therefore decided that we needed a third pc in the lab to approve/deny orders (part of the scenarios the users were to be subjected to), so the system would appear to be as realistic as possible to the users.

6.2.2 Schedule

Based on the arrangement in the laboratory, we planned the test to last for two days, one for the installation and preparation of the laboratory and one for the test. The preparation of the laboratory would have to be complete before the test day, and to be sure we had enough time to cope with any problems that could arise we reserved a day for this purpose. By experience unforeseen problems "always" arise in a new environment.

In the schedule we estimated each test to take about 30 minutes for the card orderer users in the system, and 45 minutes for the administrators (our two contacts) and security personnel. We planned for a short break for about five minutes between each pair of users for cleaning up the system (the data the previous users have entered into the server) so that it would be ready for the next group. When the next pair of users had another user level than the previous ones, we would have to enter new test data into the server. Based on our estimates, we made a schedule in cooperation with our contacts. This schedule can be seen in table 6.1. At each specified session there were two users to observe, one for each of us. This made a total of eight time-limited card orderers, four from the personnel department (those who order ID cards), four security personnel and two administrators. This gives a total of 18 users to participate in the scenario tests.

6.2.3 Scenarios

In order to carry out the scenario test, tasks for the users had to be created. We decided to create these tasks ourselves, since we knew the system in and out, without involving our contacts at UNN. We had, however, no experience with the users' work issues and the situations that could arise during a normal work day, so all tasks was created based on intuition and the example data for each task were largely imaginary. The tasks were based on our knowledge of the system, and were made so that the users would encounter situations that could occur in the system on a daily basis.

While creating the scenarios, the first thing we did was to make a list of the tasks each user should do. This list was divided into three user groups and is shown below.

Time-limited card

- Order a time-limited card to a person that exists in the database
- Order a time-limited card to a person that does not exist in the database
- Order a time-limited card to a person that has lost his card
- Edit a submitted order that is waiting to be confirmed
- Correct a rejected order according to the feedback message

ID card

- Order an ID card to a person that exists in the database
- Order an ID card to a person that does not exist in the database
- Edit a submitted order for an ID card that is waiting to be confirmed
- Correct a rejected order according to the feedback message
- Order a change in a person's personalia

Security Personnel

- Accept and confirm an incoming order for a time-limited card to person one
- Order an ID card to person one
- Accept and confirm this order for an ID card to person one
- Accept and confirm an incoming order for a new ID card to a person that has lost his old card
- Deny and reject an incoming order, with a message of why it was rejected
- Terminate a card where the job has ended but the card is not registered as returned
- Register a card that is handed in as received
- Accept a change in a person's personalia

From the above list, scenario tasks were made with some help from Kaner's *Twelve Ways to Create Good Scenarios* [14]. Each scenario task was constructed in two versions for each user group, but with different data, to allow two test subjects to do the test simultaneously. If both test subjects entered the same data into the system, multiple identical entries could create confusion because the test subjects would have no way of knowing which entry was meant for them.

Chapter 4.1 listed the characteristics that a good scenario should have. The following list shows how our scenarios conform to these requirements:

1. It should be a story. Our scenarios try to tell small parts of stories throughout the tasks. The stories are quite simple but more elaborate ones may not have been suitable in this context.

2. It should be motivating. Our scenarios were realistic and not too complex, we tried to recreate plausible situations from normal work.

3. It should be credible. We have tried to make the stories as credible as possible given the information available, however they are still fictitious.

4. It should be complex. Our scenarios are quite simple, but with some complex elements. If they were too complex they may have been less motivating to the test subjects.

5. It should be easy to evaluate. Our scenarios had clear tasks and it can be easily evaluated whether the task has been completed or not.

The completed scenarios are shown in appendix B.

6.2.4 User observation

The most important aspect of this test would be observing the users behaviour and collecting key data while they worked on the scenario tasks described earlier in this chapter. Thus the data collection method most relevant to this project was observation, see chapter 5.2.1. According to [17], an important aspect of observation is deciding which role you are going to play. An observer must decide which degree of participation he/she should have. The roles one can have varies between total observer, participating observer and total participant. In most roles the evaluators can choose to perform either openly or hidden. If the role is a hidden one, the users don't know that they are being observed. An observation could also either be done directly or indirectly as described in chapter 5.2.1. As [17] says, the observation role could change over time.

[13] mentions the outsider/insider myth. Some believe that good research can only be done by observing form afar and maintaining objectivity, while others believe that it is necessary to understand a group from the inside to fully understand it. Both points are worth considering, and [17] points out that the best solution often is a compromise where you try to understand the test subjects situation, while trying to maintain a certain distance. We planned to take an open and partially participating role ourselves, where we could give the test subjects hints but leave it to them to explore the software system, to be able to observe which parts that were problematic.

We came to the conclusion that we didn't want to count and classify any errors and bugs

exposed during the observation, only note down the errors in order to correct them at a later stage. We could have recorded the time each user used on each task in order to capture the usability of the system, but we felt that such a measurement would not be good enough for such a complex quality factor. What we wanted to look for during the observation was how the users perceived the system, if they understood its functionality and how their daily work was to be done. We also wanted to register every function that seemed like a source of irritation for the users, and the users suggestions for improvement of the system. We felt the best way to do this was to observe the test subjects closely without looking for pre-determined metrics so that we could continuously assess which information was important. A questionnaire was created for the test subjects to answer afterwards, containing questions that may not come up naturally during the test session. On request from our supervisor, professor Tor Stålhane, we also made a registration form in order to register each user with name and either a phone number or e-mail address. This way they could be contacted at a later stage to clarify any confusion or ask additional questions.

6.3 The field test

A detailed plan of the field test is presented in this section.

6.3.1 Schedule

We estimated this test to last for approximately three to four weeks, and start when the implementation of the changes derived from the scenario test was completed. For this test the system would be installed in its working environment, the university's intranet/SQL-server. The system's functionality would have to be tested after installation to check for any errors due to changes in the operating environment. This would have to be done before the test started, to ensure that the test would not be hampered.

6.3.2 Data collection

We would not have the time or opportunity to observe the users on site, i.e. while they worked normally. We would thus have to collect data in some other way. We had the possibility to collect data through a feedback function accessible from the system itself, or from questionnaires at several points of time during the test. Making a feedback function in the system was clearly the best option, because this way the users could register errors or other feedback at once while using the system. During the test we could also check the feedback from the users to continuously improve the system and fix errors. They would, however, have to use this function actively, or else we would be left with no results from the test at all. Thus, we would have to clearly specify to the users that their opinions and meanings about the system would be highly valuable, and that the feedback received could lead to changes being made.

6.4 Project plan

The project plan is shown in figure 6.1. This plan was made in the beginning of this project and was meant to be a guideline to help us keep track of the time spent on each activity. Each activity/phase is connected to a date. The date was when we ought to be finished with that activity, and ready to start the next. Each activity depended upon the

completion of the activity before. The figure doesn't show the actual time used on each activity, only the planned usage. The different activities in the project plan are described below.

- *Implement changes.* Some changes had to be implemented in the system. These changes were corrections and improvements discovered during the in-depth study [12] last term. It was important not to spend a large amount of time on this activity in order to begin the testing as soon as possible.
- *Plan simulated user test.* This was the phase where the scenario test would be planned. During this activity theory and literature also would have to be studied.
- *Performing user test.* The scenario test was planned to be conducted in Tromsø at UNN during a day or so.
- *Analyze results and implement changes.* The results of the scenario test would have to be analyzed and the identified issues resolved before the extensive full test at UNN.
- *Plan a full test.* The full test would have to last for a month or so, in order to observe the performance of the system over a longer period of time. The planning would have to be done carefully in order to properly collect all data from the users.
- *Execute the full test.* The execution of the full test would require significant resources from UNN and much coordination would be required.
- *Analyze results and complete the code.* Results from the full test would have to be analyzed, and erroneous and superfluous functions should be improved in order to complete the system so that it could be delivered to UNN.
- *Write documentation.* In this phase technical documentation of the system was planned to be made so that UNN could modify the system at a later stage without our help.
- *Write the report.* Writing of the report was planned as an ongoing activity during the whole project.

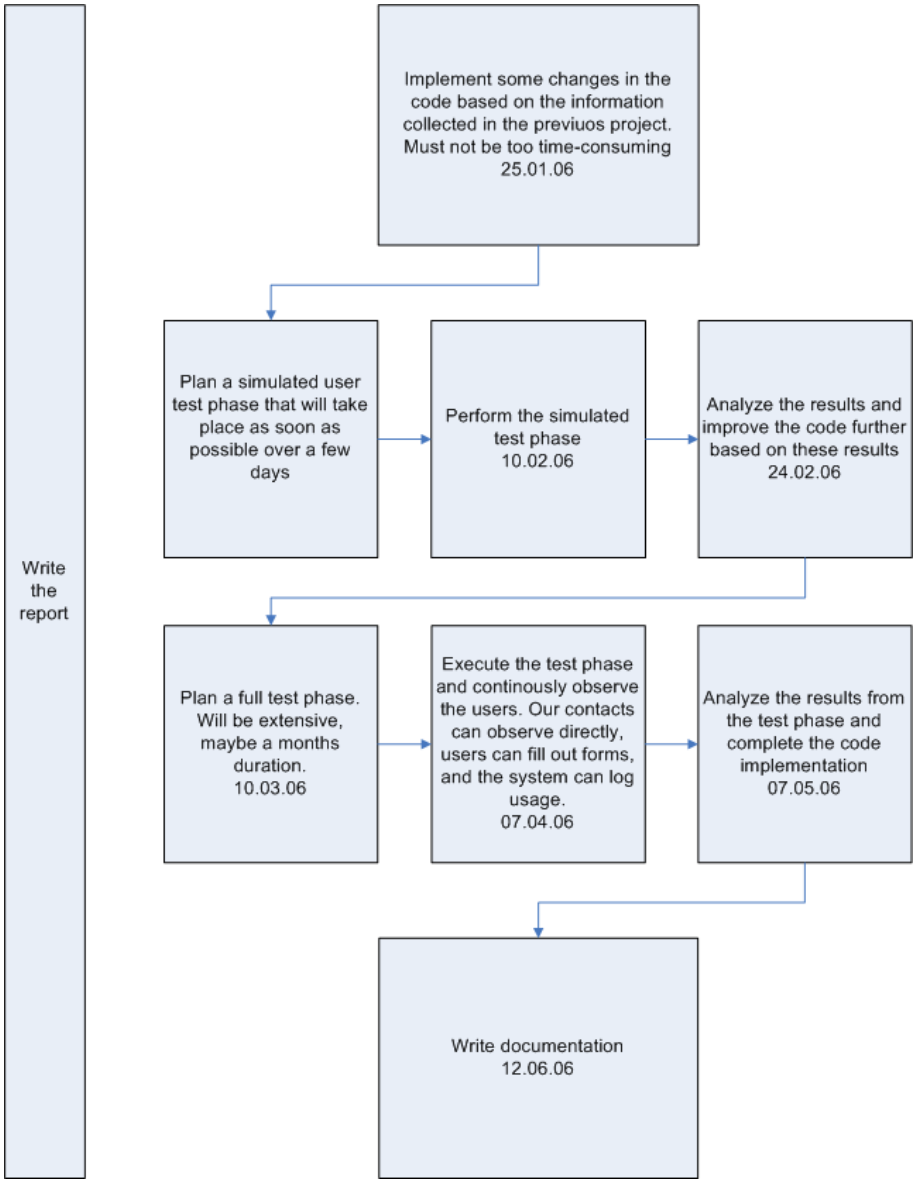


Figure 6.1: The project process

Part IV

Test sessions

Chapter 7

The scenario test

> *One: demonstrations always crash. And two: the probability of them crashing goes up exponentially with the number of people watching*
> Steve Jobs

The scenario test will be described in this chapter. Chapter 7.1 details the work done before the test and the results from the test are presented in chapter 7.2.

7.1 Preparation work

After the planning of the project (see chapter 6.2) we had to clarify some issues before the scenario test session was conducted. Implementing changes from previous project, finding a suitable date, organizing test subjects and preparing the laboratory were important activities.

7.1.1 Initial implementation

Based on the results obtained from our previous project [12] we were already aware of some errors, in addition to other elements that were designated as future work in the existing requirements specification. We had to choose the most critical and at the same time not too time-consuming elements to fix before the scenario test, because the testing had to be completed as early as possible. To ensure that the test yielded the best possible results, the test subjects should not have to worry about known errors. The following changes were implemented prior to the scenario test:

- The date fields had an awkward syntax and was replaced by a graphical calendar (probably the most important issue to fix)
- The 'Stilling' field was included in some pages were it was missing (not critical, but easy to fix)
- Some security holes were plugged (not necessarily critical in this phase, but still important)
- One of the user classes could not register new users in the system. The error was fixed
- Added possibility to remove user privileges in the register new users page. The layout of this page was also improved.

- Added a page for security personnel for accepting changes to users names and departments (necessary function in daily work)

7.1.2 Laboratory preparation

The installation and preparation of the laboratory went well for the most part, the IT-department at the hospital had arranged three computers for us in a conference room with access to their intranet. We encountered some technical difficulties mostly because of a bug in MS SQL[1] server that it took some time to work around.

We had a short meeting with our contacts to explain our plans for the following test day. They suggested that to increase the level of realism of the test and decrease the number of factors that could thwart it, we should update our scenarios with the real names and departments of the applicable test subjects. They also suggested that we should use their existing paper-based ordering system where applicable in our scenarios to further increase realism.

We allotted one hour for these changes during the preparation, however none of them were done because of a critical technical problem we encountered towards the end of the preparation which had to take priority. The system has been developed exclusively for IE[2] because no other web browsers are used at UNN, but a bug of unknown origin prevented the system from working properly in the laboratory environment, and the Firefox[3] browser had to be used.

7.2 Results from the test

The test yielded few quantitative results but produced useful qualitative feedback. As such results are subjective and are difficult to quantify, some discussion will instead be provided in the following sections. The observations can be seen in appendix C.1, replies to the questionnaire in C.2 and the collected functional requirements, or additions to the initial functional requirements specification, are located in appendix D.

7.2.1 GUI - intuitiveness

As determined by the ATAM[4] analysis in [12], a high level of security and usability are the most important qualities this software must possess, and although modifiability[5] is important, the former two takes priority. Security had already been tested in [12], therefore the scenario test session was for the most part a test of the GUI and its intuitiveness to the end users. The test subjects did have the opportunity of stating their opinion on and inquiring about any part of the system, including its level of security, but almost no one did. Performance has not been a priority throughout the development of the software system (see [12]), primarily because of a relatively low amount of users compared to the operating environment's capabilities. The hardware environment for the KAS is

[1] Microsoft's implementation of the SQL database standard

[2] Microsoft Internet Explorer 6.0, Microsoft's current web browser at the time of writing

[3] Mozilla Firefox, a popular web browser. See http://www.mozilla.com/firefox/

[4] Architecture Trade-off Analysis Method, a way to reveal how well a software architecture satisfies particular quality goals, and provide insight into how quality goals interact, and trade off. See [5] for more information.

[5] The ability of the software to handle changes and additions to the code without spawning bugs and negative ripple effects throughout the system

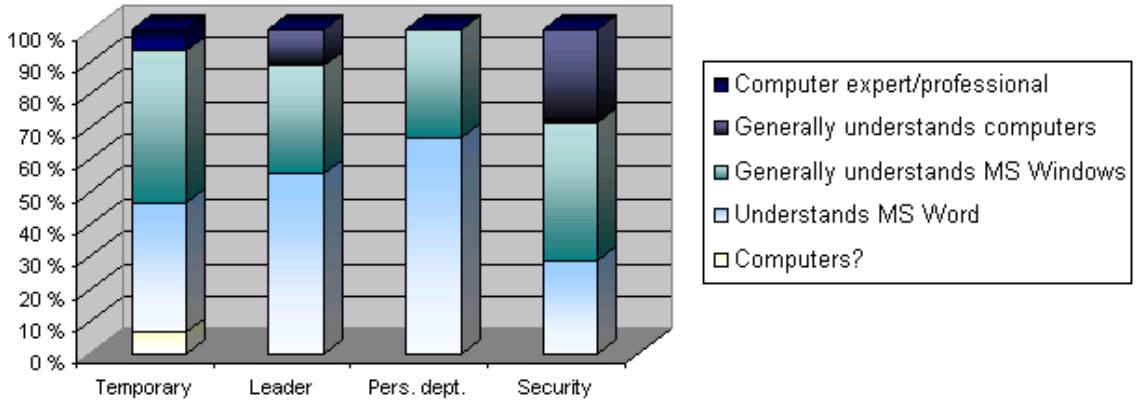


Figure 7.1: The computer skills of the participants in the previous in-depth study

already scaled for handling much more time-critical and performance-hungry systems.

Most of the feedback from the test (see appendix C.1 and C.2) did indeed concern usability issues, and the system's GUI. The test subjects had not received any training prior to the test, they all saw the system for the first time. But apart from minor errors, no one seemed to have any significant difficulties completing their tasks. We inquired each test subject about his/her computer skill, and this skill level seemed to be the most important factor concerning feedback on the system and the time spent on solving the given scenarios. This tendency was also previously identified in the previous study [12]. Figure 7.1 shows how the participants of that study assessed their own computer skill level by completing a questionnaire. These data are based on a larger population (46) relative to the current study (18), and its quantitative approach makes it better suited to generalization of the results to the entire population. Both studies are based on the same population, all the employees at UNN that will use this system, which is about 500 people.

In the previous study [12] the test subjects' computer skill was assessed by themselves through a questionnaire. In this study the test subjects were asked during the test to assess their skill level, which was also later observed by us. In the first study the participants from security rated their skill level highest and the participants from the personnel department lowest. In the current study the test subjects from the personnel department was observed to have much better skill than previously assessed, and they had least trouble learning the system during the scenario test. Both studies show that the security personnel had relatively high computer skill. The administrators did not participate in the first study but they have, to our knowledge, the highest skill of all user groups. The security personnel had significantly more advanced tasks to complete than the other groups, which should explain why some of them needed much time to complete their tasks. For all types[6] of test subjects it was apparent and not surprising that those that required the most time and help were indeed those that pointed out their lack of computer skill. As it was apparent that good computer skill lead to short time used on the tasks and quick learning of the system, it was apparent that this again lead to increased happiness about the system. Thus, we observed a correlation between computer skill and happiness with the software.

[6]Temporary, personnel dpt., security, admin

Some users reported that completing tasks in the system took more time than with the existing system (see appendix C.1). This appeared, through the test subjects observed behaviour, to be a general problem. This should, however, be greatly alleviated by a minimum of training, and good user manuals. Indeed, the test subjects worked faster as they became familiar with the system. The quality of the key card orders will in any case be much higher than before, which was a primary concern and a driving force behind the creation of this software system. This was also pointed out by some of the test subjects. A primary goal of the test was to expose where the GUI failed so that it could be improved to satisfy the requested level of quality, which should also reduce time usage in the future.

We had anticipated that the test subjects would be sceptical of yet another computer system in their daily work, but the feedback was almost entirely positive (see appendix C.1 and C.2). One of the test subjects, who according to himself had very low computer skills, thanked us because the task of learning a new system did not seem as daunting as he had feared. This again shows that our GUI is simplistic and intuitive, and that the windows standard not necessarily is the best (see section 2.5 for a description of differences) for creating easy-to-use graphical user interfaces.

7.2.2 Test setup

The number of test subjects was small and any conclusions based on quantitative data collection would have had a low level of confidence. We therefore felt that direct observation followed by an informal conversation not as rigorous as an interview, would be more productive than devoting our attention as evaluators to record time spent, or other typically checklist-based data. We chose to adapt the test to each test subject instead of rigorously measuring quantitative data for each subject. This was a success as the test subjects felt more at home and loosened up during the test, speaking more freely and giving feedback as they saw fit. We received many thoughts and perspectives we had not previously thought of, and we believe the test setup was an important reason for this. This valuable feedback would probably have been lost if the test subjects only were to answer our pre-defined questions. We are certain that encouraging test subjects' creativity during a test will yield better feedback, and valuable alternative perspectives and opinions.

Our role throughout the test was supposed to be an participatory observing one. To better observe which parts of the system that appeared cumbersome to the test subjects or to measure quantitative data we should have maintained a more observatory role. It was, however, difficult to avoid helping and guiding the test subjects through the given scenarios. Some were not comfortable with computers and some were confused by Firefox's auto-complete function[7] when filling out forms. We have, as [17], experienced that actual field work requires the researcher to fight for his/her definition of the role, and that this role can change over time.

We did not have time to alter the scenarios to reflect the names and departments of the test subjects as previously mentioned. Even so, it did not seem like the they had any problems relating to fictitious scenarios. Our scenarios seemed to be straight-forward and nothing indicated that fictional data were a source of confusion.

[7] A window pops up when you highlight an input field on a web form, presumably to show auto-complete alternatives

The questionnaire (see appendix C.2) was too large and extensive. Some of the questions were hard, if not impossible to answer meaningfully for a user with only 15-20 minutes of experience with the system. None of the test subjects answered all the questions. The questionnaire seemed to create a sense of "a lot of work" as opposed to the intended "quick to complete". It was, however, evident that some of the questions were useful in highlighting issues that did not emerge naturally during each test session. The questions and replies from this form can be found in appendix C.2.

The schedule was a perfect fit and every test subject showed up on time thanks to our contacts at UNN. The test subjects were happy with the arrangements, and some department heads were anxious to put the system into use and even volunteered their departments for further testing phases. We did not collect contact information for the test subjects as planned, because it did not seem necessary and we managed to clarify any issues that emerged, during the tests.

7.2.3 Functional requirements

The test yielded a lot of additions to and refinements of the functional requirements specification. These were the most important results from the tests with regard to the software system itself. These requirements were refined and weighted based on similar principles as [5] weights scenarios in an ATAM analysis. Each requirement received a high/medium/low rating on its importance, and its estimated time needed to implement. A requirement with a rating (H,L) denotes that the requirement has high importance and should take little time to implement. The requirements were then prioritized according to these values and the result was a prioritized to-do-list that had to be implemented before the final test phase. This list can be found in appendix D. The scenario test was thus successful as a high-level software test, as we feel that the overall quality of the software system was greatly improved by the test.

7.2.4 Other results

It became apparent that the existing paper-based system was not being used as intended by all employees, which was reflected by some of the new perspectives we received from the test subjects. They pointed out several workarounds they normally used to overcome weaknesses in the existing system. If such information had been available during the design of the software system, changes would probably have been made to the functional requirements specification. This was a useful result from qualitative research.

Both we and our contacts were somewhat surprised when some of the users pointed out that they did not want it to be too easy to complete some of the tasks in the system, as it could compromise security. We had not anticipated that the end users had such concerns, but it should be mentioned that the users in question were department leaders, with the accompanying responsibilities.

Chapter 8

The field test

If people never did silly things, nothing intelligent would ever get done
Ludwig Wittgenstein

This chapter describes the preparation done prior to the field test. It also describes why this field test must be conducted at a later stage.

8.1 Preparation work

We implemented the new functional requirements from the previous test phase, see appendix D. This phase was supposed to take two weeks, but it actually lasted for about six weeks. We should probably have set a limit for how much time we were going to use implementing these requirements, but we wanted the system to be as good as possible. Almost all requirements were implemented due to either high importance or low implementation time.

When the implementation was done there was some issues that had to be addressed before the system could be installed at the hospital. Since the security personnel are the most important part of the chain, they had to be taught how to use the system. Their tasks are also quite complex. There was two ways to teach them, we could either teach every security personnel during a one day course or we could make a user manual or a combination of both. We planned to write a draft of a user manual, and send it to UNN so that they could return it with comments. We could then do the necessary changes to make it acceptable for use. But because of issues described in the following we made complete user manuals for all user types in the system, not only for the security personnel. These user manuals were illustrated and the system's functionality was thoroughly explained in order to be able to teach the users the important aspects of the system. The user manual for the security personnel is shown in appendix F.

The system was now ready to be installed at UNN. Ideally the system should have been tested by us, since we had the best knowledge of the system, to confirm that the system was working after the installation at UNN. As previously noted, we experienced unforeseen technical difficulties when installing the system locally on the intranet during the scenario test phase. However, the IT-department at UNN wouldn't give us remote access to their intranet, but they tested it themselves shortly after the installation. This test was, however, probably not thorough enough as we received no transcripts or results from this test. And as far as we know they did not find any errors.

8.2 Results from testing

It turned out that the field test could not fit inside the time frame of this project after all. While the technicalities were well planned, UNN did not have the time or resources to start the test before the beginning of June 2006 because of employee vacations.

As previously mentioned, the security personnel had to know how to use the system before the test could be initiated. UNN did not find user manuals to be sufficient for such training, and demanded a course for these users. They pointed out that user manuals are well suited as glossaries, but it can be hard to learn a new system from one, especially for users with low computer skill. While we could have held such a course it could not have started before June, and we had little time to plan one, so we had to cancel it. Since we wouldn't get any results to this report from this test (the report's deadline was June 12, 2006), we had to postpone this testing phase to a later date. This phase will have to be completed before this system will be put into active use. In retrospect we should have set a date for this test at a much earlier stage, and stopped the implementation before all requirements had been completed. Although for further work on the software we have laid a solid foundation, and a field test can be initiated at any time.

Part V

Final results

Chapter 9

Discussion

9.1 Validity

A discussion of validity of the methods used, the decisions made and the results collected, is important to ensure that the results are reasonable and trustworthy. We must ensure that the data has been collected in a way that allows us to draw conclusions that have a reasonable level of confidence. There exists many threats that can affect a project, be it a case study, survey or experiment, both positively and negatively. Wohlin [23] discusses validity threats especially for experiments, but some of these are also threats to other empirical strategies. The following validity discussion is based on his work, although his classifications, separating different kinds of validity threats, are experiment-specific and did not fit our use. The intended scope of the collected data was for use within UNN, and the maximum population they concerned were all the employees at UNN who would use the KAS. This amounts to about 500 people.

The qualitative characteristics of the data collected during the scenario test made them unfit for statistical analysis and subsequent generalization to the whole population. The data was analyzed using experience and knowledge. We have used the data to spawn new functional requirements to the system, but we have not asserted that all the future users will agree with the assumptions made. The field test was supposed to provide a broader user base with the opportunity of stating their opinions about the software system and thus add confidence to the generalization of the requirements. This is one of the reasons why this test should be conducted before the system is operational.

Wohlin points out the importance of making sure you have identified the factors influencing a variable leading to an observed tendency. For example, if the variable was how well the test subjects liked the software system, its layout and intuitiveness may not be the only factors with influence. Other possibilities are computer skill level, experience from other software systems, expectancies about the software, their current state of mind etc. In the previous study [12] some of the test subjects from the scenario test participated in a survey of expectancies about the system by completing a questionnaire. This may have created some bias about the system before the scenario test. We observed that the test subjects were overly positive about the system. Another factor that may have influenced this tendency is the selection of test subjects done by one of our contacts at UNN. We have little knowledge about the criteria for the selection process, it is for example plausible that he selected test subjects he knew would be interested in trying new things and were open to new influences. Thus the feedback could be disproportionately positive. On the other hand, the selection could have been purely random. Being present

during the tests ourselves, as experts of the system, to answer the users questions was probably also an important reason why we received such surprisingly positive feedback.

Wohlin mentions *mono method bias*, which he defines as using a single type of measure or observation. If the method is biased, the results will be misleading. We have indeed used a highly subjective form of observation throughout the tests, which easily could have produced misleading results if we were not aware of this risk. We have supplemented the observations with questionnaires and interviews to ensure a broader base of collected data and at least some objective data collection. As we used several methods of data collection and tried to make the test subjects speak their mind, we do not believe that mono method bias is a valid threat to the results.

Another validity threat is *evaluation apprehension*, which describes people's tendency to try to look better while being evaluated. A person being evaluated while solving a task is likely to do his/her best, which may again produce overly positive results and feedback. It was hard to assess the degree to which this tendency occured, but we believe a few test subjects tried, to some degree, to appear more skilled than they really were.

We tried to make the scenario tests as comfortable as possible for the test subjects. They received help and support from us when needed, and were given coffee and cookies. These factors may have given the test subjects a better than normal impression of the software system resulting in more positive feedback.

The results are also influenced by the level of realism provided by our laboratory setting. The purpose was to make it as realistic as possible, and this was highly dependent on our understanding of the daily routines at the hospital. While the test subjects were comfortable with the tasks given to them, we had little knowledge about their normal work situation. It could have been useful to base the scenarios on direct observation of routines in practice or surveys allowing the users to explain their normal work routines.

Technical difficulties required the use of alternate software[1] during the scenario test which degraded the visual quality of the KAS considerably. This adversely affected the quality of the test and the test results. Readability and/or intuitiveness was crucial to most parts of the test, and was one of the main characteristics being tested.

To summarize, we do trust the data we have collected, but believe that the positivity about the KAS is somewhat exaggerated because of the issues mentioned above. We do not believe that all the data can be safely generalized to the entire population without data from field testing. However, the test subjects were in agreement about most of the results which give the data confidence.

9.2 Time consumption

We did for the most part stick to the initial project plan during the project, with some exceptions which we will describe in the following section.

The scenario test was conducted a week later than originally planned in order to fit the schedules of our contacts at UNN. While this did not impact the project plan, the follow-

[1] The Mozilla Firefox web browser had to be used instead of Microsoft Internet Explorer

ing implementation phase took a lot longer than we planned, and was concluded several weeks overdue. This, combined with UNN's rigorous schedule made the field test impossible in the scope of this project. The documentation phase was also cancelled as it can not be written until the system reaches its working version.

The requirements identified by the scenario test spawned a to-do list, which was weighted by importance and time consumption as described in chapter 7.2.3. The list was sorted by priority, and requirements that were deemed highly time-consuming were given low priority, and vice versa. Some unimportant requirements were done earlier than some more important ones because of low time consumption. While it can be discussed whether many unimportant features are better than a few of medium importance, we did implement all the requirements on the list. This lead to the excessive amount of time used on this phase, and some functionality should probably have been sacrificed to enable the field test to be initiated. We did, however, not know that UNN would be unable to fit the test in the schedule at that point. The field test phase will have to be completed before the system is put into active use, but now it has a good foundation for such future work.

9.3 The scenario test

9.3.1 Our scenarios

The scenarios used for the test were based on stories written by us using the information available about the employees' normal work. The names were hypothetical and departments were randomly chosen because we did not know anything about the test subjects apart from their user level in the system prior to the test. Our contacts felt that this could adversely affect the test results as the test subjects could be confused by anything that differs from their normal work environment. As previously mentioned, we did not have time to alter the scenarios to reflect real names and departments. We emphasized the hypothetical nature of the scenarios at the start of each test session, and the test subjects did not have any objections to this scheme.

9.3.2 Test setup

We concluded that conducting the test in a mainly quantitative way as suggested in ISO 9126 would be too time-consuming, and not dynamic enough. An important goal was to get as much data as possible given the time available with each test subject. We could have used checklists, recorded time used on specific tasks and counted errors made. However, we felt that while this would spawn some quantitative data, it would reduce the amount of qualitative data received. [19] also warn against using checklists in stressful situations, and time was a concern during our tests. We therefore decided not to utilize such methods.

Complementing the observations with a subsequent informal conversation/interview and a questionnaire did significantly help us to clarify issues and increase the amount of data recorded. No questions or preparations for these conversations were planned. They occurred naturally as the tests were concluded. The test subjects seemed to feel more relaxed and comfortable during these conversations, and thus revealed more data and interesting opinions. [19]'s research in evaluating user interfaces also points to the importance of using more than one method of data collection during user observation. Our experience is that direct user observation, while valuable, requires a lot of work from

the observer, and its success is highly dependent on the observer's skill. User observation should also be complemented by other methods of data collection that are not too interfering, like post test interviews and/or questionnaires, for best results and to avoid mono method bias mentioned in the validity discussion.

An extra day for preparation would have been preferable, but was not practically possible. This could have removed the technical problems we encountered during the preparation of the laboratory, which later caused the test subjects some confusion.

9.3.3 Our roles as observers

We decided to observe the test subjects closely and write down as many observations as possible instead of using checklists, thus making the test more dynamic. We could then adjust the time used on the different parts of each test as needed. It is hard to define to which degree an evaluator participates during a test, as the boundaries between observation, partial participation and full participation are rather blurred. If we follow [17]'s chart over degree of participation, we had planned to stay somewhere between participating observer and complete observer during the test. However, we did not manage to keep this role and ended up somewhere between participating observer and full participation.

The main problem was that we had to restrain ourselves not to help the test subjects too much. The test subjects kept giving positive feedback on the system which increased our enthusiasm and relation with the future users. We came to sympathize with their situation, having to form opinions about something they had never seen before. A normal human response to appreciation and flattery is wanting to return the favour in form of help with the tasks we had given them. This became even harder when we gave people with low computer skill tasks they had no possibility to complete without help, and we realized that to get any useful data we would have to guide some of the test subjects through the processes. In these cases it was difficult to stay in the background. The test subjects with better computer skill could be observed according to the plan. These changes would have compromised a rigorous quantitative testing scheme, but in our more flexible scheme we could use the time freed up by helping the test subjects, to concentrate on aspects they had an opinion about. It became evident that the allotted time was not enough for every test subject to find out how they should do their tasks without our help. If we paid close attention, we could register where they encountered difficulties and help them to move on. The alternative was to wait until they solved the problems on their own. While testing the first test subjects we did not know how much time the test scenarios would take to complete, which also may have made us hurry the tests. Although it might have been interesting to know *if* they could find out what to do on their own, it seemed more productive to help them proceed further through the test.

Many of the difficulties encountered by the test subjects would have been eliminated with a minimum of training. The tests would then probably have gone more smoothly and more time could have been used for discussion etc. But with training the users would have learned to traverse areas that are not very intuitive, and how to work around known problems. This may have lead to problems never being exposed, because as long as the test subject knows how to use a system, easily or not, he/she might not think of it as a problem that should be fixed, thus degrading the quality of the results.

All in all, we feel the test was a success and it yielded a lot of useful feedback. This highly qualitative, semi-laboratory, semi-field type of test with ourselves in a partially participating role was perfectly suited to gather the information we needed to root out the worst problems with the software.

9.4 Scenario testing in general

This section discusses scenario testing in general based on our experiences throughout this project.

9.4.1 Experiences

In general

The test subjects did not find any code errors during the test, which indicates either that scenario testing is not useful for bug hunting or that the system contained few errors. We knew, however, that the system had poor exception handling, and our scenarios only tested normal use of the system, which is not suitable for domain testing, stress testing or other means of raising exceptions. Killer soaps (see chapter 4.4) should be more suitable for exposing such errors, as they require the test subjects to enter more extreme values into the system. We could have used more extreme values in our scenarios to stress the software further, but we did not want the stories to loose their credibility.

Pairing each test subjects with one of us was a successful approach, and it was manageable to observe the test subject while taking notes of the important observations. More than one test subject per evaluator would have been too much, and data would have been lost. In addition, notes have a tendency to loose some of their quality as time passes, because they do not contain the context from which they are taken. An alternative is video recording. It requires more resources and infrastructure in the laboratory, and more time to analyze the tapes afterwards, but each evaluator can concentrate on other issues and possibly more test subjects. For these reasons, video recording is preferable if it is readily available, especially if several test subjects are to be observed simultaneously.

Scenario tests are more engaging for testers than methods like checklists or test cases because they include the tester in the problem solving process during the test and encourage them to participate and be creative. If it is based on a believable story it also gives the tester a role that has significance for the outcome of the test, like in a role-playing environment, further engaging the tester.

The complexity of a good scenario test makes it valuable. It requires the tester to think through a problem and find solutions that may require testing several aspects of a system along the way. This provides high-level test results that no other widely used method can do, which is useful for exposing high-level errors, and problems with the system's functionality.

GUI testing

Scenario testing was an excellent method for testing the GUI of the KAS. As the users tried to solve the tasks they were given through the scenarios, they were able to see and learn the dynamics of the user interface and determine how well the GUI let them

interact with the software's functionality. They could then provide us with constructive feedback on the software. The scenarios provided them with a realistic environment for assessing whether they could do what they would need to do in the system, should it be taken into active use. This also highlighted the system's functionality proving that the test is valuable for assessing whether the overall level of quality of a software system, as expected by the specification, is sufficient for the users, or if more work must be done.

Learning

This scenario test can in many ways be compared to an introductory course in use of a software system, as the test subjects definitely learned a lot from trying to traverse our scenarios during the test. It is a well known fact that people learn by doing. Testing software through scenarios involves a combination of creativity, thinking through your tasks trying to figure out what to do and trial-and-error. This is an excellent method of learning a system, much better than learning through a manual.

The scenario test may expose functionality that should be added or improved, but this may not be possible at such a late stage in a system's development. If this is the case the test results can instead be used to identify which parts of the system the users need to be trained to use.

If both a software test and a course for future users will be conducted during the development of a software system, it might be worth considering scenario testing to combine the two, which can save both time and money while yielding interesting test results from new perspectives. It is also a reason to consider user testing if this has not been planned during the development of a software system. Scenario tests should in our opinion be explored further for combinatory educational and testing purposes.

Normal/advanced product use

There are many different ways to use a software system. Some users become advanced users and push the system's capabilities to its limits. It is sometimes interesting to explore both normal and advanced use of a product, and scenario tests should be suitable for both. Scenarios, being based on stories abstracted from technical details and focusing on *what* should be done rather than *how* it should be done, should be equally suitable for identifying the needs of both beginners and advanced users.

Our scenarios tested whether the users were able to complete the tasks they were supposed to during normal work. This is from a software engineering perspective a high-level test which intuitively can be used to test the functional requirements specification for errors and omissions. The purpose of a functional requirements specification is after all, at least ideally, to give the users a system that is usable.

Specification-based testing

Scenario tests can also be used to test if the software contains functionality defined in the functional requirements specification if the scenarios are based on these requirements. It may be hard to evaluate such a test, since many entries in the specification are too low-level to be tested directly through a complex story. This would make the test results ambiguous, as there could be several ways to complete a scenario, and several valid answers or variations of answers. Some requirements are better suited for being tested

in this way, for example GUI-related requirements like "the software must be usable by untrained personnel" or "the system must contain no words or statements that require computer education to understand". Tester could assess such requirements through scenarios test, while more low-level tests could be used for testing low-level requirements.

9.4.2 Problems

Performing scenario tests is not without drawbacks. When planning and conducting scenario tests there are several issues to consider. Those we identified during this project are addressed below.

When involving users in the testing session, problems can occur because of the differences between users, due to education, experience, age, motivation and work-load. This could affect the test results if one wants to generalize them. If the objective is just to test the system for errors and bugs, the background of the users wouldn't matter. However, if the objective is to see how well users understands a new software system, the results will be highly dependent on their computer skill.

Also, if simple functional errors exist in the system during the test, it could make the test a lot less efficient. The system's functions should therefore be tested alone before involving users.

Another thing to have in mind, is that a scenario test is not designed to cover every aspect of the program. Covering all program statements is a complex job, and would be time-consuming both for the testers and the evaluators. Scenario tests only cover the parts of the program that are testers cover by solving the tasks from the scenarios.

[10] mentions that scenario tests should not be conducted at an early stage of development. In [19] the author reports his experiences from conducting user tests on a prototype. His experience is that the prototype makes the test results problematic. Although this was not a scenario test, the results should be applicable to scenario tests as well because of their similarities. Through our experience we agree with these authors: The closer to the full version the system under testing is, the better results you will receive from the testing phase. If much work remains on the system, the results will be less informative as the system would probably contain many errors and unfinished functions.

Any software test should be designed to make it easy to tell whether the program passed or failed the test. Every test result should ideally be easy to evaluate. In *Art of Software Testing* [16], Glen Myers points out that the more complex the test is, the more likely it is that the tester will accept a plausible-looking result as correct. However, we want to point out that scenario test are, as previously mentioned, best suited for qualitative higher level testing. It should not be used for testing simple parts of a system where one can easily assess whether the system passed the test or not with a simpler testing method. However when scenarios are used, there may intuitively be several ways to complete them given their complexity. As long as the tester can easily assess whether the desired result is achieved, this should not be a problem since it is not important how, but only if, the scenario can be completed in a correct manner.

Chapter 10

Conclusion and further work

10.1 Conclusion

Software testing is still an essential part of any software development. As software grows in size, complexity and cost to develop, the methods of testing it must develop as well. The current trend in the industry is utilizing cheap testing methods that do not involve end users. Scenario testing is a testing method that so far lacks good documentation, but has promising characteristics that merit further study. During this project we have studied its usefulness as a high level testing method involving users at a late stage in a software system's development.

By utilizing a qualitative approach, scenario testing can collect unique feedback on how usable a software system is to its users. Users are encouraged to think freely and be creative throughout the test, providing the developers with interesting new perspectives. Our scenario test provided us with important user feedback which spawned many new functional requirements to our software system KAS. These requirements increased the quality of the KAS, by adding new functions, improving old ones and improving the layout of the system. The users' feedback also improved our understanding of how users interacted with the system which enabled us to better satisfy their needs.

As the users explore the software they are testing, trying to accomplish the scenarios' goals, they start learning how the system works. Through trial-and-error and creativity the user quickly learns the important functionality of the software. As a combined method of testing software and teaching its future users how it works, scenario testing has great potential.

A testing method will not be widely used if it is not cost-efficient. Scenario testing used in the manner discussed carries high initial costs and must therefore provide equally rewarding benefits down the road. It was evident during our test that by engaging the tester through a fictional story and making him or her contribute to the test's results, scenario testing added extra value to our software which other methods have not. By the end of the test the users had also received the value of an introductory course in using the system. Such a course can then be omitted or at least scaled down when the system is put into use. Through high-level testing and close observation of the testers, scenario testing also enables the developers to make the system much better suited to the users' needs in the future, making it a valuable long term investment. On the short term, it also provides insight into where effort is needed to teach the users to use the system effectively.

Through this project we have gained experience that can be useful to others conducting scenario tests or doing research in software testing in the future. Based on this we made a list of our recommendations to others conducting scenario testing. This list can be seen in the appendix E.

Based on all qualitative results, feedback and experiences collected throughout this project, we believe scenario testing is a valuable testing method despite its high initial costs. We also feel that our approach with a semi-laboratory, semi-field type of test with ourselves in a partially participating role was a reason for our success in gathering such a large amount of interesting data.

10.2 Further work

During this project we have found several areas that could be interesting for further research in scenario testing. These areas will be described below. As for the KAS, it has to go through an extensive test at some departments of the hospital, before it is taken into use. We will probably do what we can after the conclusion of this project to get a full field test started in order to get the system into use.

One of the most interesting areas that require further research is the use of scenario testing for a combination of educational and testing purposes. It should be interesting to examine scenario testing as an alternative to a combination of, for example, test cases and an introductory course in using a software system.

Another important area of research is to assess how good the cost-benefit ratio of scenario testing is, compared to other testing methods. Such research will probably be necessary before the industry will adopt this method as a viable alternative to others. An experiment could be conducted where the same software is tested simultaneously through both scenario testing and test cases to see the difference in results received and try to assess their value. As scenario testing may reap special benefits on the longer term, some research should also be conducted there.

We have defined scenario testing as not necessarily involving users, but we acknowledge that it probably has its highest strength when users are involved. A future research goal could be to analyze scenario testing without involving users (for example on dedicated testers or developers, which is much cheaper) and compare this to scenario testing involving users. Or compare scenario testing without users to test cases to see the differences.

[7] claim that scenario tests have high error detection power. This is plausible, but during our scenario test we hardly exposed any errors. Error detection may not be one of scenario tests greatest strengths and it may be worth researching this area further. The more extreme versions of scenario testing (see chapter 4.4) should also be further analyzed and their usefulness assessed.

More research on scenario testing in general is necessary to create better documentation. Further research should challenge our experiences to assess whether they are generally useful or only random observations.

Part VI

Appendix

Appendix A

Glossary

Black box testing A testing method where you test the software as if it is a "black box" you do not know the contents of, but you do know which input it accepts and what it is supposed to output according to the functional requirements of the software system. Can be seen as a complementary to white box testing.

Employee Here: Every person that is employed by UNN, either on a permanent or temporary basis. Every employee need access to departments and the respective doors and is therefore in possession of a key card.

External testing The ISO-9126 standard states that software quality characteristics can be measured externally by the extent to which the capability is provided by the system containing the software.

Internal testing The ISO-9126 standard defines the the capabilities of a software system as internal attributes that can be measured through internal metrics.

Key card Is needed to open most doors in the hospital. The key cards have different access levels which give access to different rooms and departments at the hospital. Could be either a time-limited (temporary) or permanent(ID) card.

Orderer An orderer is any person who orders a key card to someone through our software system, KAS, or by means of the existing paper-based system.

Orders Is submitted by an orderer. The order is either for a time-limited key card or an ID card. The security personnel must acknowledge the order before a key card is issued.

Quality Quality is the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs (ISO9126).

Quality attribute A mapping of a system's functionality into a software structure that is meaningful in measuring quality. It is a categorization of software design desicions that affect a certain type of quality.

Quality factor See quality attribute.

Scenario A scenario is a hypothetical story, used to help a person think through a complex problem or system. Scenarios aim at describing realistic situations with tasks that reflect real use.

Scenario testing Scenario testing is a software testing method where the tests are defined through tasks described by fictional stories. The stories describe what the tester should do, but not how to accomplish it. The stories should be credible, and encourage the tester to be creative and explore the system's functionality.

Security personnel Accepts or rejects orders for key cards. When accepting an order they issue a key card to the person in question and enters its respective access level into the access control system. These are the most important users of the KAS.

Stakeholder A stakeholder is any person with an interest in the software system, be it a user, investor, corporate head or developer.

Temporary card See time-limited card.

Test case A set of conditions or variables under which a tester will determine if a requirement upon an application is partially or fully satisfied. It may take many test cases to determine that a requirement is fully satisfied. In order to fully test that all the requirements of an application are met, there must be at least one test case for each requirement unless a requirement has sub requirements. In that situation, each sub requirement must have at least one test case (Wikipedia).

Time-limited card Time limited card in this project is key cards where the employee holding it is hired on a temporary contract. These cards are also referred to as temporary cards in this report.

Usability Usability is the most important quality attribute in this project. It concerns how easy it is for the user to accomplish a desired task and the kind of support the system provides.

Users An individual person that uses the software product to perform a specific function. In this project it means anybody that will use the KAS to complete a task, either an orderer, the security personnel or the administrators.

White box testing A testing method which can be seen as a complement to black box testing. Here you have knowledge of the code and structure of the program you are testing, which you use to execute certain functionality and check if the program performs correctly.

Appendix B

Scenarios

In this appendix the scenario tasks we made for the users are presented. There were two scenario tasks for each user group, but only one of each user group are presented here, as the only difference between the two are different example data like names, dates etc. The tasks are presented in norwegian, as they were originally presented to the users, and ordered by user group.

Time-limited card orderer

Oppgave 1:
Du har i oppdrag å bestille ekstravaktkort til Kåre Hansen, født 20.09.1970. Han skal begynne den 18. februar klokken 09:30 og avslutter jobben den 24. februar klokka 23:00. Kåre skal være vikar på Akuttmottak med undergruppe sykepleier. Ellers skal han ha standard tøy og må ha adgang til medisinrom ved avdelingen. Utfør bestillingen til Kåre før du leser videre.

Oppgave 2:
Du har fått et nytt oppdrag, denne gangen til Mette Paulsen. Mette er født 04.02.74 og er leid inn fra Manpower. Hun skal ha vakt fra 03.03.06 kl 16:00 til 09.03.06 16:00. Hun jobber på Administrasjon avdelingen, undergruppe "administrasjon og andre avdelinger med kontorer der". Hun må ha frakk men trenger ikke tilgang til medisinrom ved avdelingen. I tillegg vil du spørre vekterne når kortet er klart til henting. Send av gårde bestillingen før du går videre.

Oppgave 3:
Med en gang du har sendt av gårde den siste bestillingen får du beskjed om at det var en liten feil i opplysningene du hadde fått om Mette. Hun skal jobbe til 10.03.06 16:00 isteden for 09.03.06. Så dette må du rette opp før du går videre.

Oppgave 4:
Det viser seg i midlertidig at den første bestillingen du sendte har blitt avvist av vekter. Du må rette opp bestillingen i henhold til tilbakemeldingen som vekteren har gitt.

ID card orderer

Oppgave 1:
Marita Johansen er en ny fast ansatt og må ha et Id-kort. Hun begynner i jobben 1.

mars og er ansatt på ubestemt tid, slik at hun ikke skal levere kortet tilbake før nærmere beskjed er gitt. Marita er født 1.juni 1980 og skal jobbe ved KIR POL avdelingen. Hun trenger standard tøy og er ansatt som kirurg. Utfør denne bestillingen før du går videre.

Oppgave 2:
Det er enda en ansatt som trenger et nytt id-kort. Gunnar Jakobsen er ansatt på NEVRO LAB avdelingen som labansvarlig. Gunnar er 43 år gammel og er født 02.01.63. Han er ansatt ved UNN og ikke av noen innleide vikarbyråer. Saken er at han har hatt et id-kort men har mistet dette og derfor trenger han et nytt. Han mistenker at han har blitt frastjålet kortet. Han trenger det nye kortet fra og med 17. februar og skal beholde det til og med 20. juni. Han må ha standard tøy. Bestill kortet til Gunnar før du går videre til oppgave 3.

Oppgave 3:
Det viser seg at opplysningene i Marita Johansens bestilling var noe feil. Det riktige er at hun starter i jobben først en uke senere, nemlig 8. mars. Rett opp dette før du går videre til neste oppgave.

Oppgave 4:
Hvis statusen på bestillingen til Gunnar Jakobsen du gjorde i oppgave 2 sjekkes, vil du oppdage at bestillingen har blitt avvist. Din oppgave her er å rette opp dette i henhold til gitt tilbakemelding slik at bestillingen blir godkjent.

Oppgave 5:
I disse navneendrings-tider så har Marita Johansen funnet ut at hun skal legge til et mellomnavn. Hennes fulle og hele navn vil etter denne endringen være Marita Cathrine Johansen. Som følge av hennes navneendring må også navnet hennes i bestillingssystemet forandres. Din oppgave er å melde inn denne personalendringen.

Security personnel

Oppgave 1:
Dagens gjøremål står for tur, og det som først må gjøres er å sjekke om det er noen bestillinger som skal programmeres i dag. Det viser seg at det ligger inne en bestilling på ekstravaktkort til Bård Andersen som må programmeres. Kortnummeret som skal leveres ut er 2356. Gjør dette før du går videre.

Oppgave 2:
Du har fått inn en bestilling på ark som du først må føre inn i systemet før du kan programmere denne. Det er et id-kort som skal bestilles, Bård Andersen skal nå ansettes fast og trenger et idkort. Han skal ansettes på HUD POL. Det er markert på arket at han trenger frakk og er ansatt som lege. Bård trenger kortet fra og med 18 februar. Han er ansatt på ubestemt tid, og derfor skal det ikke settes noe dato som kortet må leveres tilbake. Før inn denne bestillingen i systemet før du går videre til oppgave 3.

Oppgave 3:
Når så bestillingen fra oppgave 2 er lagt inn i systemet, kan du programmere og godkjenne bestillingen. Det er veldig viktig å få med seg informasjonen som står i "ekstra informasjon"-feltet. Dette feltet spesifiserer alt som må gjøres angående bestillingen. Kortnummeret som skal legges inn er 4783 og adgangsnivået skal være det vanlige nivået

på avdelingen. Dette må gjøres før du går videre til oppgave 4.

Oppgave 4:
Det ligger flere bestillinger i listen 'programmere bestillinger'. Denne gangen er nok et id-kort som skal utleveres, men nå er det Sigrid Persen som har mistet sitt id-kort og derfor trenger hun nytt. Utfør denne bestillingen, og pass på at du får med deg alt av viktig informasjon. Kortnummeret på det nye kortet er 9582 og adgangsnivået skal være det samme som på det kortet som er mistet. Når du har utført denne bestillingen vil du få beskjed om hva som skal gjøres med det kortet som er mistet. Gjør dette før du begynner på oppgave 5.

Oppgave 5:
Den siste bestillingen som skal programmeres i dag er en bestilling på ekstravaktkort til Marit Gulbrandsen. Denne bestillingen har start og slutt-dato før dagens dato. Du må avvise denne bestillingen med melding til bestilleren om hvorfor den ikke kan godkjennes.

Oppgave 6:
Oversikten over vakter som burde avsluttes innen 24 timer viser at det er en vakt tilhørende Birgit Jakobsen som må avsluttes. Dette betyr at personens vakt går eller har gått ut, og derfor må adgangsnivået på kortet deaktiveres slik at det ikke kan brukes mer. Din oppgave er å sørge for at vakten avsluttes.

Oppgave 7:
Birgit Jakobsen har nå levert sitt kort i skranken og du må derfor registrere dette kortet (3765) som innlevert.

Oppgave 8:
Det er bestilt endring på personalia til Marit Gulbrandsen. Hun har skiftet etternavn, og i forbindelse med dette har personalavdelingen bestilt endring på personalia. Gå til personalendringssiden og godkjenn denne endringen.

Appendix C

Test results

C.1 Observations from the scenario test

Important non-functional data collected during the observation phase is presented in the tables below. These tables show the negative and positive observations of the different user groups. Only the most important observations are included, and the majority repeated themselves throughout the test.

TIME-LIMITED CARD ORDERERS AND DEPARTMENT HEADS	
Negative	**Positive**
• Knowing when to add a new user is not self-explanatory • Entering dates is hard, although the error message is intuitive • Timestamps should not contain colons • Not all departments have archives and date of birth is therefore a practical problem when registering a new user • "Undergruppe" (The requested access level for a key card) is an unknown or ambiguous term to many card orderers • It takes more time than before to fill out orders • Some had low computer skill, mediocre in general	• A replacement for the old system is highly appreciated • Searching for, selecting and adding users is easy • The feedback function from the security personnel is very good • Example in-data beneath makes the data entry fields intuitive • 3 departments have volunteered for further testing because the system seems much better than the old one • The system is easy to learn

Table C.1: Qualitative results collected through observation part 1

ID CARD ORDERERS	
Negative	**Positive**
• The scenarios contain quite an amount of data which takes time to process • Auto-complete (in Firefox) is confusing • Filling out a new, complete order if someone looses his/her card does not seem logical • In general, the functions concerning changes in a persons personalia and/or duty details are complex and require training	• Placing orders is easy • Everything looks pretty, intuitive and well-arranged • The system does what it intuitively should • The system removes problems which we have had to work around with the existing system • Good computer skill in general

Table C.2: Qualitative results collected through observation part 2

SECURITY PERSONNEL	
Negative	**Positive**
• The calendar is small and one might hit the wrong date • One of the test subjects had very low computer skill, but the rest of them had good computer skill in general	• The introduction of the new system is positive and will ease the work load • The system's layout is easy to understand • The system eliminates errors that now occur due to the large number of persons involved in an order's life cycle which creates many sources of errors • Much easier to use than anticipated

Table C.3: Qualitative results collected through observation part 3

C.2 Feedback from the questionnaire in the scenario test

All users in the scenario test session were given the opportunity to provide feedback by filling out a questionnaire. In this appendix we present all the questions that were on this form, and the participating users answers. The questions and answers are translated into English. The answers are divided into the different user groups in the system; Temporary (time-limited cards), ID cards (personnel department), security and administrators.

Of all 18 participating users from the scenario test session, we received 13 schemas with feedbacks. 6 of those were from time-limited, 4 from personnel department, 1 from security, and 2 from the administrators.

The questions from this questionnaire are presented here:
Question 1: What do you think was good/positive in the system?
Question 2: Did something surprise you in the system? In that case, what?
Question 3: Were there any functions/fields from todays paper system you missed? If so, which?
Question 4: Were there any functions/fields you felt were superfluous? If that is the case, which?
Question 5: Were there something that you didn't experience as user-friendly? In that case, what?
Question 6: How can the system become more user-friendly?
Question 7: Did you find any errors in the system?
Question 8: What do you think was bad/negative in the system?
Question 9: Do you think this system can replace todays paper system? If not, what must be done?
Question 10: Other comments?

The answers to these questions can be seen in tables C.4-C.6. These tables show each user's answer to each of these questions. The user type column denotes which type of user has answered the question, and the number behind is to distinguish between users in the same user group.

User type	Question 1	Question 2	Question 3
Temporary 1	Simple and well-presented to make orders via computer		
Temporary 2	Easy and well-presented		
Temporary 3	Simple layout without much unnecessary text		
Temporary 4	Simple and easy-to-grasp	No	No
Temporary 5	Good, a few operations too much	Could have been more pre-filled fields, example: choice of department when the name is recognized	
Temporary 6	User-friendly, simple and easy to navigate	No	No
ID card 1	I think the system is easy to use	To edit an order, after the orders are submitted	
ID card 2	Ok, looks like the paper-system we use today	It is nice to be able to see what has been ordered, and what has a status of "waiting"	
ID card 3	Easy to operate		No
ID card 4	I think it is well-presented and easy to use	To be able to edit orders, plus to be able to check that we have ordered a card when the person shows up for work	Fix the birth number to be on one line
Security	Easy-to-grasp	No	Maybe another sound/spelling on some words
Admin 1	Ok menu system, sorting		
Admin 2	Simple operations, well-presented	Not really	No, nothing

Table C.4: Feedback from questionnaire, question 1-3

User type	Question 4	Question 5	Question 6
Temporary 1		Editing an order, everything must be filled out all over again	
Temporary 2		The calendar	
Temporary 3		Timestamp,(remove ":" in the time), date should appear in a new field after choosing it. Filling out "tøygruppe"+"adgang med.rom", the picture "jumps" between each choice. This is irritating and unpleasant, and one have to search to find out where one are	
Temporary 4	No	If changing an order, had to fill out everything all over again	
Temporary 5	Date, more tolerance, i.e. desire pre-filled timestamps	OK	Should be able to send more orders for the same person without searching after the person again
Temporary 6	No	When editing an order, everything must be filled out again	More pre-filled fields, and a default timestamp
ID card 1	I think the system is easy to use	To edit an order, after the orders are submitted	
ID card 2			
ID card 3			
ID card 4			
Security	No	No	
Admin 1		To return from a picture to the previous, in a natural consistency without going all the way back to start	
Admin 2	No	Some inexplicable texts due to links and on the command-button, more understandable texts on boxes and buttons is required	As explained earlier, colorful/bold text on the user data

Table C.5: Feedback from questionnaire, question 4-6

User type	Question 7	Question 8	Question 9	Question 10
Temporary 1				
Temporary 2			Yes	
Temporary 3		See question 5	Yes	
Temporary 4		Dropdown list on time	Yes	
Temporary 5			Gladly even more simpler/faster if possible? Must be available easily over the intranet	
Temporary 6	No	This looks very good!	Oh yes, do it as soon as possible	Make sure a shortcut to this system is placed on the start page of the intranet, in the menu on the left side
ID card 1			I think so	
ID card 2			Ok, I think you guys have been good!	
ID card 3		Everything OK	Yes, without a doubt	Nice!
ID card 4	Yes, date of birth stops with day, month, year	Very easy-to-grasp	Yes	
Security			Yes!	Too little time to evaluate this properly
Admin 1	Heading text, "Id-kort", in "ekstravakt" orders	Nothing special was bad	Yes	
Admin 2	Yes	Some other colors on some fields with data	Yes	Nice work!

Table C.6: Feedback from questionnaire, question 7-10

Appendix D

Functional requirements from the scenario test

All the functional requirements discovered through the scenario test are presented here. Each requirement received a high/medium/low rating on its importance, and its estimated time needed to implement. From this weighting the requirements are ranged from highest to lowest importance. "<>" means that the change has been implemented, "xx" means that it has been decided not to be implemented.

Example:
(Importance, Time to implement)Functional requirement

Most important:

<>- (H, L)Noe må gjøres med undergruppefeltet for å gjøre det mer intuitivt, for eksempel endre navnet til gruppe?
<>- (H, L)Tekstfeil i overskriften på bestillingsbekreftelse når man skal bekrefte ekstravakt og personen ikke har noe kort fra før, står id-kort i steden for ekstravakt
<>- (H, L)Bestillingssiden (både idkort og ekstra) burde sjekke at datoen ikke er "utgått" på bestillingen.
<>- (H, L)Vekter burde ikke ha tilgang til å slette/endre noen loggoppføringer
<>- (H, L) Burde stå på ekstravaktbestillingssiden at ekstravaktkort bare gjelder i maks 3 måneder
<>- (H, L) Vekter burde få beskjed når han godkjenner bestillinga om at han må gå å skifte på adgangskontrollen
<>- (H, L) Fikse informasjon til brukerne på startsiden
<>- (H, M)Bekreftelsesboks på programmering av bestillinger, må få beskjed om å programmere på det andre systemet før bestillingen bekreftes, passe på å få med navn, avdeling, stilling, fra og til dato og annen viktig info som nøkkelkortnummer
<>- (H, M) Redigering av bestilling, brukere liker ikke at man må fylle ut alt på nytt, dette burde ordnes.
<>- (H, M) Gjøre noe med klokkeslett på bestilling. Vi innfører klokkeslett uten skilletegn først, fordi da kan de som ønsker presise klokkeslett bruke det, mens de andre kan skrive hele timer eller bruke standardforslaget
<>- (H, M) Admin vil ha oversikt over hvem som er bestillere. De vil se hvem som kan bestille på en gitt avdeling, og hvem som har de forskjellige tilgangsnivåene.
<>- (H, M)Vi må ha med info om hvem som har hatt kort tidligere i alle perioder.
<>- (H, M)Vekterne burde få opp dagens gjøremål som startside der man kan "hoppe" til

de respektive sidene, bare at de er koblet til dagens gjøremål, mens de andre menyvalgene kommer på venstresiden.
<>- (H, H) Autopostback på kalenderen er ikke så bra, personen blir litt satt ut av at den hopper opp til begynnelsen igjen, og bruker litt tid på å skjønne hva som skjedde
<>- (H, H) Stort sett var det etterlyst OK/Slett bekreftelse på alt man foretar seg

Medium: (noen lave med her fordi de tar kort tid)

<>- (M, L) Kalendern må fikses slik at den ser bedre ut
<>- (M, L) Dagens dato burde vært markert i kalenderen, er vel uansett ikke dumt med en default dato
<>- (M, L) Når en måned velges i jobbfradato burde jobbtildato skifte automatisk til samme måned
<>- (M, L) Forskjell mellom viktig info på detaljer om bestilling, selve feltet som beskriver hva det er som navn, avdeling osv er mindre viktig, mens selve navnet, avdelinga og resten av inndataen er viktig og burde dermed utheves/annen farge for å se ting bedre. Hvem som har bestilt en bestilling er mindre viktig og kan feks skrives med mindre skriftstørrelse
<>- (M, L) Tilbakemeldingsteksten burde utheves/farges, dette er jo en viktig tekst. Samme med merknadsteksten som bestiller skriver til vekter.
<>- (M, L) Hadde vært bra hvis "ingen-treff" knappen ble litt forandret, feks med en overskrift på hvor mange treff og eventuelt en "legg til" knapp
<>- (M, L) Admin firma: en tilbakeknapp fra "ny avdeling" til admin firma, ikke bare helt tilbake til startsiden
<>- (M, L) Avslutt-vakt knappen burde ikke hete avslutt men avslutt vakt og kanskje en avbryt knapp i tillegg
<>- (M, L) Bestillingsbekreftelse, forandre på hvilken type informasjon som er viktig
<>- (L, L) forslag om å skrive DDMMÅÅÅÅ i fødselsdagfeltene fra før av for å gjøre det helt idiotsikkert
<>- (M, L) Forslag om å kunne sende flere bestillinger til en person uten å søke opp samme personen på nytt igjen.
<>- (M, L) Standard undergruppe burde velges automatisk, ialle fall der det kun finnes 1, impliserer at avdeling også må velges automatisk
<>- (M, L) Endring av tilbakemeldingsmelding på lever inn kort. Vekterne må få samme beskjed som under bekreftelse av bestilling, at ting må gjøres på det andre systemet.
<>- (M, L) Forslag om passordregler, lengde, små/store bokstaver, inneholde tall
xx- (M, L) Bestillingsbekreftelse: burde være mulig å endre sluttdato før bestillingen godkjennes
<>- (L, L) Mange som synes det var mer logisk å skrive etternavn før fornavn på søkesida, burde dette endres?
<>- (L, L) Et praktisk problem er at folk må bytte vakter på kort varsel, og da er det ikke tid til å bestille, trenger mulighet til å bestille flere adganger samtidig (i tilfelle de trengs). Det kan de forsåvidt føre opp i merknadsfeltet. Skriv heller på bestillingssia eller en plass at man må skrive sånt i merknadsfeltet
<>- (L, L)Makes no sense å endre gamle loggoppføringer, slik at admin trenger egentlig heller ikke denne muligheten, men slette må de kunne
<>- (L, L) Lage en slags popup eller noe lignende som viser tilbakemeldingen når man ser på bestillingslisten uten å måtte gå inn i rediger bestilling.
xx- (L, L)Adgangsnivå burde kanskje sorteres alfabetisk i dropdownlista
xx- (L, L) Skifting av brukernavn? Kanskje bare admin som burde kunne gjøre dette (er

ikke så viktig egentlig)
xx- Passord case sensitiv

Lowest:

<>- (L, M) Tøygruppe eksisterer ikke mer, men de forholder seg til det, vekterne er oppmerksomme på dette, de andre vet ikke.skal fjernes
<>- (M, M) En del lurer på om man må til startsiden for å bestille nye kort, menyen burde kanskje vises på venstresiden (i allefall de viktigste/mest brukte alternativene)
<>- (M, M) Personalendringer må gjelde avdeling og stilling, de skal kun godkjennes, trenger ikke 2-veis kommunikasjon, Må ha merknadsfelt, Se personalendringsskjema.
<>- (M, M) registrere bestiller, var litt tungvindt at de bare kunne gjøre en operasjon om gangen før personen må søkes opp på nytt
<>- (M, M) Burde få sett en liste over bestilte personalendringer, i alle fall en bekreftelsesside før bestillingen blir godtatt, det var for lett å gjøre feil
xx- (M, H) Felt takler ikke "rare" tegn som % & " osv
xx- (M, H) Brukerne brukte tab og enter mer enn vi hadde trodd (ta hensyn til highlight rekkefølge, både på knappan og feltan)

Appendix E

Recommendations for scenario tests

Planning

Costs

Start by considering the budget of the test, and the desired results. Used by developers or dedicated testers as an alternative to test cases, the cost of the scenario test will be marginally higher than test cases because writing the scenarios is more time-consuming. The results should differ from test case results in being more usability and functionality oriented. If the test will involve future users, the initial cost in both time and money will be relatively high but the long term benefits potentially great because of exposed errors and omissions that only users can effectively find, and the valuable training the users will receive by exploring the system. The parts of the functional requirements specification covered by the scenario test will get an overhaul before the software is shipped, allowing for last minute changes or additions that may cost more money to fix after the product is finished. If changes are not possible at such a late stage in development, the test can be used to highlight which parts of the system the users need to be taught how to use effectively.

Location

For a user test we recommend using a "laboratory", a place where the test environment can be controlled. Choose a location that minimizes the amount of travel for the users. If available, use a laboratory with video recording capabilities. While it could be tempting to test the software in its real environment simultaneously, we do not recommended it because unforeseen technical difficulties could adversely affect the results from the scenario test. Instead, recreate the development environment to minimize potential problems while making it seem as realistic as possible to the users. You will then have the advantages of a field test's realism, while being able to control the environment as in a laboratory test.

Participants

In general, select test subjects randomly. In our experience users with low computer skill can produce unique feedback on usability related issues, while users with high computer skill like the software better and yield the most positive feedback. The latter can also evaluate advanced product use, and their test results will resemble those produced by dedicated testers. A mix of user types is therefore preferable. When choosing the amount

of users involved, it is better to have few users giving a thorough evaluation, than many users that do not have enough time to produce feedback they have thought through.

What to test

If users are involved, choose to test the parts of the system they will use often, and/or functionality essential to the software's usefulness. Users are good at assessing whether the software is useful to them or not (usability and performance), but bad at assessing whether it is secure or modifiable (portability). Known disturbing bugs should be removed from the system before a scenario test to avoid confusion.

Preparation

Creating scenarios

Use requirements from the functional specification to create the scenarios. Avoid basing them on uncomplicated requirements that can be tested by a cheaper testing method. Requirements like "the user must be able to intuitively register a complete order" are good for scenario testing, while "the sorting algorithm must be faster than O (log N)" or "there must be links page" are not. Requirements concerning the system's GUI, usability or functionality are suitable. Try to create goals that makes the tester test simpler requirements along the way. The stories should be complex, but not to such an extent that they loose credibility or become difficult to understand. They should not be written by the same person that will use them because they will loose credibility and become less interesting. The quality of the results will be lower because one of the scenario test's strengths, is the creativity involved in figuring out how to reach it's goals. This should not be thought through beforehand.

To increase the test's error detection power, more extreme in-data for the software can be entered where applicable. Not unrealistically extreme as the story will loose its credibility and realism. A killer soap can be used as a second scenario test, primarily by the developers to reduce costs, as an advanced domain test.

Data collection

When testing with users we recommend using observation, either directly by one evaluator per test subject or through video recording. Video requires much time after the test to watch through to register the important data, but captures more data. If more than one user per evaluator will conduct testing simultaneously, video recording is a necessity. Taking notes is cheaper, but the results must be evaluated immediately after the test or important contextual information will be lost. This method is also highly dependent on the observers skill in identifying important data as they are revealed and being objective. The observer should also have extensive knowledge of the system being tested. Observation should be supplemented by a post test interview and/or a questionnaire to create more credible data. A questionnaire must not be too large or require much insight into the software. It should only capture data you want to record from every test subject and make it possible for the test subjects to convey any final thoughts before the test is concluded. We do not recommend using checklists during the test unless quantitative data that can not be collected afterwards are essential.

Developers or dedicated testers can register their experiences themselves while conducting the scenario test. Checklists can then be useful for capturing important data, as the tester can follow its structure while testing.

Schedule

When involving users, a detailed schedule should be created. First, a reasonable amount of time for completing the scenarios must be assessed. The users must also have time to fill out any questionnaires, possibly explore the software on his/her own and complete any post test interview. Time to ready the lab for the next batch of users should also be embedded in the schedule.

Testing

Try to avoid participating too much while observing users, let them be creative and find out on their own how to complete the scenarios. It can, however, be necessary to interfere if the users are stuck. While it can be interesting to see whether the users can traverse any obstacles on their own, it is not productive to let them be stuck for a lengthy period of time.

Important data to look for while observing include errors, new functional requirements, parts of the system that is troublesome to the users and what the users like and don't like. Troublesome functionality is important not only for improving the GUI, but can be used to determine where to focus user training at a later time. It may be prudent to record contact information if any issues emerge later that need to be clarified.

It is important to be open with users and explain that it is not they who are being tested, but the system. They should not feel any pressure to perform and appear talented. Otherwise the results will not be realistic and important feedback will be omitted.

Data analysis

If data has been collected through observation and notes, they must be analyzed as quickly as possible after the test has been completed, not more than a day or two later. Identify the important data and register them in a way that can easily be used later and that does not become ambiguous as time passes.

Documentation

Document any choices made during the test and if they were a success or not. Any deviations from the test plan should also be registered with an explanation.

Appendix F

User manuals

We have written four user manuals to cover the different types of users to the KAS. The most extensive one is intended for the security personnel and is presented here in its original form. It is intended for a norwegian audience and has not been translated. The user manual has its own page numbering and layout.

Brukermanual

Vekter

Innholdsfortegnelse

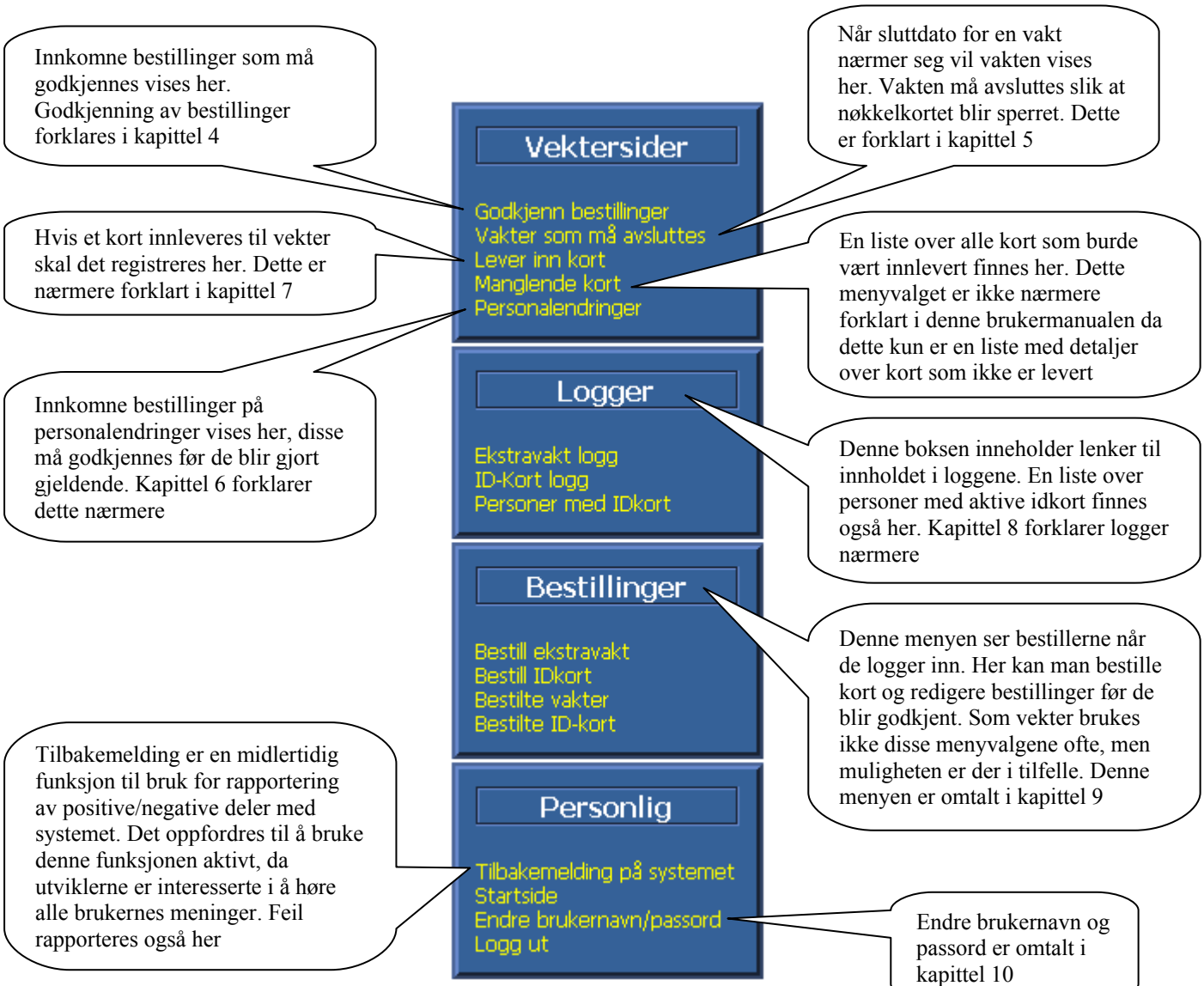
1 Innledning

Denne brukermanualen er en hjelp til bruk av administrasjonssystemet for nøkkelkort ved UNN. Systemet er konstruert med tanke på brukervennlighet og at ting skal være selvforklarende, og denne manualen er derfor ment som et supplement og vil gi en kort innføring i hvordan systemet skal brukes.

Systemet er delt opp i flere deler, og tilgangen og funksjonene man har i systemet avhenger av hvilken type bruker man er (ekstravakt bestiller, personalavdeling, vekter osv). Denne versjonen av manualen er beregnet på ***vektere***.

2 Hovedmenyen

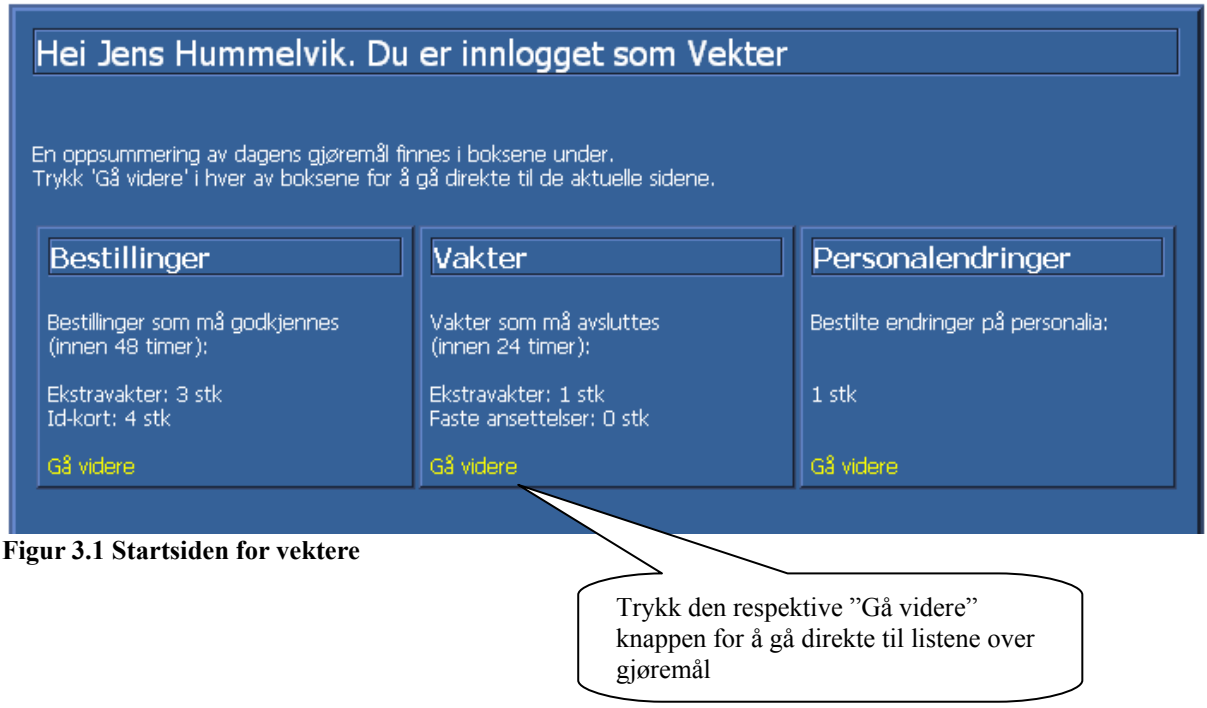
Hovedmenyen for vektere vises i figur 2.1. Den innholder alle funksjonene og mulighetene vekterne har i systemet. Denne vises på venstre side uansett hvor i systemet man er. De viktigste menyvalgene er forklart videre.



Figur 2.1 Hovedmeny

3 Startsiden

Startsiden for vektere vises i figur 3.1. Dette er siden som vises direkte etter innlogging i systemet. Her vises en oversikt over dagens gjøremål. Dette gjelder hovedsakelig innkomne bestillinger som må godkjennes(se kap.4) og vakter der adgang skal sperres(se kap.5). Bestillinger på personalendringer vises også her(se kap.6). Disse gjøremålene finnes også ved hjelp av hovedmenyen på venstre side, men startsiden er ment som en oversiktlig oppsummering slik at man raskt ser hva som må gjøres hver dag.



Figur 3.1 Startsiden for vektere

4 Godkjenne bestillinger

I figur 4.1 vises et eksempel på en liste over innkomne bestillinger, både ekstravakter og Idkort. Bare bestillinger som starter maksimum 48 timer fra nåværende tidspunkt vises her. Det er mulig å legge inn bestillinger lang tid i forveien, men de vil først vises i denne listen når datoen nærmer seg. Begge listene sorteres ved å trykke på overskriftene over ekstravaktbestillingene.

Figur 4.1 Liste over bestillinger

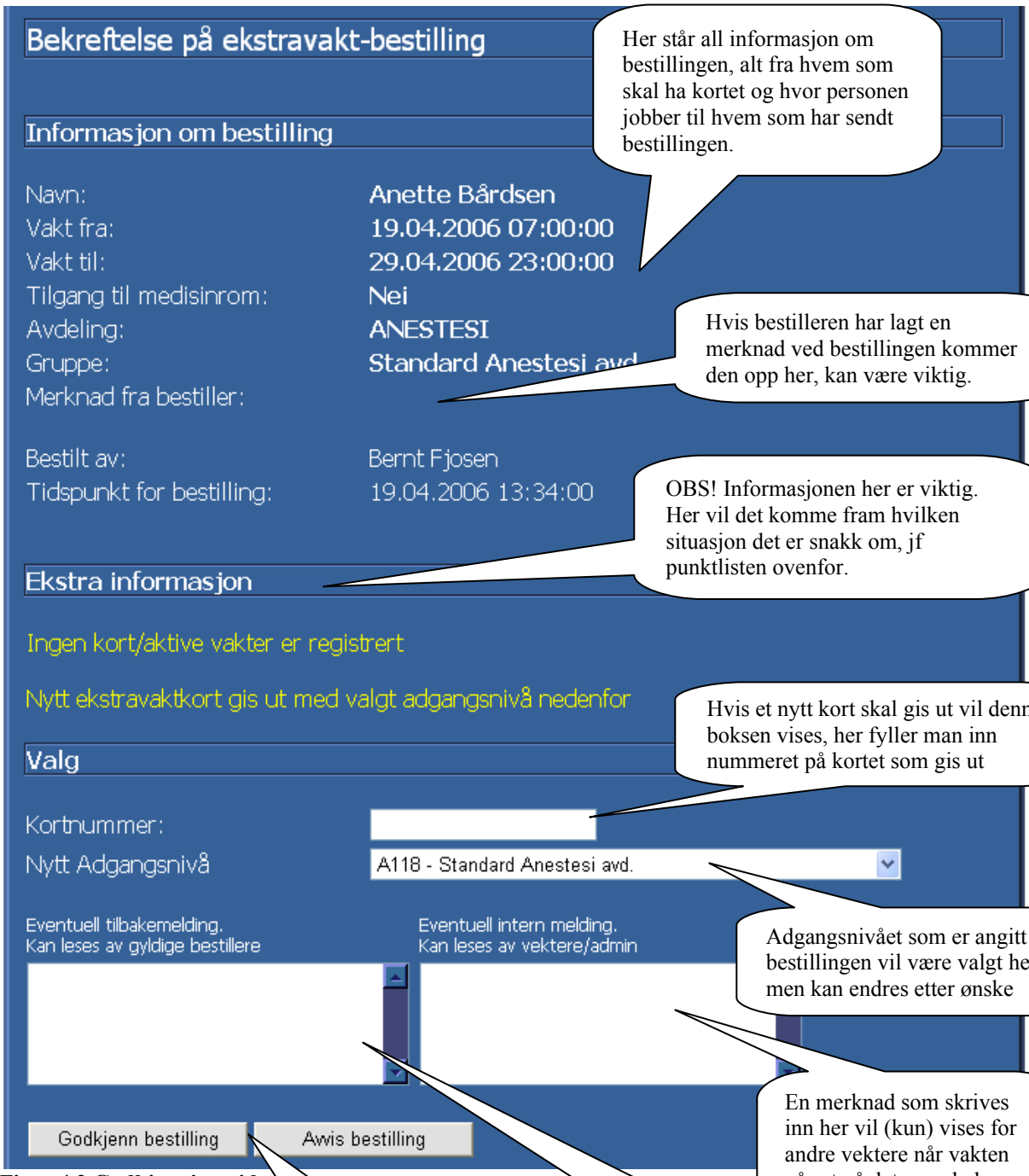
Når bestilling er valgt vises godkjenningssiden som vist i figur 4.2. Denne vil se litt forskjellig ut avhengig av hvilken type kort som er bestilt, om personen har registrerte kort/vakter fra før og om personen har krysset av for mistet kort på bestillingen. De forskjellige mulighetene systemet tar hensyn til er som følger:

Ekstravaktbestillinger:

- Personen har ingen kort fra før og har bestilt en ekstravakt. Ekstravaktkort gis ut
- Personen har et idkort fra før men har bestilt en ny ekstravakt. Den nye ekstravakten aktiveres på idkortet.
- Personen har et idkort som brukes på en ekstravakt, men har bestilt en ny ekstravakt. Den gamle ekstravakten nullstilles og den nye ekstravakten aktiveres på idkortet.
- Personen har et ekstravaktkort fra før, men har bestilt ny ekstravakt. Den gamle ekstravakten nullstilles og den nye aktiveres på ekstravaktkortet.
- Personen har flere kort fra før, nytt kort kan ikke gis ut før et av de andre meldes savnet eller leveres inn.

Idkortbestillinger:

- Personen har ingen kort fra før og har bestilt et idkort. Idkort gis ut.
- Personen har ekstravaktkort, men trenger et permanent idkort. Idkort gis ut, og ekstravakten nullstilles når ekstravaktkortet leveres inn.
- Personen har bestilt et idkort og merket av for ”mistet kort”, men ingen idkort er registrert i systemet på denne personen. Nytt idkort gis ut.
- Personen har mistet idkort, gammelt idkort sperres og nytt idkort gis ut.
- Personen har mistet idkort som brukes på ekstravakt. Den gamle ekstravakten nullstilles og idkortet sperres, nytt idkort gis ut.
- Personen har allerede et idkort, men prøver å bestille nytt uten å merke av for mistet kort. Bestillingen må omgjøres til å gjelde ’mistet kort’.
- Personen har flere kort fra før, nytt kort kan ikke gis ut før et av de andre meldes savnet eller leveres inn.



Figur 4.2 Godkjenningssiden

Her står all informasjon om bestillingen, alt fra hvem som skal ha kortet og hvor personen jobber til hvem som har sendt bestillingen.

Hvis bestilleren har lagt en merknad ved bestillingen kommer den opp her, kan være viktig.

OBS! Informasjonen her er viktig. Her vil det komme fram hvilken situasjon det er snakk om, jf punktlisten ovenfor.

Hvis et nytt kort skal gis ut vil denne boksen vises, her fyller man inn nummeret på kortet som gis ut

Adgangsnivået som er angitt i bestillingen vil være valgt her, men kan endres etter ønske

En merknad som skrives inn her vil (kun) vises for andre vektere når vakten går ut på dato og skal avsluttes

Her velger man om bestillingen skal godkjennes eller avvises

Hvis en tilbakemelding skrives her, kan den sees av personen som la inn bestillingen (og andre med samme tilgang). Dette kan være særlig nyttig for å oppgi en grunn til at en bestilling er avvist. Hvis det står en tilbakemelding her allerede er det fordi en annen vekter har avvist en tidligere versjon av samme bestillingen. Husk i så fall å fjerne denne.

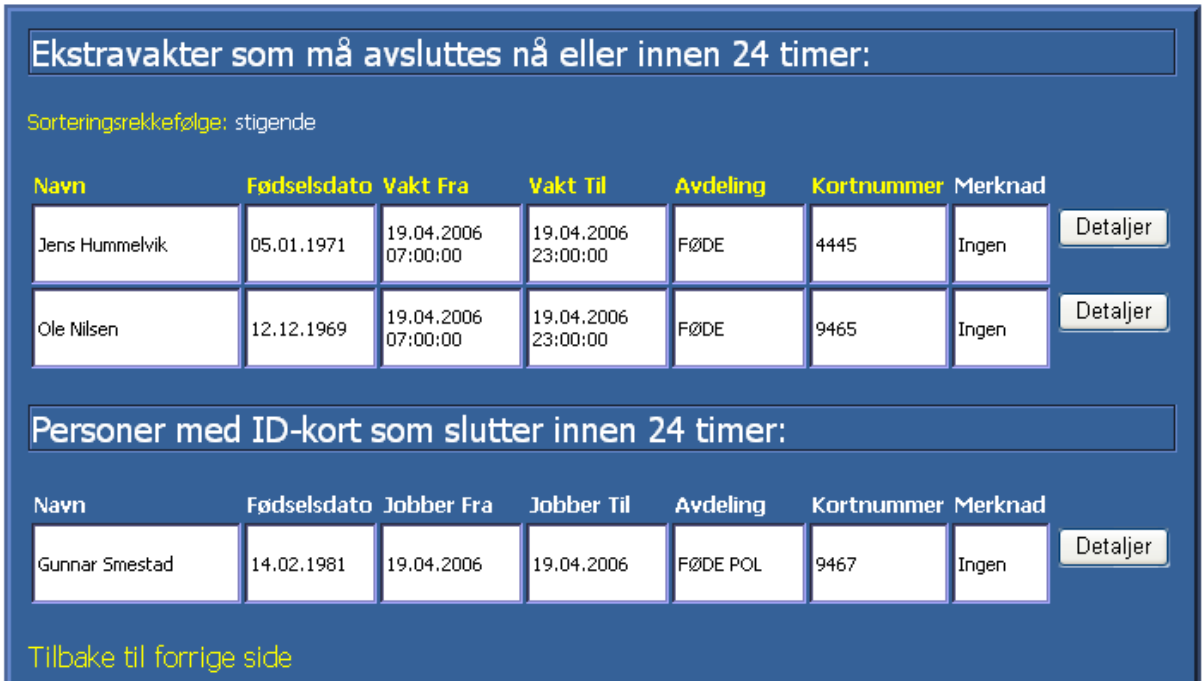
Når ”Godkjenn” knappen trykkes vil siden på figur 4.3 vises for å påminne om informasjonen som skal legges inn i adgangskontrollen. Merk at bestillingen allerede er godkjent i dette systemet, ’Informasjon registrert’ knappen er der for å sikre at bestillingen blir gjennomført på begge plasser hvis man blir avbrutt i arbeidet.

Figur 4.3 Registrering på adgangskontroll

Når informasjonen er registrert er kortet klart til utlevering, og den som bestilte kortet vil kunne se at bestillingen har fått status ”godkjent”.

5 Vakter som må avsluttes

Å avslutte en vakt består i å registrere adgangsnivået på personens nøkkelkort som sperret selv om kortet ikke er registrert som innlevert. Slik får ikke personen adgang til dører etter at jobben er ferdig selv om personen er i besittelse av nøkkelkortet. Etter at en vakt er avsluttet og når nøkkelkortet blir levert inn fysisk, må det registreres som innlevert i systemet, som beskrevet i kapittel 6. Et eksempel på listen over vakter som må avsluttes (innen 24 timer) vises i figur 5.1. Trykk ”Detaljer” på den respektive vakten for å gå videre. Vaktene kan sorteres ved å trykke på de gule overskriftene og ved å velge sorteringsrekkefølge. Eventuelle vakter som skulle vært sperret allerede vil også vises i denne listen.



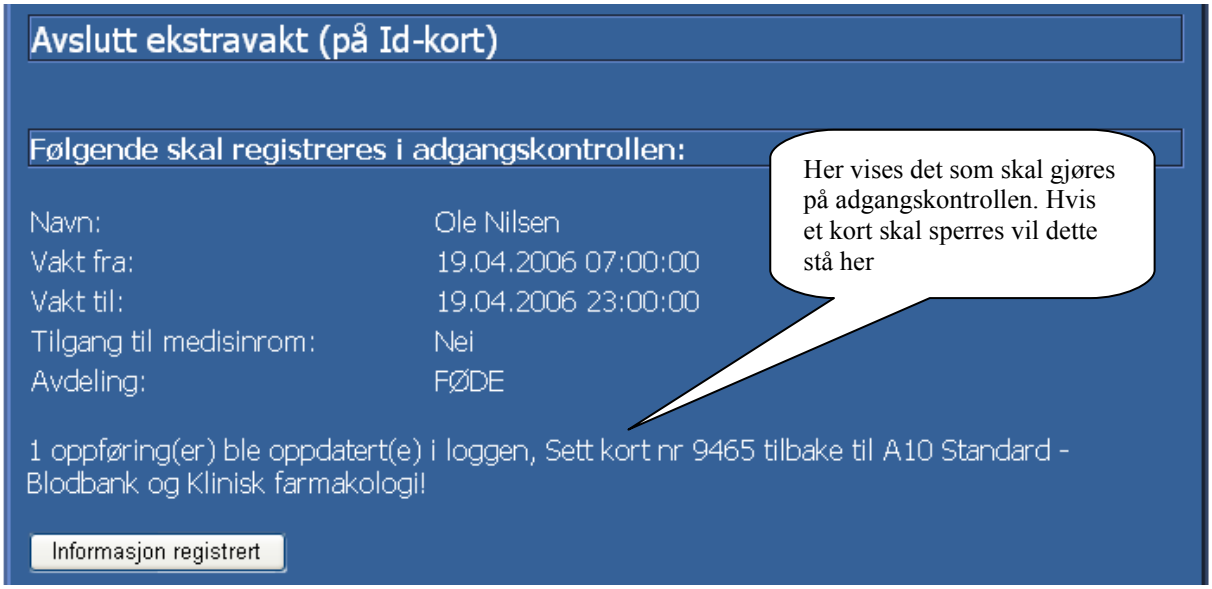
Figur 5.1 Vakter som må avsluttes

Etter man har trykket på detaljer kommer man til siden som vist på figur 5.2. Denne siden vil se litt forskjellig ut avhengig av situasjonen, i dette eksemplet har en ekstravakt på et idkort gått ut og skal avsluttes. De forskjellige situasjonene systemet tar hensyn til er som følger:

- Ekstravaktkort er utgått, burde vært innlevert men er ikke det. Ekstravaktkortet sperres.
- Idkort er utgått, burde vært innlevert men er ikke det. Idkortet sperres.
- Ekstravakt på idkort er utgått og eventuell tilgang skal sperres. Idkortet settes tilbake til det adgangsnivået det hadde før ekstravakten startet.
- Idkort som brukes på ekstravakt er utgått, idkortet sperres. Ekstravakten blir registrert som avsluttet.

Figur 5.2 Detaljer om vakt som skal avsluttes

Hvis man trykker ”Avbryt” kommer man tilbake til listen i figur 5.1, mens ”Avslutt vakt” registrerer endringene i systemet. Man kommer så til en ny side som vist i figur 5.3, for å minne på at endringer skal gjøres i adgangskontrollen.



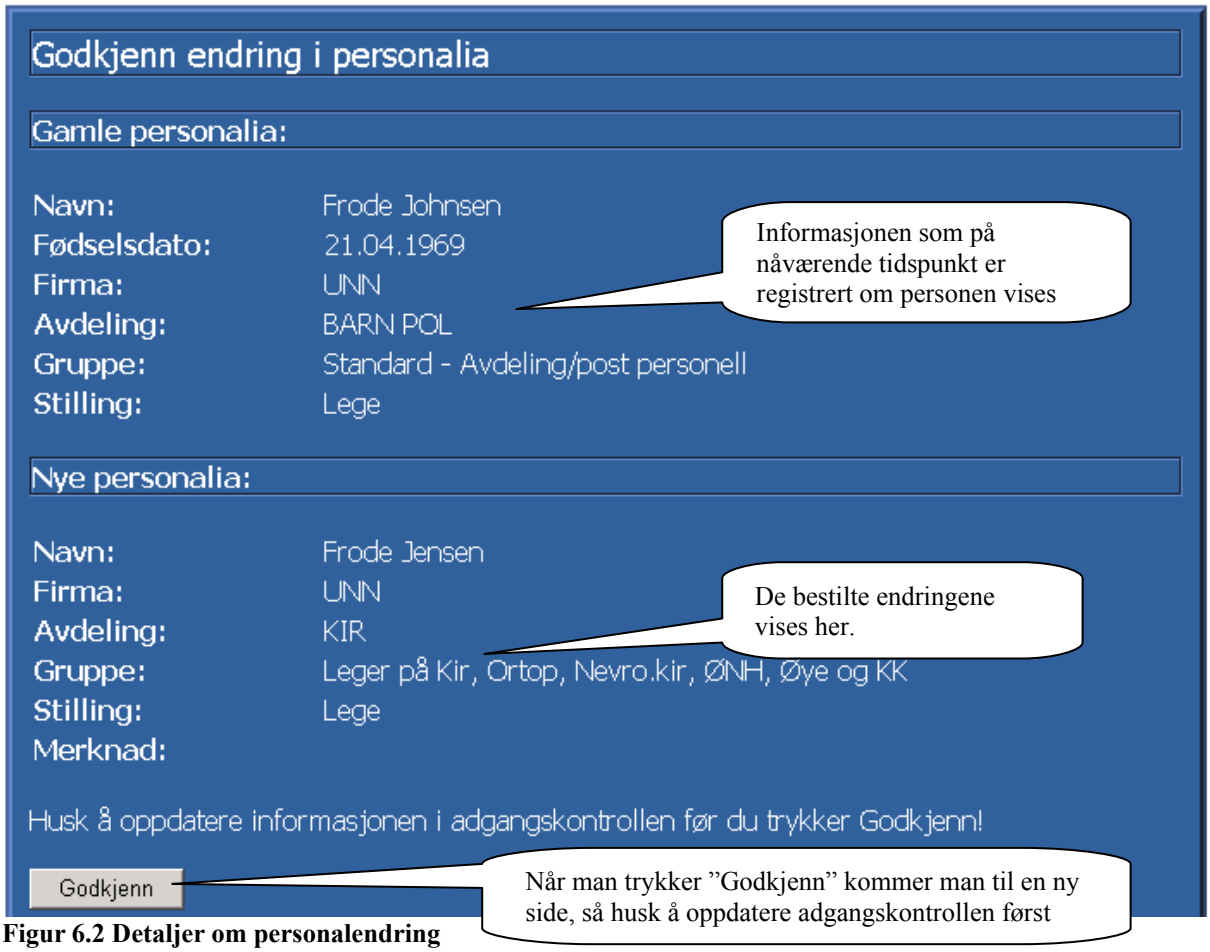
Figur 5.3 Bekreftelse på avsluttet vakt

6 Personalendringer

Hvis en fast ansatt (en person som har et idkort) skal bytte navn, avdeling, adgangsnivå, firma eller stilling, vil personalavdelingen sende en bestilling på personalendring. Det er disse som vil ligge i listen over bestilte personalendringer, se fig. 6.1.

Figur 6.1 Bestilte personalendringer

Når man har valgt en bestilling kommer man til en ny side som viser alle detaljene, både for det eksisterende ansettelsesforhold og de bestilte endringer, se fig. 6.2



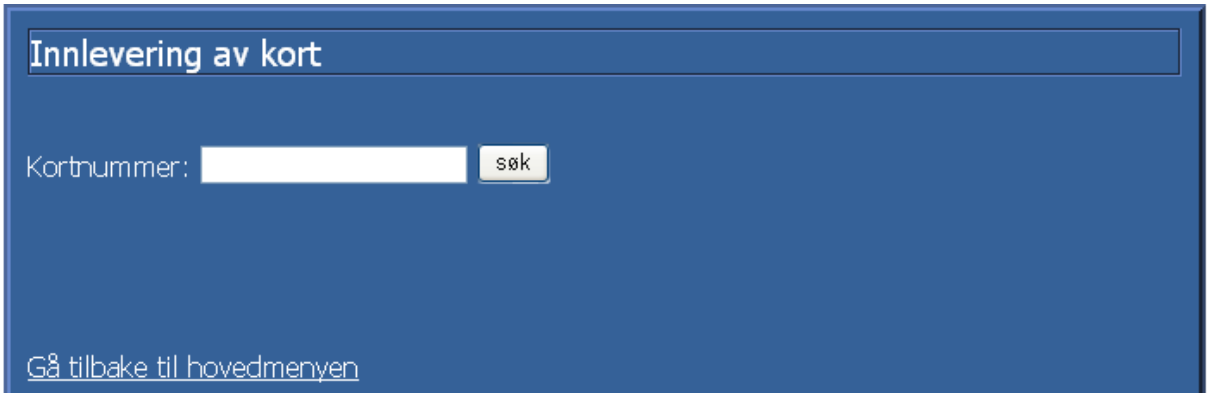
Figur 6.2 Detaljer om personalendring

7 Lever inn kort

Når ”Lever inn kort” på menyen trykkes vil man komme til bilde vist i figur 7.1. Denne funksjonen er for å levere inn kort som blir innlevert etter at en vakt er avsluttet. Ved innlevering av kort der vakten ikke er avsluttet (for eksempel hvis kortet leveres inn før utløpsdatoen er nådd, eller et idkort med ubestemt sluttdato leveres inn) vil systemet også avslutte vakten direkte når kortet blir innlevert. For å levere inn kort må kortnummeret skrives inn og søkes etter. Man vil da få opp detaljer om kortet som vist i figur 7.2. Disse detaljene vil variere etter situasjonen, og vil være en av følgende:

- Ekstravaktkort blir levert inn, settes til sperret adgangsnivå
- Idkort blir levert inn, settes til sperret adgangsnivå
- Idkort som brukes på en ekstravakt blir levert inn, kortet settes til sperret adgangsnivå og ekstravakten avsluttes
- Angitt kortnummer finnes ikke i systemet. Ingenting blir gjort i dette systemet, men det burde sjekkes på adgangskontrollen at det aktuelle kortet er registrert/sperret der

Når knappen ”Lever kort” trykkes blir man bedt om å oppdatere kortets adgangsnivå på adgangskontrollen, som vist i figur 7.3, og kortet vil bli registrert som innlevert i systemet.



Figur 7.1 Innlevering av kort

Figur 7.2 Detaljer om kort

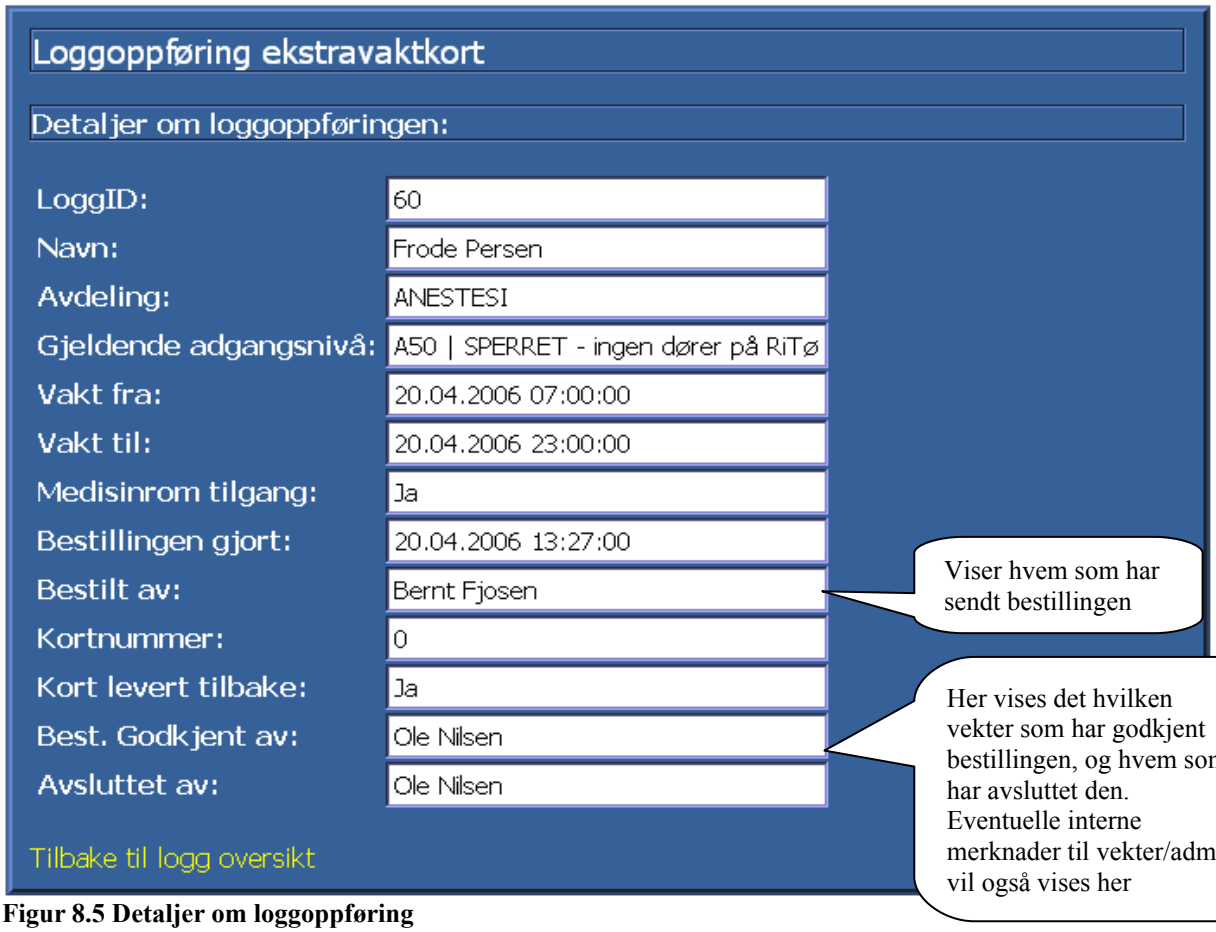
Figur 7.3 Bekreftelse på innlevering

Kortet er nå registrert som innlevert og kan legges vekk.

8 Logger

Under menyen ”Logger” er det mulig å se all informasjon som er lagret om kort som har blitt godkjent. I menyen er det et menyvalg for å se informasjon om alle loggoppføringer på ekstravakter, og det er et menyvalg for å se alle loggoppføringer på idkort. Det er også et menyvalg under ”Logger” som viser en liste over alle personer som er i besittelse av et idkort på nåværende tidspunkt. Et eksempel på oversikt over loggoppføringer i ekstravaktloggen er vist i figur 8.1. Denne listen vil typisk bli ganske stor i lengden ettersom flere og flere bestillinger kommer til. Administrator har mulighet til å slette oppføringer som er gammel og bare tar opp plass. Sortering av denne listen kan gjøres på samme måte som tidligere ved å trykke på de gule overskriftene, og kombinere dette med stigende eller synkende sorteringsrekkefølge. Både oppføringer der kort er levert eller ikke levert vil vises i denne loggen. Figur 8.2 viser ytterligere detaljer om valgt loggoppføring.

Figur 8.4 Oversikt over ekstravaktlogg



Figur 8.5 Detaljer om loggoppføring

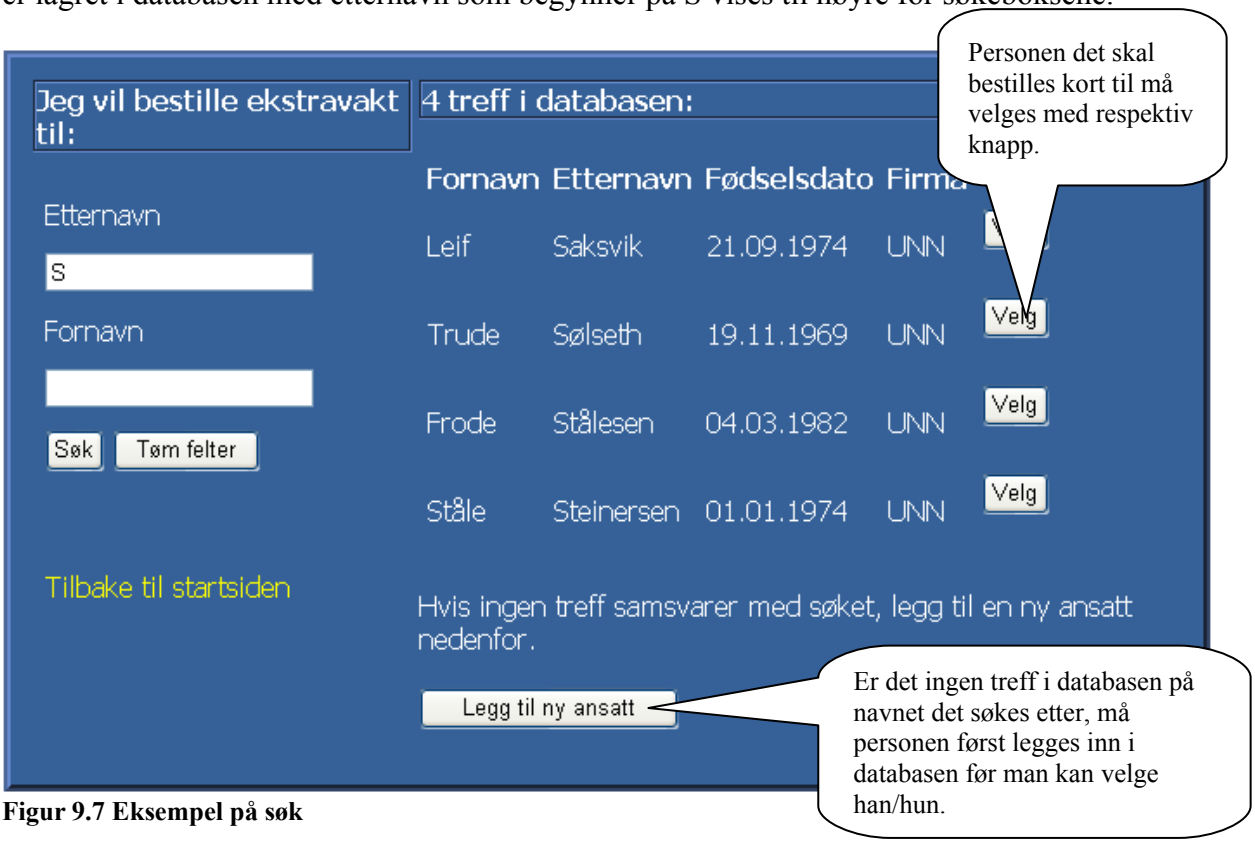
9 Bestillinger

For å bestille kort til en person må først personen søkes opp i databasen, dette gjelder både ekstravaktkort og idkort, for så å velge den personen som kortet skal bestilles til. Søkesiden vises i figur 9.1

Figur 9.6 Søk etter ansatt

Søking kan gjøres på både etternavn og fornavn, eller bare en av delene. Søkes det for eksempel på Hansen som etternavn, vil alle som er registrert med etternavn Hansen komme opp. Eventuelt kan det søkes etter de første bokstavene i et navn, og alle med et navn som begynner på disse bokstavene vil komme opp.

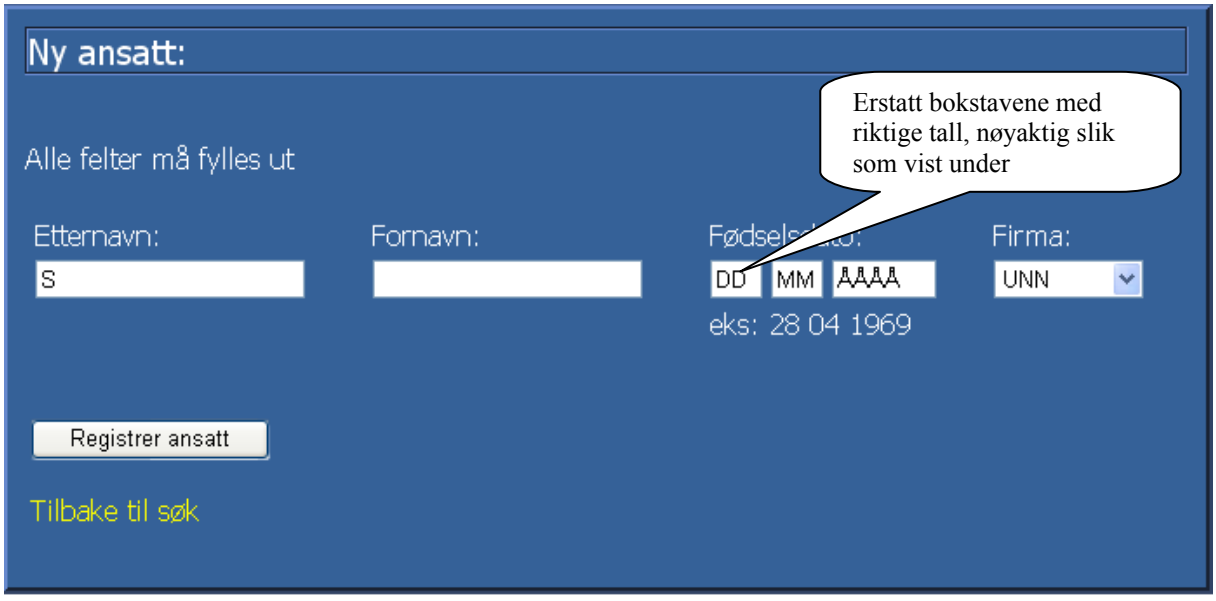
Figur 9.2 viser et søkeeksempel på bokstaven ”S” som etternavn, og resultatet er at alle som er lagret i databasen med etternavn som begynner på S vises til høyre for søkeboksene.



Figur 9.7 Eksempel på søk

Legg til ny ansatt

Her vil både etternavn og fornavn være forhåndsutfylt på bakgrunn fra de opplysningene som ble oppgitt i søkeboksene på forrige side, se figur 9.3. Det er mulig å endre disse hvis disse ikke stemmer, eventuelt legge til. I tillegg må fødselsdato fylles inn, og firma velges. Firma kan være UNN hvis personen er ansatt ved UNN eller det kan for eksempel være et vikarbyrå.

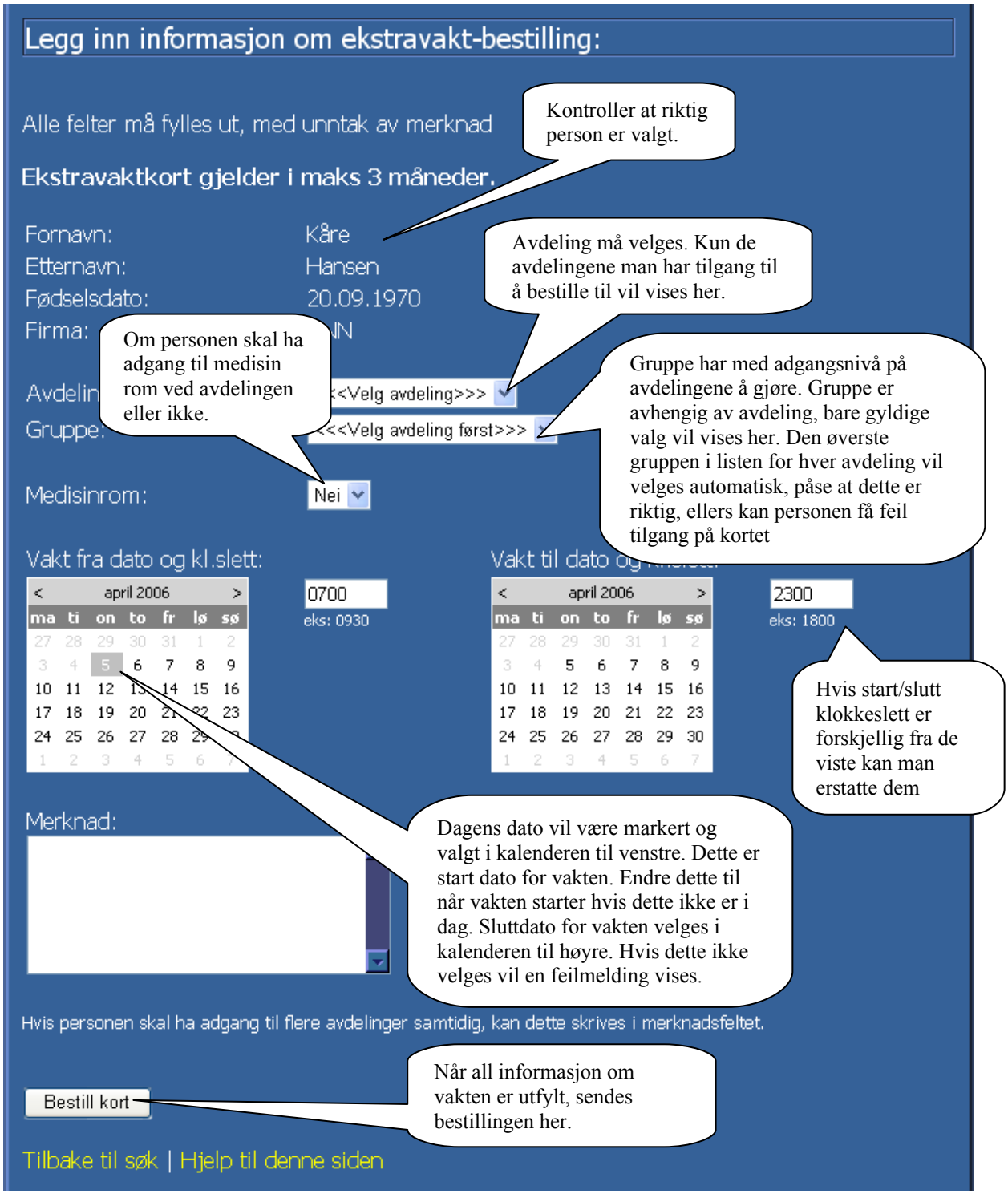


Figur 9. 8 Ny ansatt

Når personen er registrert, vil man bli sendt tilbake til søkesiden. Hvis det nå søkes etter denne personen, vil man få treff i databasen og man kan deretter velge personen for å komme videre til selve bestillingen.

Bestille ekstravaktkort

Når man har valgt person, kommer man til bestillingssiden som vist i figur 9.4:

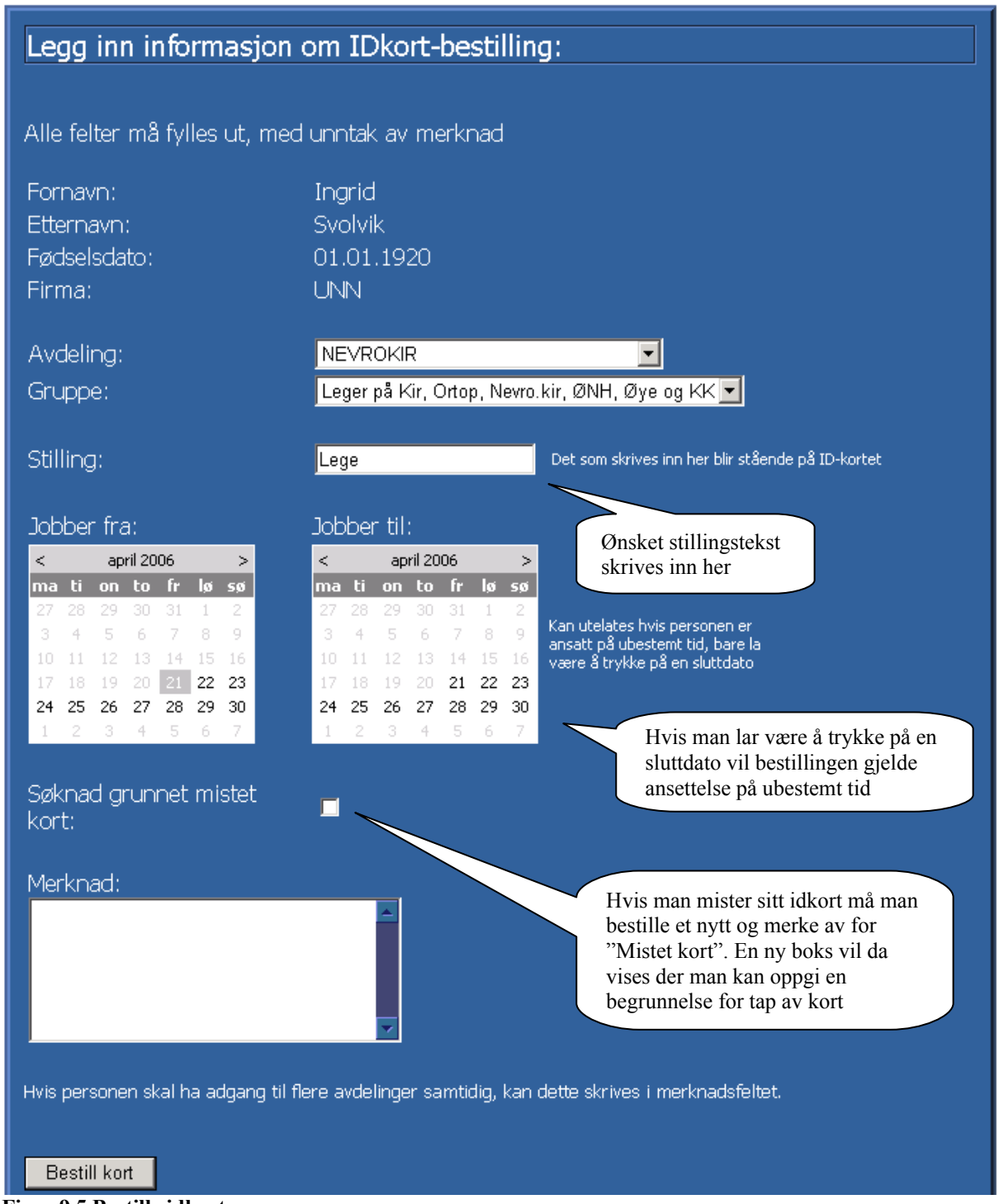


Figur 9.9 Bestillingssiden

Her må viktig informasjon om vakten fylles ut. Det er viktig å legge merke til at en ekstravakt kan maksimalt gjelde i 3 måneder. De feltene som er valgfritt er klokkeslett og merknad. Klokkeslett kan forandres eller det valgte klokkeslett kan brukes. I merknadsfeltet kan eventuelle meldinger eller spørsmål til vekter skrives. Hvis personen skal ha adgang til flere avdelinger samtidig skrives dette også her. Hvis feil person er valgt, kan man gå tilbake til søkesiden ved hjelp av linken nederst på siden.

Bestille Idkort

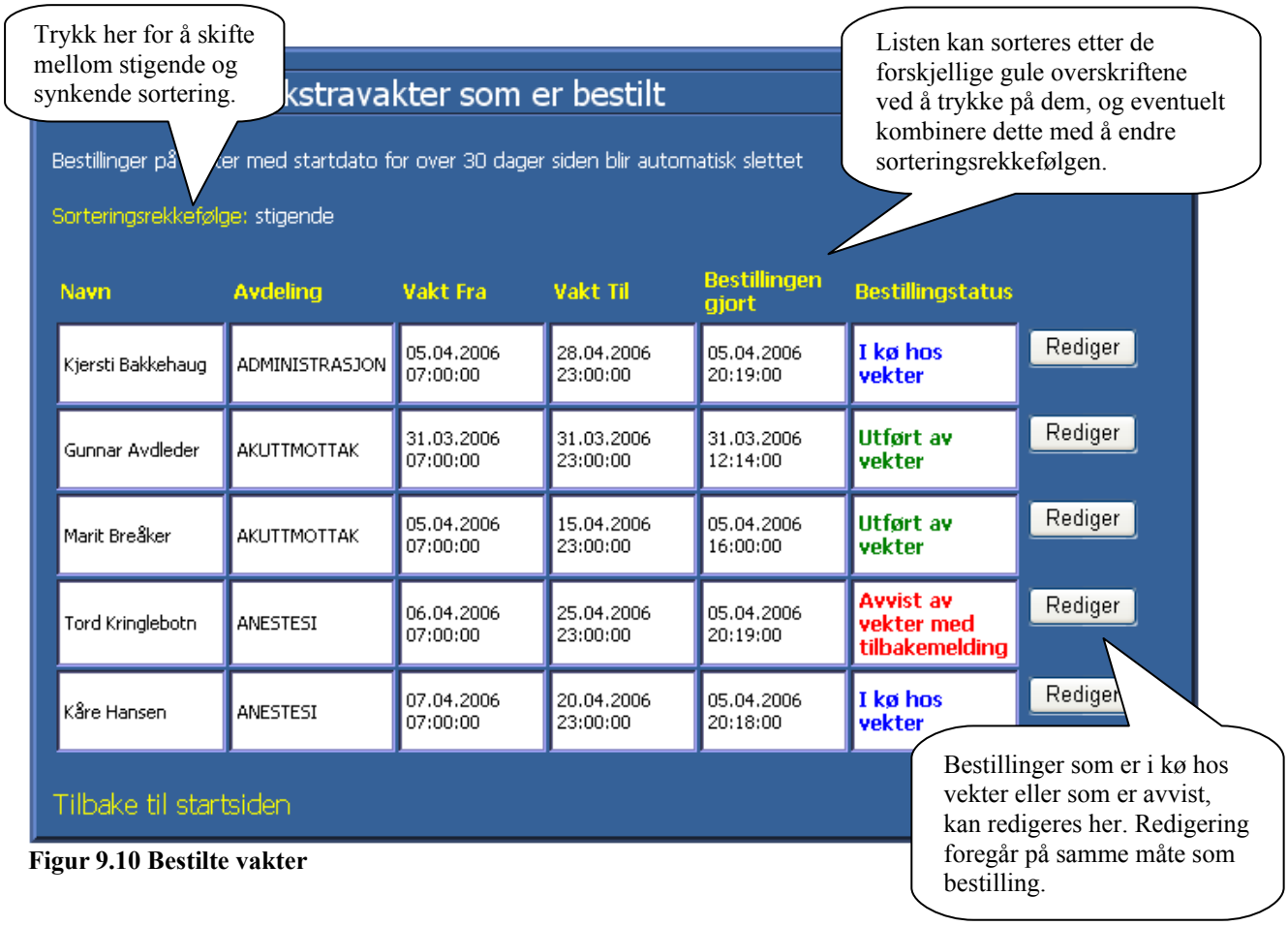
Idkort bestilles på samme måte som ekstravaktkort, med noen små forskjeller. Eksempel på dette kan sees i figur 9.5. For det første velger man ikke medisin rom tilgang. For det andre skriver man inn en stillingstittel som blir stående på idkortet. For det tredje velger man ikke start/slutt klokkeslett, bare dato, og for det fjerde trenger man ikke velge en sluttdato.



Figur 9.5 Bestille idkort

Bestilte vakter

Dette menyvalget viser en oversikt over alle bestillingene sendt den siste måneden fra din(e) avdeling(er). Et eksempel på dette vises i figur 9.6. Det er kun bestillinger sendt den siste måneden som vil vises her.



Figur 9.10 Bestilte vakter

Det er tre forskjellige bestillingsstatuser, og hver av disse vil forklares nærmere nedenfor.

- *I kø hos vekter*: Indikerer at bestillingen er sendt, men ikke behandlet av vekter. Det er mulig å redigere bestillingen helt til vekteren har tatt i mot og behandlet bestillingen.

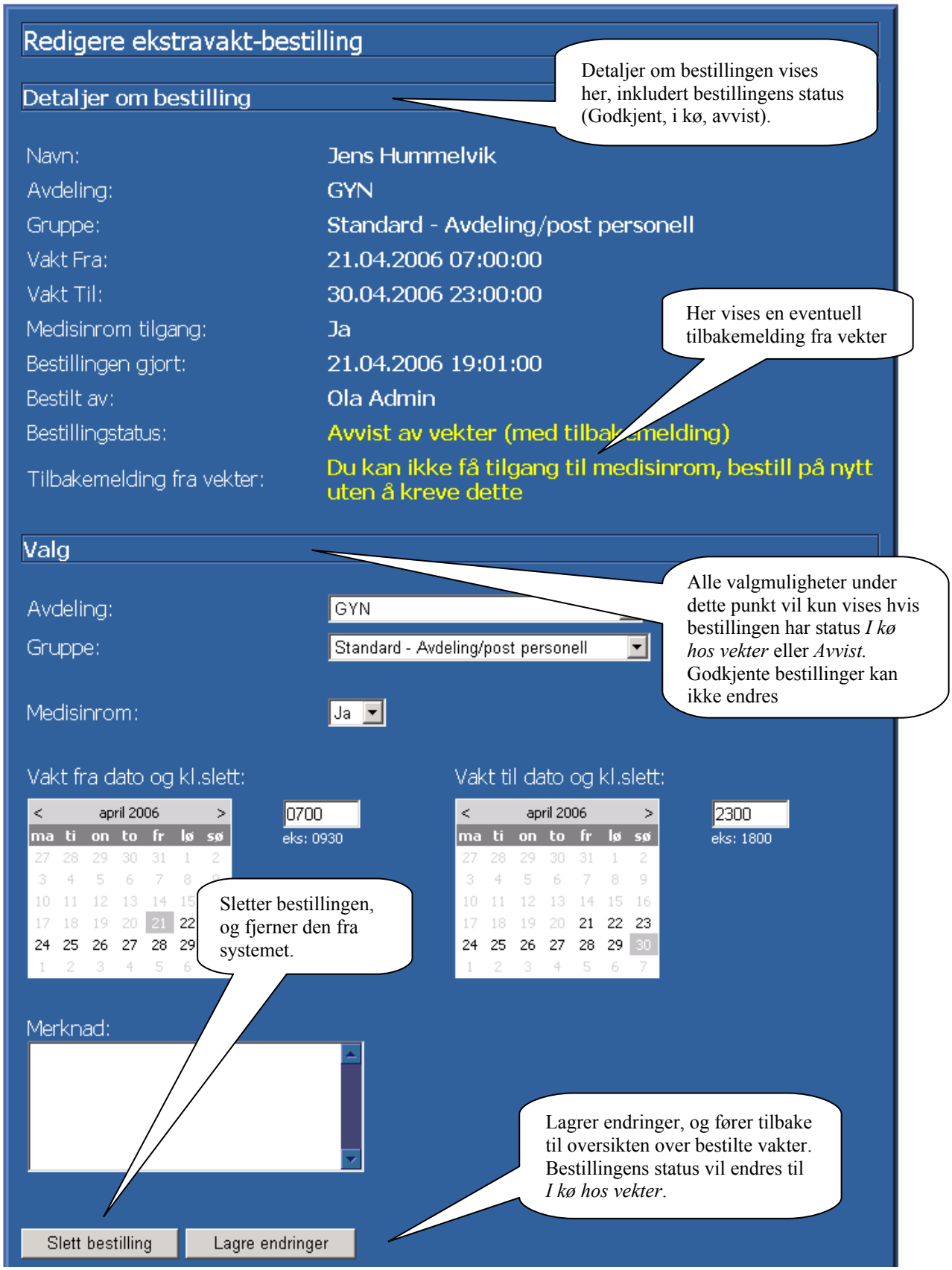
- *Utført av vekter med/uten tilbakemelding*: Hvis statusen indikerer utført er bestillingen godkjent og kortet vil ligge klart i resepsjonen. Det går ikke an å redigere en utført bestilling, men man vil få opp ytterligere detaljer om bestillingen hvis ”Rediger” trykkes. Hvis bestillingen er utført med tilbakemelding vil man få opp tilbakemeldingen blant disse detaljene. Det kan også trykkes direkte på ”Utført av vekter med tilbakemelding” for å få opp tilbakemeldingen.

- *Avvist av vekter med/uten tilbakemelding*: Hvis statusen indikerer avvist er bestillingen ikke godkjent, og nøkkelkort vil ikke bli lagd før eventuelle endringer på bestillingen blir gjort. Ved hjelp av ”Rediger” knappen vil man få opp de tidligere sendte opplysninger og en eventuell tilbakemelding fra vekter om hvorfor bestillingen ikke kan godkjennes. Her vil man så kunne endre bestillingen i henhold til

tilbakemeldingen. Det kan også trykkes direkte på ”Avvist av vekter med tilbakemelding” for å få opp tilbakemeldingen.

Redigere bestilling

Figur 9.7 viser siden for redigering av bestilling. Den nederste delen av figuren fra ”Valg” vil kun vises hvis bestillingen kan redigeres. Er dette ikke mulig, vil kun den øverste delen med detaljer om bestillingen vises. Redigering fungerer på nøyaktig samme måte som ved bestilling og inneholder nøyaktig de samme valgmuligheter. Vil man ikke gjøre noen endringer kan man gå tilbake til oversikten ved hjelp av linken nederst.



Figur 9.11 Redigering av bestilling

10 Endre brukernavn/passord

Det er mulig å endre brukernavn og passord. Siden for endring vises i figur 10.1. Det er ingen regler for brukernavn, men passord må bestå av mellom 6 og 20 tegn, og må inneholde minst 1 tall. Gammelt passord må skrives inn, i tillegg til at nytt passord skrives inn to ganger for å være sikker på at det er riktig skrevet. Det nåværende brukernavnet vil være fylt inn allerede, men det er mulig å bytte det ut mot hva som helst. Velges det et brukernavn som noen andre har vil det komme melding om dette.

Figur 1.12 Endring av brukernavn og passord

Bibliography

[1] ISO/IEC JTC 1. Iso/iec 9126 information technology - software quality characteristics and metrics, 2001.

[2] IEEE Std 610.12-1990. Ieee standard glossary of software engineering terminology, 1990.

[3] Asbjørn Aune. *Kvalitetsdrevet ledelse - Kvalitetsstyrte bedrifter*. Gyldendal Akademisk, third edition, 2002.

[4] James Bach. Exploratory testing explained, 2003. `http://www.satisfice.com/articles/et-article.pdf`.

[5] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley, second edition, 2004.

[6] Hans Buwalda. Soap opera testing. *Better Software*, February 2004.

[7] Ph.D. Cem Kaner J.D. What is a good test case?, 2003. `http://www.testingeducation.org/articles/what_is_a_good_test_case_star_2003_paper.pdf`.

[8] J. Crellin, T. Horn, and J. Preece. *Evaluating evaluation: A case study of the use of novel and conventional techniques in a small company*. Human Computer Interaction - Interact '90. Elsevier, IFIP, 1990. page 329-335.

[9] J. Rumbaugh G. Booch and I. Jacobson. *The Unified Software Development Process*. Addison Wesley, 1999.

[10] The EAGLES Evaluation Working Group. Evaluation of natural language processing systems, final report, 1995. `http://www.issco.unige.ch/ewg95/node84.html`.

[11] The EAGLES Evaluation Working Group. Evaluation of natural language processing systems, final report, 1995. `http://www.issco.unige.ch/ewg95/node88.html`.

[12] Y. Halmoe and G.A. Jenssen. *An empirical study of the KAS - Analyzing quality requirements and their definitions*. Norwegian University of Science and Technology (NTNU), 2005.

[13] M. Hammersley and P. Atkinson. *Feltmetodikk. Grunnlaget for feltarbeid og feltforskning.* Gyldendal, 1996.

[14] Cem Kaner. An introduction to scenario testing, 2003. `http://www.kaner.com/pdfs/ScenarioIntroVer4.pdf`.

[15] C. Karat. *Cost-benefit analysis of interative usability testing*. Human Computer Interaction - Interact '90. Elsevier, IFIP, 1990. page 351-356.

[16] Glenford J. Myers, Tom Badgett, Todd M. Thomas, and Corey Sandler. *The Art of Software Testing*. John Wiley & Sons,Inc., second edition, 2004.

[17] Kristen Ringdal. *Enhet og mangfold, Samfunnsvitenskapelig forskning og kvantitativ metode*. Fagbokforlaget, 2001.

[18] Andy Tinkham and Cem Kaner. Exploring exploratory testing, 2003. `http://www.testingeducation.org/articles/exploring_exploratory_testing_star_east_2003_paper.pdf`.

[19] A. Vainio-Larsson. *Evaluating the usability of user interfaces: Research in practice*. Human Computer Interaction - Interact '90. Elsevier, IFIP, 1990. page 323-328.

[20] Wikipedia. Black box testing, 2006. `http://en.wikipedia.org/wiki/Black_box_testing`.

[21] Wikipedia. Software quality, 2006. `http://en.wikipedia.org/wiki/Software_quality`.

[22] Wikipedia. White box testing, 2006. `http://en.wikipedia.org/wiki/White_box_testing`.

[23] Clas Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering, An introduction*. Kluwer Academic Publishers, 2000.