UNIVERSITY OF OSLO
Department of Informatics

Customizing an Open Source Web Portal Framework in a Business Context: Integrating Participatory Design with an Agile Approach

Master thesis

Damir Nedic &
Espen Aune Olsen

November 1st 2007

Abstract

In this thesis, we have investigated how an open source web portal can be used in a business context. This investigation required development of components for a web portal to meet the company's needs.

The motivation for this thesis was to design a web portal to organize the rapid growth of information and to improve the communication both internally and externally. The company that we collaborated with experienced expansion of the working staff, as well as in the customer base. Because the requirements for the web portal were uncertain, the development demanded a lot of communication and collaboration between us, as developers, and the company. With help from the Agile methodology and Participatory Design we were able to clarify the requirements and to overcome the problems during the development.

We have especially utilized Agile's Extreme Programming method as the developing technique. We argue that by applying this method in the given context, we attained the necessary foundation needed for the successful realization of the project. In addition, in situations where we supplemented it with techniques from Participatory Design; we gained a better understanding of the components to be made.

Table of contents

Acknowledgments

Our research process has been a journey through new and various technological terrains. It has involved different types of theories and demanded the production of many lines of code. The customization of the components required us to familiarize us with an unknown programming environment and it proved to be a huge challenge. During the process we have learned not only the technological aspect of a web portal, but also gained knowledge of working in team and collaborate with various participants. Since this project involved a real customer and can be interpreted as a task from “real life”, we are convinced that this experience will definitely be useful in the future.

Thanks to our supervisors Anders Mørch and Tone Bratteteig for useful feedback and motivational speeches when we really needed it.

We would like to thank Renate Andersen, Shazia Mushtaq and Kathrine Nygård, the other participants in the KIKK project, for all the interesting discussions, as well as providing us with a pleasant working atmosphere.

We would also like to thank the company we collaborated with. They gave us the opportunity to be a part of the developing process, which proved to be very interesting as well as educational. They also provided us with an office supplemented with all necessary equipment during the development process, and for this we are very grateful.

A special thanks to our girlfriends, friends and family who believed in us and supported us through the entire process.

Finally, we would like thank to each other. This process has had its ups and downs, but together we were able to make it possible.

1 Introduction

In our master thesis we describe how a web portal contributes to strengthening the communication between a company and its customers as well as among the employees. The motivation for writing the thesis is based on the desire to get involved in a "real life" setting relevant in today's system development market. We also wanted to get experience with a brand new technology, while getting a feeling of how open source applications can be utilized in a business setting. In the project we have worked with both professionals and fellow researchers.

Our starting point was an already existing open source web portal framework. The framework consists of a complete web portal that can be extended through the addition of independent components that serve different needs. Through customization of components we attempt to satisfy the collaborating company's needs. Close interaction with the company and its representatives enabled us to achieve this task

1.1 Research question

We have decided to focus on one key research problem.

1. What are relevant aspects to consider when adopting a software engineering (SE) methodology for developing a web portal in a business context, as seen from the following perspectives:
    a. The requirements are unknown
    b. Interaction with customers is vital
    c. Development should be based on customization of existing components

We were involved in a development project where the company's representatives were unable to specify the requirements. The reason for this uncertainty was that the representatives only recently discovered the problem they were facing and hence only had a vague idea of what the possible solution should be. By applying the Agile methodology and Participatory Design (PD) we were able to uncover the requirements and needs necessary for getting started with the development. During the process of development the very same methods helped us in managing and solving the problems we encountered.

The project relied largely on communication between the participants rather than written documentation. The people centric techniques of the Agile methodology and PD supported the company's representative and the developers in expressing requirements and needed functionality through face-to-face communication in meetings, design workshops and presentation of prototypes. The prototypes were a good starting point for further discussion and customer feedback. These sessions often resulted in the discovery of requirements that none of the involved participants were previously aware of.

Our main task in the project has been to develop components with specific functionality. These components were to be used within an open source web portal framework. Basically, the development was based on two premises in order to be considered successful when finished:

1. The components should meet the requirements obtained from the company's representatives.
2. The components should integrate fully with the framework.

Because the requirements often were vague, the development often included several iterations before the component could be considered to be finished. Through collaboration with the company's representatives we gradually approached the appropriate solution. Hence, most of the development was based on adaptation to recently discovered requirements. It is also important that the components fit within a given context, namely the web portal framework. The framework consists of a set of predefined rules and the developer must act accordingly. In other words, the components must be customized to meet the requirements given by the framework as well as the requirements stated by the company's representative. Therefore, our focus lies on the overall process that starts with requirement gathering and ends with a working component.

1.2 Thesis structure

The thesis is divided into four parts. Each part has a different object but put together they form the structure necessary for the reader to understand our work from the initial planning of the project to the delivered prototype. The four parts are:

1. Introduction and context
2. Background information
3. System development
4. Discussion

The first part is meant as an introduction to our work. We have already presented our research questions, and in the next chapter we explain the context surrounding the KIKK project. We introduce the company we have been collaborating with as well as our fellow researchers at InterMedia.

The second part is devoted to the theories we have applied in the process of developing the components according to the specific needs. We also present the background information necessary to understand how and why we did things the way we did. Our main focus has been the people centric approaches of Agile methodology and PD. The chapter on PD covers the most important concepts and then explains how it can be applied in a practical setting through the use of several different techniques. In the chapter on Software engineering we discuss the choice of the Agile approach rather than relying on the "traditional" method of plan driven development. We also present some of the advantages of working with the Agile methodology and mention the most well known methods. We have chosen to give an in depth presentation of Extreme Programming as this was the method we relied upon during the period of development. In the chapter on Programming environment, we explain some of the tools we have used while working with the master thesis. The goal of the section is to give the reader a better understanding of how we approached the task of building and customizing the web portal.

Part three of our thesis focuses on the development process. In the chapter on the Prototype we give a presentation of the result of our work. For each module, we present a relevant scenario that will give the reader an understanding of the functionality and the overall feeling of using the module. In the Development chapter we present relevant findings from the process of customizing the modules. While explaining how the agile methods prepared us and helped us in overcoming the challenges we met, we also give examples of how we took advantages of the services offered by the web portal framework.

In the last part of the thesis we discuss some of the limitations of using the techniques we have applied during our work. We try to point out some of the things we did wrong or could have done better. We also discuss how the expectations we had before starting the work were met. At the same time we look at how we handled some of the challenges and hindrances we experienced. In the chapter on directions for further development we list some of the requirements we simply did not have time to implement during the period of development. In the conclusion we sum up our thoughts of the work and list the things we feel we have accomplished during our work with the master thesis.

2 Context: The KIKK project

Our experience is based on the KIKK (Knowledge Management for Internal Communication and Customer support) project. This project is a collaboration between InterMedia, the University of Oslo (UiO) and an external company (From now on, referred to as "the company"), a middle sized Norwegian company that develops planning software for the offshore and building industry in Norway. The company's core business has been the development and support of project management tools for large, complex projects. The company has primarily focused on the offshore business in Norway as a lot of their employees have experience from this section of the industry. The company is originally based in Stavanger. Recently however, the company has expanded to other sector of the industry because of an enormous demand for project planning software. They are now turning towards the building industry as this is an area where project planning is crucial. As a consequence of this, the company has opened an additional office in Oslo and also keeps hiring new employees at a steady rate. The company aims to double their number of customers over a period of approximately two years.

When facing such a rapid growth, the company has realized that it is looking at two major challenges. The first challenge is to keep providing fast and reliable support for the customers. The company's support strategy has always been to provide personal support and often in less than 24 hours. Because some of the company's key employees have a close relationship to the offshore industry a lot of the support inquiries are made directly to a specific person in the company, simply because of previous mutual work experience. This has lead to some problems. Today the experienced individuals in the company are often overworked because of the high number of support inquires by phone and e-mail.

The second challenge concerns the new employees and the geographical distance between the office in Stavanger and Oslo. The new employees have a hard time learning and adjusting to their new roles because they just finished their education and have little or no experience in the field (Nygård and Mørch, 2007). The workforce expansion along with unorganized internal communication leaves the company in a position where a lot of burden is put on the senior employees.

The goal of the KIKK project is to structure some of this knowledge, held by the key employees, by, among other things, introducing a web based knowledge portal (web portal for short) (Nygård and Mørch, 2007). This will lead to changes in the how the communication is handled both internally and externally, as the web portal will have two main functions.

1. Provide a central knowledge base for the employees and consultants in the field.
2. Support and improve customer interaction.

The company aims to use the web portal as a knowledge base for handling communication between the two offices and the employees. One goal is to preserve some of the knowledge that the senior employees have gained through years of experience. Because a large percentage of the support is given directly to customers through phone and e-mail, it is hard for the new employees to gain knowledge from this process. If support is handled through a web portal the knowledge will be stored in a common medium that the employees can benefit from. The web portal will also provide a means of communication for the consultants out in the field as they will be able to access the stored information from any location.

The second goal of the web portal is to support customers when they encounter problems with the company's software. This function will allow each customer to log in and leave a support request, view status on their requests or gain knowledge by reading other customers' support requests and the company's replies. The web portal will also provide the customers with news from the company, information about software updates, software manuals and lists of frequently asked questions. The building of the web portal will be further elaborated in this master thesis.

2.1 Project progress

In this chapter we will explain the proposed time line for our involvement in the KIKK project. However, as the project went through the different phases, there were changes to the schedule and we present the final version here. The project was initially divided into three different phases.

1. Pre-analysis of the company
2. Design and development of the web portal
3. Launch of the web portal, user feedback

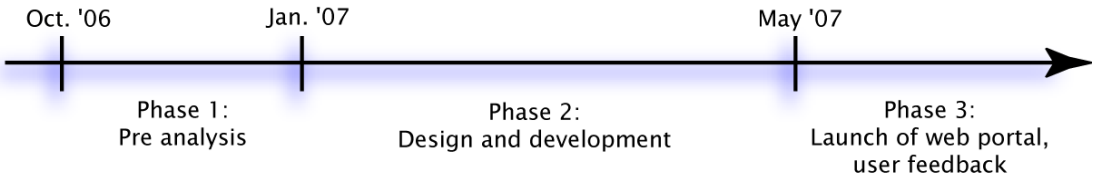

Figure 1: Timeline of the KIKK project

Phase one is a pre-analysis phase and will focus on exploration of the company and their employees. This exploration will be based on interviews with key individuals in the company. The goal of this phase is to map the work and information flow as it is today. Phase two focuses on design and development. The phase will be introduced with a design workshop where people from both InterMedia and the company will be invited to participate and express their opinions. After analyzing the results from the workshop we will start the work with developing the web portal. The development will be done in the spirit of the Agile methodology with frequent demonstrations and feedback from actual users. At the end of this stage we review our results so far by demonstrating the portal and through constructive discussion in a second workshop. In addition to our work with the web portal, the other members of the KIKK project will contribute by doing analysis of the company and propose interventions through either written reports or presentations on how the company may change in order to enhance the flow of communication.

The third phase is a post development phase. At a future stage the web portal will be launched to the company's employees and customers. By doing some user testing and interviews we will receive feedback that can be used for analysis. The goal of this phase is to figure out if our intervention actually meant something to the company. This will be done by analyzing the work flow both internally and externally and see if there has been any changes.

2.2 Project participants

InterMedia's research team consists of four master students, one Ph.D. student and our research supervisor. In the KIKK project we have mainly focused on the development of a web portal. The entire team however, has contributed in gathering data and doing empirical analysis to make the development process possible. The other three students went to Stavanger to interview key employees from the company before we got involved in the project. However, the KIKK project has a

broader aspect than just creating a web portal and each of the other members of our research team focuses on their specific tasks. As mentioned above, our task has been to create a web portal to enhance knowledge among the employees and the customers. The other students have mainly focused on knowledge building and how communication may be enhanced both internally and externally (Nygård and Mørch, 2007).

2.3 Research method

In the project we apply the *people centric approaches of Agile methodology* for the development and *Participatory design for specifying requirements* and to get a general insight of the structure and policies within the company. The main reason for this is that the project involves multiple participants, and the general focus of the KIKK project is broader than our contribution. In the next chapters we will outline some of the theory we have applied during the research for our master thesis.

3 Participatory Design

Participatory Design (PD) is an approach where the users are involved as active participants in all stages of system development. The method has traditionally been concerned with workplace practitioners' involvement in the process of design and decision-making. There are several reasons for relying on user participation in the design process. Consider the following example from Ellen Bravo (Bravo, 1993). A law firm decided to install a new carpet in the entrance to their office. However, the lawyers were concerned that the secretary's movable chairs would make tracks on the brand new carpet. The lawyers' solution was to nail the chairs to floor without even consulting the secretaries. Unfortunately, the next day the secretaries were unable to do their job because they were dependent on rolling the chair between the computer, the telephone and the typewriter.

This story is a good example of the importance of user participation while making design decisions. Allowing the users to be involved in the process of development is more than the benefit of a democratic work place. It also ensures that the users have the chance to express their opinions. PD also contributes to the enhancement of skills. This means that the designers are paying attentions to things that are often left out of formal specifications, like tacit knowledge and communication (Greenbaum and Kyng, 1992; Bratteteig, 1997).

In our work with the thesis we have seen some of the advantages of applying PD to the design process.

- Mutual learning, where the participants (future users and developers) learn about each other's domain during the design process (Bratteteig, 1997). Create a common language that can be used when experts from different domains come together to design a system.
- Understanding the context surrounding the software. PD considers software to be a network of people, practices and technology, rather than just a piece of software from a box (Brynhildsen and Mørch, 2005).
- Allow end users to participate in the development of a system that is likely to influence their job in the future (Bjerknes and Bratteteig, 1995).
- Feeling of ownership to the developed system and reduced resistance to change (Brynhildsen and Mørch, 2005).

When working with users it's vital to develop a real understanding between the involved participants. Mutual learning (ML) means that the developers as well as the users learn from each other during the initial stages of the development. ML is often necessary when developers come together with users from different domains to develop an information system (Bratteteig, 1997). This gives the system developers an insight of the users' domain, but it also gives the users a better understanding of how the developed system will work and interact. It is important to create this understanding as it will lead to a more acceptable outcome.

A common reason to strive for mutual learning is the fact that users don't always do what they say that they are doing in work situations (Bratteteig, 1997). If ML exists developers are able to understand why work practitioners sometimes don't go by the book. For instance, experienced nurses don't always follow the formal procedure because they, after years of experience, know how to handle certain situations. This knowledge can only be shared with developers through communication and ML (Bratteteig, 1997).

Through the process of ML, the developers and users create a *common language*. This is useful because users sometimes are unable to understand the technical language surrounding system development. At the same time, it's important that domain experts express their needs in language understandable by the developers. This creates a bridge between the two domains that is a good starting point for further discussing design and implementation (Brynhildsen and Mørch, 2005).

A closely related topic is how PD can help the developers in getting a better insight of the work processes of the users. In order to design the system the users actually need, the developers need to broaden their perspective beyond the exact context where the system is to be developed and deployed. In this way they can benefit from the knowledge of how the organization is structured and how political decisions are made.

Another goal of PD is to let the users feel like they are part of the solution. When the users are allowed to play with simple mock-ups or prototypes they will feel ownership to the product when their suggestions during the design phase have been realized in the finished product (Mock ups and prototyping will be described later). The users will also get a clearer understanding of the idea behind the interaction. This ownership will ease the introduction of a new software system in the

work place. It will also lead to a spread of positive attitude towards the new software, thus making the employees less futile to change (Brynhildsen and Mørch, 2005).

However, PD is more than a set of techniques that can be applied to a given situation. It also has a long of history of improvements in workplace democracy and equality among the employees.

3.1 History

Participatory Design started as an initiative to include the workers in the development and introduction of computer-based systems in the workplace. This Scandinavian approach was developed in Norway by Nygaard and his associates in the NJMF project (Nygaard, 1979). In this project the iron and metal unions demanded that the workers were included in the development of new systems as they feared that the computerization of the workplace would make the regular workers obsolete (Nygaard, 1979; Kensing and Blomberg, 1998). The unions also feared that the new systems would be designed based on the managers' decisions without involving the employees. Nygaard worked with the unions to introduce a development model that supported the employees' point of view by involving workplace practitioners in the process of design and decision-making.

3.1.1 Democracy at work

As mentioned above the purpose of introducing PD originally was to increase democracy in the work place. One goal was to listen to a wider range of voices when making decisions that could possibly have big impact on a company. However, it also led to a point where the actual users could participate in the process of developing the system (Bjerknes and Bratteteig, 1995).

PD is democratic in the sense that all the users that are involved, whether they are a manager or just a regular employee, have an equal saying when it comes to making suggestions in design workshops or other types of development discussions. PD rejects the thought that computerization of the workplace will make the common worker obsolete. Instead it is seen as an attempt to give the worker a better tool for doing their job. PD contradicts "traditional" design methods because it looks at the representative users as the main source of information, where "traditional" design methods consider expert users as the most valuable asset. It also embraces the idea that computer and

software should be looked at in the context of the workplace and its users, instead as isolated products (Schuler and Namioka, 1993).

3.2 Participatory design techniques

PD is known to be practiced differently from project to project depending on the experience, knowledge and visions of the people involved. This makes it hard to formulate specific points that have to be included in the design process and decision-making when using PD. However, it's commonly agreed that the most valuable information is usually gathered from discussion where multiple participants attend (Clement, 2005). In the next section we will elaborate on the techniques we applied while working with the master thesis.

- Design workshop
- Scenarios
- Mock-ups and cooperative prototyping

3.2.1 Design workshop

Design workshops are an efficient tool during the design phase because users rarely do what they actually think their doing when explaining it to design experts. Hence, interviewing the users to get their opinion is seldom the most effective way for the design experts to learn about the workers practice. In design workshops the users and the designers come together to discuss how things are done in practice today and how they want things to be done in the future. During this process the users and design experts create a *common language* which allow the participants to have clear, meaningful discussions where everyone is able to understand each other (Brynhildsen and Mørch, 2005). However, before this workshop can begin it is important for the designers to understand the domain they are dealing with. An efficient way to achieve this understanding is by visiting the workplace. These visits may include interviews with the employees or employees giving demonstrations to the designers (Bødker, Grønbæk et al., 1995). Future workshop is one example of a design workshop.

Future workshop

Kensing and Madsen have described Future Workshop (Kensing and Madsen, 1992). The vision behind the development of this method is quite simple:

> ***"Empirical research shows that while a lot of resources might be used on interviewing users about their current work, few or no resources are used in helping users and designers generate alternative ideas about how they would like their work situation to be in the future"***

– Kensing and Madsen (Kensing and Madsen, 1992)

(Kensing and Madsen, 1992) divide the workshop into three phases. *The critique phase* is a brain storming session where the participants discuss current problems at work. To make this session as fast and effective as possible each speaker is limited to 30 seconds. The users are also encouraged to throw out any idea they might have without having to defend it, thus allowing the less vocal workers to participate. The second phase is *the fantasy phase*. During this phase, the users are asked to discuss how they see their work practice to be in the future. The discussion may be based on turning the negative things that came up during phase one into positive things. Again, no suggestion is dismissed and everything that gets thrown out is accepted. During *the implementation phase* the participants discuss the suggestions that have come so far. The completely unrealistic suggestions are either removed or revised into more realistic visions.

It is important to notice that the language in this type of workshop is a natural language that everyone can understand. Both users (which are "experts" in their own domain) and the designers normally use a language which may be domain specific. However, it is important to develop a common language so everyone can understand the ideas and suggestions that are mentioned. Especially when design experts use a natural language the users feel more confident and hence will feel more like a part of the solution.

3.2.2 Scenarios

Scenarios first appeared on the design scene in the early 1990's. Scenarios have some unusual features. First of all, it doesn't seem to have any relations to prior techniques used in design. Secondly, there are no best practice-routines for how to use them. Different projects or stages in projects require a different approach. In addition to this, there is no common definition of how narrow or broad the scope of the scenario should be (Kuutti, 1995). This makes the method versatile because scenarios can be used in a numerous of different ways to make it suitable for the present problem at hand. It can be used at any stage in the development process. At the early stages it might be used as a tool for the designers to understand the users. Later on it might be used to simplify the designers' solution to the users. This is one of the advantages when using scenarios. They are very informal and there are no strict guidelines as to how a scenario should be written concerning language or form. In design workshops, scenarios may be used to describe how users will interact with a computer system. It also works as an illustrative story that makes it easier for both designers and user to understand how the interaction between a system and the user actually will be like when the system is finished (Kensing and Blomberg, 1998).

3.2.3 Mock-ups and cooperative prototyping

Another aspect of design workshops is making the end user get a *hands-on experience* of how the system may work (Ehn and Kyng, 1992). This can be done with mock-ups or simple prototypes. Mock-ups can be an efficient tool in the early stages of a design process. They encourage the users to think outside the box, which may lead to solutions that otherwise would not been taken into consideration. It also allows the users to think beyond what is possible with today’s tools and technologies. Instead the user can be creative and think about the services that are needed, regardless of the actual possibility to implement them. By introducing mock-ups in familiar materials such as wood or cardboard the users quickly feel like they are dealing with something *understandable*, although the mock-up is simulating something much more advanced. It is also a cheap way to introduce something that otherwise would have cost a lot of money, i.e. producing time consuming prototypes. Hence, designers can experiment with different solutions within a fairly limited budget. Mock-ups are also fun to play with and can easily be modified with the use of

familiar tools such as scissors, knives, hammer or a pen (Ehn and Kyng, 1992; Brynhildsen and Mørch, 2005)

During development users may participate in the process through cooperative prototyping. With this technique the users are invited to sit next to the developers. The aim of this process is to establish a design process where the both users and developers are taking advantage of their different qualifications in order to combine the specific domain knowledge with the technical aspect. Ultimately, the goal is to stimulate user participation and creativity (Bødker and Grønbæk, 1992).

3.3 Limitations of PD

Generally, one might argue that most research has been performed by the technique's followers and practitioners. As a result, most studies focuses on the positive sides of PD, rather than trying to highlight the weaknesses and flaws (Silva and Breuleux, 1994).

Another critique against PD has been that some of its followers puts too much into the concept of political awareness. Even though PD has great potential in leading way for political emancipation, it doesn't necessarily do so. It is argued that other approaches than PD may contribute just as much (Beck, 2002). It has also been argued that PD can range from, at minimum, the concern of how to involve users in the design of information systems, to, at a maximum, a determination to empower the users through workplace (Bossen, 2006). This wide range of use might require a highly skilled PD practitioner to participate in order to practice PD "correctly" and thus lower PD's status as a technique that empowers the common user or employee.

A third objection against PD is that it requires a large amount of resources. The users must be taken out of their jobs and that is rarely economically justifiable in a business context, and it's hence, mainly used in academia (Hansson, Dittrich et al., 2006).

A fourth objection against PD is how its techniques sometimes lead to workaround of the old problems, instead of focusing on new solutions involving up to date technologies (Bødker and Iversen, 2002).

4 Software Engineering

> *"Modern software systems are hard to build, because we've already built the easy ones"*
>
> **- Tom DeMarco**

Software engineering (SE) is an old discipline that originated in a NATO Software Engineering Conference in 1968 (Bauer, Bolliet et al., 1968). It has been practiced ever since and it is continually evolving. It applies engineering principles to the creation, operation, and maintenance of software systems.

To make sure that SE terms are used properly, IEEE has established a terminology referred to as *IEEE Standard Glossary of Software Engineering Terminology*. It ensures agreements among all terms and concepts like programming languages, requirements and testing (IEEE, 1983). On an annual basis there are held several conferences where SE topics are discussed. Two of the most important are ICSE (International Conference on Software Engineering)[1] and COMPSAC (Annual International Computer Software and Applications Conference)[2].

A *software process* is also part of SE. It is defined as a set of activities that produce a software product (Sommerville, 2004). These activities include analysis, planning, implementation, testing and maintenance. *Software development* is the part of a software process where the software is constructed. Different software development methods define a structure to be imposed on the development of a software product.

There are two major development methodologies that we will discuss in this thesis. The first one is the Plan-driven methodology and the second one is the Agile methodology. Each of them has a specific application domain where they work best, and where the other will have difficulties.

[1] URL to the latest conference: http://web4.cs.ucl.ac.uk/icse07/

[2] URL to the latest conference: http://conferences.computer.org/compsac/2006/

The Plan driven methodology is known for its disciplined, detailed and bureaucratic approach where as the Agile methodology is quite the opposite. There are many different development methods within each of the methodologies (we will mention some of these). These methods suggest which processes that have to be followed, notations to be used as well as rules and guidelines during development (Sommerville, 2004).

4.1 Plan-driven methodology

Plan-driven methodologies or "engineering methodologies" (Fowler, 2000) was inspired from engineering disciplines such as civil or mechanical engineering. These disciplines put a lot of emphasis on planning before building. The Plan-driven methodology can be compared with building a bridge. Every piece of the bridge has to be known and precisely specified in advance before starting construction work. In software development, it means that every part and each piece of the functionality is specified in the project description. This is possible when the project is predictable and therefore can and should be planned in detail from start to end. In other words, plan-driven development is a formal, well defined approach to creating an application.

A typical example of the Plan-driven methodology is the Waterfall model. In the Waterfall development model, specification, development, validation and evolution are all divided into separate phases of the process, such as requirements specification, software design, implementation, verification (testing and releasing) and maintenance (Figure 2). Each process of the software development has to be finished before the process can proceed to the next (Sommerville, 2004). The development schedule is set at the beginning of the project and conforms to this schedule until the end of the project. To be able to predict the outcome is good, as long there is no need to do changes during the development process. If changes are unavoidable, the project schedule is displaced and the project's final due date is uncertain (Hayes and Andrews, 2004). Ironically, the customers and the developers are aware that requirements can indeed be changed, since everything around them is changing. For instance, technology, people, resources, environment (Fowler, 2000).

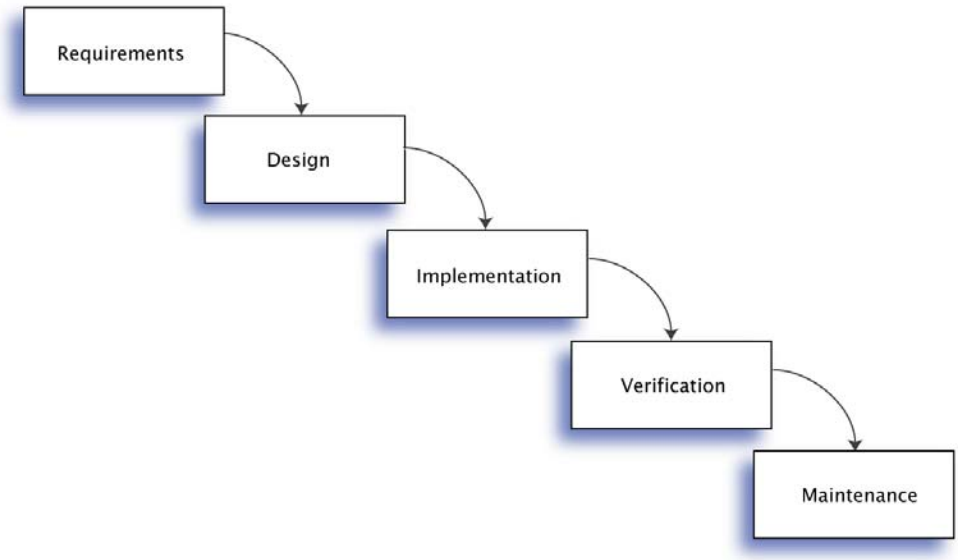

Figure 2: The waterfall model

Generally, Plan-driven methodologies have not been known for being terribly successful (Fowler, 2000). One argument is that they are too bureaucratic. It requires a large amount of paper work to follow the methodology, which may slow down the development of what the customer really want – a working product. Nevertheless, the Waterfall method has areas where it should be applied. In projects where the requirements are well understood and where there are small chances of major changes, the Waterfall development method will be suitable (Sommerville, 2004).

Nevertheless, many people have suggested new methods which could compromise between no planning and too much planning. A goal is to provide just enough time for structuring and planning to get a reasonable payoff. As a solution, Boehm suggested the spiral model.

4.1.1 The Spiral model

One of the key aspects of the spiral model is to help in managing risks (Boehm, 1986; Sommerville, 2004). Boehm proposed that instead of having the steps right after each other like in the Waterfall model, each spiral in the model should represent a phase of the software process. As a consequence, each phase in the process starts with detailed project planning and

identification of risks (Figure 3). The first sector is where *objects and constraints are determined*. After that, the *Risk assessment and reduction sector* is used to analyze the risks and alternative strategies are considered to reduce those risks. Next step is *development and validation.* Depending on the risk analysis from the previous step, the appropriate development model is chosen. In the *Planning sector* the project is evaluated and if it's decided to continue to the next phase, the plans are worked out (Sommerville, 2004).

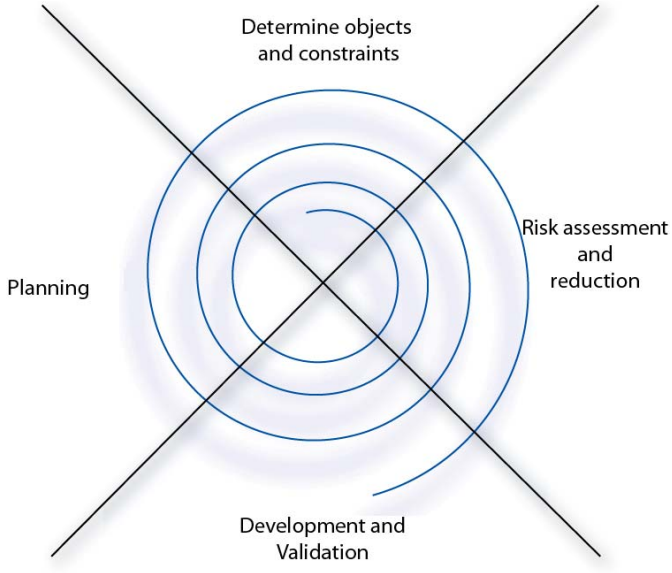

Figure 3: The spiral model

Each spiral in the model can be seen as an iteration. The Spiral model was the first model to explaining the importance of the iteration. The next chapter is devoted to the Agile methodology and its iterative development. One could argue that the Plan-driven bureaucracy of the engineering methodologies together with the creation of the iterative development contributed or appealed to the agile thinking. We enter the Agile methodology for development.

4.2 Agile software development

In the late 1990's several new methodologies began to get increasing public attention. They were proposed in order to cope with the inevitable changes in software development, as well as

changes in the business environment. The Agile methodology seemed like a more adaptive approach to system development as it always expected changes in the requirements, technology and the people involved in the project. In other words, the context was taken into account. An important factor of this flexibility was the transition from heavy reliance on written documentation, to face-to-face (F2F) communication. Written documentation generally required a lot of time and resources to write and read. At the same time it was vulnerable to changes and even the most well written documentation didn't necessarily guarantee project success. The main hindrance was the unpredictable dynamic context of use.

Agile development can be defined in the following manner:

> ***"Agile is an iterative and incremental approach to software development which is performed in a highly collaborative manner with "just enough" ceremony that produces high quality software in a cost effective and timely manner"***
>
> **- Scott W. Ambler (Ambler and Jeffries, 2002)**

Although the Agile methodology relies heavily on the agile manifesto, which we will discuss later, the Agile methodology is mainly based on a few key points:

- Adaptation
- Communication
- Short release cycles

In agile development, the projects are unpredictable (in principle) and the development process must be optimized to embrace change. When using the Agile methodology, making high quality software means a quest for the optimal solution through incremental deliveries and user feedback. The feedback is driven by regular tests and releases of the evolving software (Hayes and Andrews, 2004). This is contradictory to what we have seen in the Plan-driven methodology. Agile development can be seen as crossing a river. When crossing the river, you want to explore when and how to do it in the best way possible.

Another major difference between Plan-driven and agile development is involvement of the customer. Easy access to customers and use of feedback is the Agile methodology's primary control mechanism in contradiction to planning and comprehensive requirements documentation. A common reason to project failure is often lack of *communication*. The customers are often unable to explain what they want and this leads to misinterpretation by the involved participants, especially if the requirements have to be frozen into requirements documentation. If the project process is allowed to go along without constant feedback and communication with the users, there is a chance that the software might end up differently than intended. The odds for these kinds of problems are reduced by using the Agile methodology. Close communication with customer will ensure that misunderstandings are prevented in an early phase of the development (Beck, 2004).

A common agile credo is "release early, release often". Small releases or working prototypes is a great way to get feedback from the users. This evaluation also gives an indication of the system "fitness" as well as an overall view of the project progress (Cockburn, 2002).

4.2.1 The Agile Manifesto

The Agile cult or Agile Alliance is driven by the "Manifesto for Agile Software Development". The Agile Manifesto is a statement of the principles concluded by 17 methodologists that underpin Agile Software Development.

1. **Individuals and interactions over processes and tools**

   This value addresses the fact that regardless of what kind of tool one uses it is still up to that person to use it correctly. The most important factor is people and how they work together. To have the right tools and processes is important, but collaboration between the team that uses those tools and shares the processes is crucial. The time margin between the time a question is asked and a satisfactory answer is given should be as close to zero as possible (Hayes and Andrews, 2004).

2. Working software over comprehensive documentation

A well known proverb says: A picture says a thousand words. Often customers prefer to see what the software or the system will look like and what functionalities it will have instead of reading a lot of pages describing it. In other words, this principle refers to producing the software and showing it to the customer. And often customers will have a significantly easier time understanding the software that is produced, than they will from complex technical diagrams describing its internal workings or abstract descriptions of its usage (Ambler, 2004; Hayes and Andrews, 2004). However, documentation has its place. It can be written as a software manual, maintaining manual or as an expanded process. It can even be partly automated by tools such as Java Doc. Written properly it is a valuable guide for the users understanding of how and why a system is built and how to work with the system (Cockburn, 2002). If creating documents was the primary goal of the development process, then it would be called document development rather than software development (Ambler, 2004)

3. Customer collaboration over contract negotiation

Only the customer can tell you what they want. Often, customers do not have the skills to exactly specify the system up front. The customers may even change their minds and go for some other solution. Likewise, developers might think they know what the customer needs are. Regardless of the situation, collaboration between developers and customers is crucial.

> ***"There is no "us" and "them", there is only "us""***
>
> **- Alistair Cockburn (Cockburn, 2002)**

To have a written contract between the involved parties is important. The understanding of everyone's rights and responsibilities may form the foundation of that contract, but a contract isn't a substitute for communication (Cockburn, 2002; Ambler, 2004)

Developers that are successful often work closely with their customers and they invest the effort to discover what their customers need. At the same time they educate their customers along the way, which they generally appreciate a lot (Ambler, 2004)

4. **Responding to change over following a plan**

   This principle corresponds to the fact that changes can and will be made when following a project plan. Regardless if it is the technology, stake holder's understanding or business environments that change, the software and the project plan should have room for those changes and to be distensible (Cockburn, 2002; Ambler, 2004)

The Agile manifesto in itself is considered to be a guideline for developers everywhere when practicing the Agile methodology. However, the guidelines alone are a somewhat vague to be considered techniques that can be implied directly to software development projects. The Agile methodology also consists of a number of different methods that all correspond to the Agile manifesto. In the next chapter we will describe some of the most common methods.

4.2.2 The process of agile development

The agile methodology is based on *iterative* and *incremental* development (Sommerville, 2004). In our case iterative and incremental development are so intertwined that we refer to it simple as iterative development. Iterative enhancement contributes to develop a software system incrementally. People do mistakes and that is why the iterative method was invented (Cockburn, 2002). It allows the developer to take advantage of what was being learned during the development of earlier, incremental, deliverable versions of the system. The process starts with a simple implementation of the software requirements and iteratively enhances the evolving versions until the full system is implemented (Sommerville, 2004). Each iteration bears some changes done on the software system and leads to an executable release that may be presented to the customer. This process is illustrated below (Figure 4).

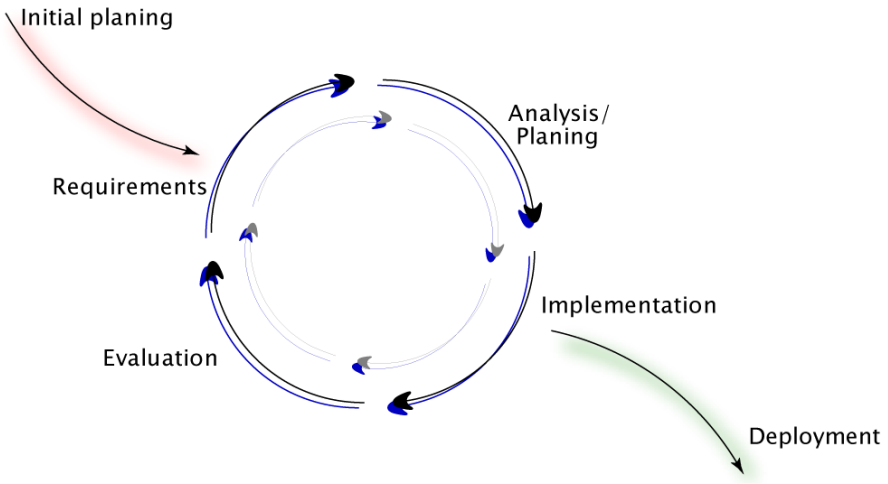

Figure 4: Iterative and incremental development

4.2.3 Agile methods

All agile methods have something in common; they deliver working software quickly to customers so that they can comment on it and propose new functionalities or to change existing ones in later releases of the system (Sommerville, 2004). This means that all agile methods are highly adaptable and, if we compare them to the methods of the Plan-driven methodology, they are all considered lightweight. However, all methods have their limitation, and agile methods are only suitable for some types of system development. Agile methods are not well suited for developing large-scale systems where the teams are placed in different geographical locations and where interaction with different hardware or software systems is complex. Similarly, it's neither recommended to use agile methods in the development of critical systems where a detailed analysis of the system requirements is very necessary to understand their safety. I.e. air traffic control systems (Sommerville, 2004).

However, agile methods are recommended when working on projects where some exploration is needed. The customer might not know what they want until some prototypes has been developed and demonstrated. At the same time, the developers might need to do some research to discover what solutions that is actually possible to reach.

The various agile methods all comply with the Agile manifesto. Even though they have many things in common, they also have strengths and weaknesses that should be taken into consideration when choosing a suitable method. Below is a list of the most common agile methods.

Scrum

Scrum is a framework for managing and controlling iterative projects. Scrum refers to the mechanism used in rugby for getting an out-of-play ball back into play (Highsmith, 2002). Development is divided into one month *Sprint* cycles. Each *Sprint* cycle contain a *Product Backlog* over features that have to be developed that month. Scrum's strength lays in the daily 15 minute team meetings where developers have to answer three questions.

1. What have you accomplished since the last meeting?
2. Are there any obstacles in the way of meeting your goal?
3. What will you accomplish before the next meeting?

Answering these questions leads to increased team coordination and communication, as well as constant focus on today's tasks only. However, it also leads to stress on the developers as there is a constant pressure of achieving the goals every day.

Dynamic Systems Development (DSDM)

DSDM is a framework that is based on Rapid Application Development (RAD). DSDM is useful in projects that are characterized by tight schedules and budgets. It focuses on the common reasons for project failure such as exceeding budgets, missing deadlines and lack of user involvement. The nine principles of the DSDM include active user involvement, frequent delivery, team decision making, integrated testing through the project life cycle, reversible changes and communication/cooperation among all project stakeholders (Hayes and Andrews, 2004).

Lean Development (LD)

LD is based on the principles of the Japanese lean production methodology, especially in the automobile industry in the 1980's. LD focuses on delivering high quality software to the customers while carefully optimizing all artifacts involved in the process. The main principles of LD include: eliminate trash, empower the team, build quality software, deliver fast, optimize and create and retain knowledge (Hayes and Andrews, 2004).

Crystal Method

The Chrystal method is a people centric approach to agile development created by Alistair Cockburn (Cockburn, 2002). This method mainly focuses on building a team where each member contributes in the best way possible. Well organized teams led by good management will decrease paperwork as well as bureaucracy and help to focus on communication, personal satisfaction and use of individual skills. The process is important, but secondary (Cockburn, 2002)

Feature Driven Development (FDD)

FDD was proposed by Peter Coad and Jeff de Luca (Hayes and Andrews, 2004). FDD consist of five (minimum) activities. The first activity is to develop an overall model of the system. By decomposing the system functionalities, the next activity is to identify a list of the features. This list gives a base to produce a development plan. After that step four and five is design and building of the features.

Extreme Programming (XP)

XP is perhaps the best known and most widely used of the agile methods (Sommerville, 2004). In our work with the master thesis, we have focused largely on XP and we will present it in more depth below.

4.3 Extreme Programming

XP was created by Kent Beck in the late 1990's (Hayes and Andrews, 2004). A typical characteristic of XP is that it focuses on adaptability rather than on tedious long term planning (Beck, 2004). In today's fast paced business market changes happen rapidly. It's not just customer requirements that change, but also fast growing technology as well as frequent change of personnel. Instead of trying to hold back or avoid the inevitable changes, XP is designed to *embrace the changes* that typically occur in software development (Beck, 2004).

Although XP is considered to be extremely flexible in handling changes, the method itself is based on strict values and practices that must be followed in order to succeed. These principles go hand in hand with the principles of the agile methodology. They are referred to as:

- Communication
- Simplicity
- Feedback
- Courage

When practicing XP, *communication* is considered to be a major problem solver. Problems that occur during the development are rarely that complex and often some individuals already have an answer. However, lack of communication may make these persons hard to find. This may slow down the project development. Another aspect is *simplicity*. This value refers to the fact that simple solutions always take less time to implement than complex ones. In other words, always do the simplest ting that could possible work first and don't be concerned with tomorrow's possible requirement for further development and flexibility (Beck, 2004). Inevitable changes lead to need of *feedback*. When following the practice of *simplicity* and short terms cycles there will always be room for receiving *feedback* from users. By receiving *feedback* the developers will get closer and closer to an adequate solution. In order to reach this adequate solution is important to show *courage*. *Courage* connects previous mentioned values. By having the *courage* to express one's views and opinions within the team, *communication* and trust is improved. This will also lead to an increasing amount of *feedback* as both the team and the

users will engage in creative discussions on further development. Furthermore, being able to discard failing solutions at an early stage as a result of user feedback also leads to simplicity.

4.3.1 The process of XP

XP inherits the model of iterative development (Beck, 2004). Customers are involved in specifying and prioritizing system requirements. These system requirements are not listed in a long documentation list. They are rather described in the form of *user stories*. *User stories* are worked out by a team of future users and developers, which mean that the customer is part of the development team itself (Sommerville, 2004). Notice, however, that the users should be the most active participants during this process. *User stories* basically encapsulate the customer's needs in a language everyone can relate to, and describes a desired working situation for a given functionality. *User stories* give basis for the development plan. This development plan is broken into releases and again into iterations, before the developing can begin. The development team then aims to implement that the *user stories* in future releases of the system software.

When the *user stories* are produced, they can be transformed into tasks and developers can estimate the effort and resources required for software implementation. This helps them to make a *release plan*. This plan consists of the customer's desired features to be implemented as well as the developer's estimation of difficulty, time to be used and resources needed (Hayes and Andrews, 2004). The next stage is a development process followed by integrating and testing with rest of the software. Releasing and evaluating the software with the rest of the system is the final step in XP's release cycle before starting the next iteration cycle (Sommerville, 2004). XP's release cycle is illustrated below (Figure 5).

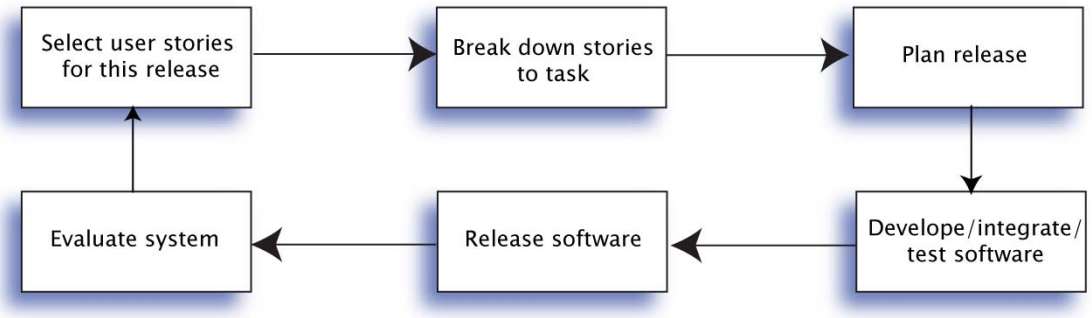

Figure 5: XP's release cycle

Here customers can choose in which order the *user stories* should be implemented. They will then be able to see the progress made and be able to use those parts of the system software that are implemented (Cockburn, 2002; Hayes and Andrews, 2004). Based on that, the customer's view may change and the customer can request that the unimplemented *user stories* should be changed or discarded. This is much cheaper than changing the program code. However, an already delivered system can also be changed, but then new *user stories* should be developed first (Sommerville, 2004).

4.3.2 XP practices

To be able to follow the values and principles of agile methods, Extreme Programming involves a number of daily practices. These practices are divided into four categories (Wells, 1999) which we will describe here:

Planning:

- *Small releases* – by incrementally adding small features to already existing releases the users are able to get hands on experience with the system and evaluate it. There are some benefits to this: The customer sees the progress made which gives them observable value and the developers get valuable feedback from the users. In addition to this, the cost of making changes is reduced.
- *Incremental planning* – this practice refers that fact that there is no shame to change the requirements or the design during the development. This contradicts the traditional Plan-driven methodology where the focus usually has been on designing the system first

and then starting the development process. Incremental design focuses on designing only what is needed for a specific part of the system.

- *Rapid feedback* – as mentioned above small releases contributes to rapid feedback from the users. By doing presentations or releasing working parts of the system on a regular basis the users and the developers can collaboratively discuss the progress and ensure that the desired system is made.
- *Embrace change* – helps the developers to learn how to further develop the system in the right direction while the customer is gaining knowledge of how to utilize the system in a way that will increase the business value.
- *On-site customer* – An on-site customer available for requirements clarification and business decisions that should be made by the developer (Deursen, 2001).

Designing:

- *Simple design* – changes will happen and therefore is important to have a design that is as simple as possible. Simple design will support the frequent changes that will arise when using XP. This practice also serves other purposes. Simple solutions make things easy to understand and it avoids the developers from creating flexibility that is not needed. System maintenance will also be easier if this practice is followed.

Coding:

- *Coding standard and collective ownership* – by following coding standards, the code will look like it has been written by one person. This will make it easy for any developers involved in the team to change a given piece of the code. This leads to spreading the collective knowledge and ownership of the code. Hence, the dependency on key individuals is lowered. Thus, reducing the risk of project failure.
- *Pair programming* – this practice is elaborated below.

Testing:

- *Unit testing* – the goal of the unit testing is to test each part of the code individually to verify its correctness before is added to the rest of the system. This testing should be written before the code. By doing this, testing becomes a design exercise rather than tedious bug finding after release (Wells, 1999).

These practices are not exclusive for XP. Other agile methods use them as well, but XP takes them to the *extreme level* (hence the name "Extreme" Programming). For instance, code review is unavoidable in other development methods, but when using the pair programming practice (elaborated in the next chapter) the code is always reviewed by two developers – all the time (Hayes and Andrews, 2004).

4.3.3 Pair programming

Pair programming is commonly practiced in XP development. It has received both praise and critique. But first of all what is pair programming?

When two people produce programming code by using one computer, one mouse, one keyboard and working together on the same task, it is called Pair Programming (Hayes and Andrews, 2004). This programming practice can be described by the "auto rally" example where one is driving the car – the driver, and the other is navigating – the navigator (Figure 6).

Figure 6: Pair programming - one is driving, while the other is navigating

With Pair programming, the person who has control of the keyboard is the driver while the navigator is continuously reviewing the work. It is worth mentioning that the development does not always involve the same pair of people working together. It is a common practice that pairs are created randomly so that all team members may work with other members during the development process.

Pair programming has a number of advantages. The most important of these are:

- *Higher quality programs* – this means when several developers overview the software it is constantly rewritten to improve clarity and structure, also called refactoring (Sommerville, 2004).
- *More enjoyable working situation for programmers* – one take initiative when the other is stuck, thus lowering frustration (Beck, 2004; Hayes and Andrews, 2004) and people enjoy their jobs more when they can do quality work, leads us back to previous point (Beck, 2004).

- *It support the idea of common ownership and responsibility for the system* - responsibility for resolving the problems and task relies on the whole team and not on one individual (Sommerville, 2004).
- *Effective learning* – When two people are working together, collaboration and communication will lead to the sharing of knowledge during the development process (Beck, 2004).
- *Higher retention of people* - less margin for the programmers to quit or drop the project if they enjoy working together (Herzog, 2005).
- *Successful in discovering a high percentage of software errors* - one line of code is looked by two programmers.
- *Lower "truck-number"* - less dependency on individuals (Higher number of individuals know about each part of the system) (Herzog, 2005).

Working in pair doesn't mean that you have to do all the work as a pair. Each person may spend some time alone to come up with solutions to a specific problem. However, when implementing the solution, pair programming requires both the participants to sit together when composing the code (Beck, 2004).

Several studies have been done on XP and pair programming (Herzog, 2005). One of the most common misconceptions surrounding pair programming is that it only produces half as much code as two individuals would separately and is hence considered by some to be less efficient. However, studies (Williams, Kessler et al., 2000) shows that the productivity in pair programming and that two people working independently is actually quite the same. Their argument is that *pairs discuss the software before development*, which gives them less false starts and rework. And also the discussion takes place in the actual context of the problem, not in a meeting or by the water dispenser. Another argument is that more errors are captured and fixed before the software reaches the testing phase. This contributes to less time spent in the test phase since errors are more difficult to spot when the whole system is present. Another study done at UiO, showed that pair programming (among students at UiO) increases the enjoyment and confidence while programming, especially for women (Herzog, 2005).

However, there are circumstances where PP may give negative effects. In (Dick and Zarnett, 2002), the writers emphasize that people developing in pair has to have certain personality traits:

- *Communication* – without having the tight communication between developers will lower the efficiency regarded to discussions, decisions, strategies etc.
- *Comfortable* – being comfortable to work with other member it's not only a manner of feeling "free" to express your suggestion and ideas, but also being comfortable working with individuals that have different work ethic.
- *Confidence* – being confident about own abilities reflects on the developing team.
- *Compromise* – confidence is important, but a developer who is too confident may lack the ability to compromise with his or her partner. If the developers have different opinions, they should be able to compromise in order to implement the best possible solution.

4.4 Software prototyping

Prototyping is related to human computer interaction (HCI), participatory design and agile development as it requires both user representation and rapid cycles of iterative development. Prototyping is a way to visualize the system in the early phases of an evolutionary development process. By producing running systems with little content and functionality, it provides a background for discussion between users and developers. This contributes to a mutual understanding of what the system will look like and how it will behave. As changes are frequently made and taken into consideration when producing code, prototyping also allows developers to experiment with different solutions throughout the phases of the evolutionary process (Budde, Kuhlenkamp et al., 1992)

A typical circle of events in software prototyping consists of *requirements gathering*, *prototyping* and *user evaluation* (Sommerville, 2004). Design workshops or other kinds of requirements gathering methods may not express the customers' needs appropriately. Alas, the first prototype may just be presented as a mock-up to help the customer visualize the directions the

development is heading. Through user evaluation the customer can give feedback which leads to new requirements in the second phase of the development. This gives the customers the possibility to see potential changes in design decisions before the system is complete.

Software prototyping is known to improve the communication between the customer and the developers. Early cycles of mock-up demonstrations and feedback, results in far less costly last minute changes as well as a substantial reduction of risk for delays at the end of the development process.

4.4.1 Different kinds of prototyping

(Floyd, 1984) introduces three goals of prototyping; exploratory, experimental and evolutionary prototyping. The exploratory prototyping approach is normally used when the problem is unclear. Initial prototypes may in this situation be used as a base for discussions between the involved parts. With this approach, which is common in the early stages of development, it's important that the designers present various solutions to prevent the rejection ideas prematurely. Exploratory prototyping is often a useful tool if the customers are from a different domain than the designers. When this is the situation, prototypes may be used as a visualization of how the future system might look like. Exploratory prototyping can determine the adequacy of the proposed system before full-scale implementation of the system (Tsai and Weigert, 1989).

Experimental prototyping is used to allow the designers to experiment with the technical limitations. This form of prototyping focuses on the technical side of the development. The technique may be used if there are uncertainties surrounding the technical platform used for development. Experimental prototyping can also be used for creating a simple, functional prototype that simulates the main flow in a proposed system.

Evolutionary prototyping is an approach that allows a system to be developed gradually. By slowly gathering a consistent set of requirements, the software increase in quality as clarification of requirements or discovery of new requirements are done along with the process of developing the prototype (Carter, Ant et al., 2001). With this approach, the development process is no longer looked upon as a single encapsulated project. It is rather a process that

continuously must fit into rapidly changing organizational constraints. Evolutionary prototyping aims at short development cycles and moves the designers into the role as technical consultants who work in close cooperation with the end users.

Another aspect of prototyping is the use of layers (Mayr, Bever et al., 1984). Software development may be looked at as the implementation of several layers. The top layer consists of user interface, while further down we find the functionality and the underlying operating system or database. In *horizontal prototyping*, the designers are concerned with only one layer. Designers of human computer interaction may work on completing the user interface to give the users an impression of what the finished system may look like. At the same time, other developers may focus on functionality and ignore the user interface to give the users a preview of how the system will behave. In *vertical prototyping* the designers focus on a selected part of the system. To achieve this, the designers concentrate on all layers from top to bottom to finish a single part of the system. This technique is often useful when designing a pilot for a project where user feedback and trial and error during the development phase is crucial.

5 Programming framework

In order to achieve a better understanding on how the prototype is developed, we will in this chapter describe tools and technologies that surround, and had impact on the prototype.

5.1 Open source programming

In our work with the research for this master thesis we have used an open source solution as a tool. In this chapter we will describe exactly what using open source software entails in a practical setting.

Open source in the software engineering world, generally refers to a piece of software in which the source code is freely available to the general public and hence, can be used or modified in any way desired. Open source projects are usually born because a single user or a small group envisions a piece of software that serves as a potential solution to solve a problem. Open source programming is often done in a collaborative manner where a number of programmers improve the code and share the changes within the community. The community is often just an online space where the programmers can collaborate from all over the world. By posting small releases or prototypes online, others can see the results of the work and the project may get increased attention from even more users who may contribute in improving the code. (Raymond, 2001)

The Apache web server is probably the most used open source software today. According to the Netcraft study (Netcraft, 2007), Apache is the most widely deployed web server on the internet. Other examples are OpenOffice and the Mozilla applications, which has become popular due to their integration with Microsoft's environment. This means it easier for "regular" user to make the transition from the closed source programs to the open source alternatives (Wheeler, 2004).

There are many advantages to using open source software. A lower software cost is the obvious advantage as the "Open Source Definition" states that:

> ***"Source code must be distributed with the software or otherwise made available for no more than the cost of distribution"***
>
> **- Steven Weber (Weber, 2004)**

But according to (Williams, Clegg et al., 2005), there are several other reasons to choose open source solutions over proprietary systems:

- *Lower software costs* - Open source software generally does not require a licensing fee, and hence, can be obtained free of charge. This can contribute to cut down on a company's capital outlay. The only expected expenditure would be support, but this not always necessary.
- *License management* - When obtaining open source software, the user is free to install the software as many times as desired. Also, installing the software does not come with a need of monitoring a specified number of licenses.
- *Modification* - The software can be modified in any way to fit the user's need. The modified version can also be redistributed which leads to further improvements of the software and growth of the community. However, this does require that existing user check for updates and in many cases downloading and installing new versions manually.
- *Evolving* - The communities surrounding the open source projects are usually far more active than those of standard out of the box software. This leads to rapid evolvement of new features and security fixes.
- *The user is not tied to a single vendor* - When buying off the shelve software the user is "locked" to a specific vendor. This means that the user is dependent on a single vendor to maintain, update and support the software. This is often costly and may just be supported for a short period of time. Open source projects however, relies on multiple sources within the community.
- *Documentation* - Open source software is often well documented, and in many cases superior to proprietary solutions.

However, there also some disadvantages in relying on open source solutions (Williams, Clegg et al., 2005):

- *The project may not live forever* - Open source projects are dependent of the community supporting them. If the developers lose interest, the user may be stuck with software that has flaws and security bugs. Solving these problems usually means that the user have to pay someone to do it.
- *Support issues* - For some companies, support is mandatory. However, the open source community does not have any obligations in solving the users' software problems. In many cases the user has to go online and search for a solution. Commercial software usually guarantees that the product will be supported at least for a given period of time.
- *New features* - Posting ideas or suggestions for improvements to the community may not lead to software updates. If the user is unable to add new features the solution is usually to pay someone to do it. This can be costly, and it may prove to be cheaper and less time consuming to rely on commercial software.

5.2 Component based development

Component based development (CBD) focuses on building software systems by integrating previously existing software components. It's based on the concept that the functionality of systems ought to be encapsulated into components. Decomposition of a system into modules has many advantages. It cuts production time as modules can be implemented by separate groups simultaneously. It also leads to increased product flexibility and ease of change, as first suggested in (Parnas, 1979).

CBD, sometimes also referred to as Component based software engineering (CBSE), emerged as a modern concept of software development in the late 1990's (Sommerville, 2004). The idea was that developers could reuse and integrate already existing components into their software. Its creation was motivated by the frustration over the object oriented approach's (OOP) lack of possibility to be reused. Another factor was that programmers began to look at their previously created component and realized that by doing minor alterations they could be reused. On the

other hand, standalone classes, as applied in OOP, were too complicated to reuse and also required extensive knowledge of the source code (Sommerville, 2004). CBSE require minimal knowledge of the code within the components. Instead, the component can be looked at as a black box. The developer only has to know about its external specification, such as input/output parameters and method calls, and hence, does not need to spend much time on understanding the inner workings of the component.

5.2.1 Advantages of component based development

The advantage of using CBD is two-fold.

1. Developing and modifying standalone components, small application and application units is less error prone than working with an entire application
2. Standalone components can be reused in other applications or frameworks when properly documented

The first advantage is related to the fact that the process of developing software should be derived into smaller pieces. In the context of application systems, these have been referred to as "Application units" (Mørch, 1995). Although this is an idea related to OOP in general from its first conception (Dahl and Nygaard, 1966), CBD takes decomposing of the programming code one step further. Notice that a component is part of a framework, rather than a standalone application. This is a considerable advantage as creating or modifying existing code of a single component will be less critical than changing the overall framework. Modifying a smaller amount of code will also make the process less error prone (Mørch, Åsand et al., 2007).

Encapsulation or black boxing of components means that the internal workings of the components are unknown to the other components. The developer only needs to know about the functionality (must be well documented) and how it interfaces with its surroundings (Sametinger, 1997; Ravichandran and Rothenberger, 2003).

Other advantages are related to the reuse of components (Griss and Pour, 2001). Reusing components while creating software can significantly reduce the development costs. This is mainly because components usually are built after well defined specifications and it's relatively

easy to assess whether or not a component can be reused to build another system or application. Reusing existing components, rather than building them from scratch will save time during development. Furthermore, it also means that developers are dealing with components that already have been implemented and tested (Griss and Pour, 2001).

5.2.2 Disadvantages of using component based development

First, reuse of a single component even without any changes, does not come free. The developer has to locate the appropriate component, understand the interface and figure out how to fit it into the context of the system under construction, which sometimes may require making changes to the component itself (Pree, 1997). Notice that in order to find an existing component; the developer has to attain a catalog of all the existing components. Reuse of components also requires a fair amount of programming understanding. Developing components that are meant for reuse also require more time and skills than creating a component designed for one specific purpose (Pree, 1997).

Reuse of components that does require the developer to make changes, is not as lucrative as it might seem. According to a study done in NASA's engineering lab, only a few minor changes raises the reuse cost to 55% compared to a similar component created from scratch (Pree, 1997). Although the cost is still close to being cut in half, major changes are likely to increase the cost even further.

When relying on ready-made components, the developer has to consider the component's trustworthiness. If the component is black-boxed and the source code is unavailable the developer must be particularly careful. The component may contain undocumented failure modes that cause the software to act unexpectedly. In a worst case scenario the component may even include a Trojan horse or security holes that can threaten the security of the entire system (Sommerville, 2004). In addition to this, programmers rarely feel comfortable with applying software written by others, also known as the *Not-invented-here* syndrome (Pree, 1997).

When considering including standard components into a system, framework or application, the developer must often make a trade-off between desired requirements and the available components. In some situations, this means that relying on reuse of components may lead to building software that does not comply with the customer requirements. If the component is black boxed as well, it might be impossible for the developer to do the required customization and the best solution may be to build a new component from scratch.

5.3 Visual Studio

Microsoft Visual Studio is an integrated development environment created to allow developers to design different kind of applications and services that run on platforms supported by Microsoft's .NET application framework. These platforms are: Microsoft Windows servers and workstations, PocketPC, Smartphones, and World Wide Web browsers. While these services and applications can be written for different platforms, they can also be written in the programming languages that the developer is most comfortable with, most notably C++, C#, J# and Visual Basic.

Visual Studio is a feature rich application that eases the process of developing software, especially for the web. It has a powerful WYSIWYG (What you see is what you get) design surface that allows for easy drag and drop of elements (Figure 7).

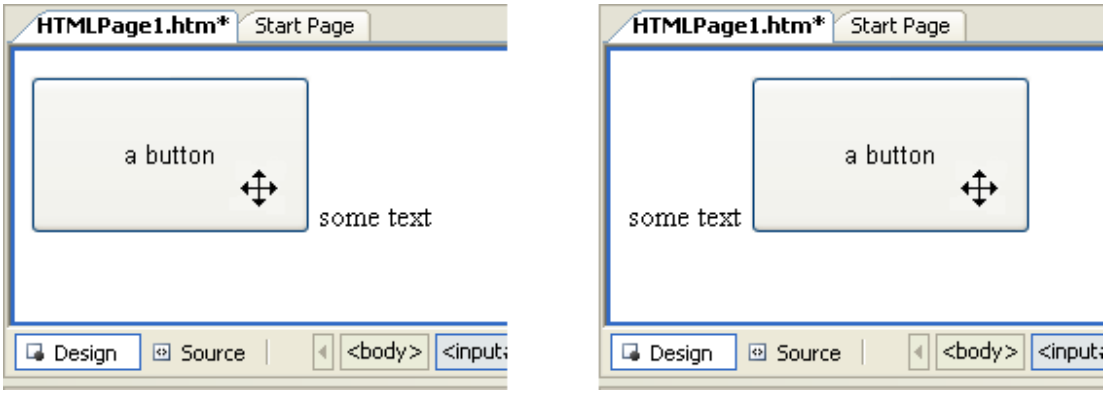

Figure 7: Drag and drop function in Visual Studio

It also features a large toolbox that can be applied to all projects, regardless of what platform you are developing for. The toolbox is divided into intuitive categories and can even be expanded with tools downloaded from the internet (Figure 8, panel 3). To allow the user to save time during development, Visual Studio has implemented a feature known as *IntelliSense*. This feature provides statement completion while typing. It quickly accesses variables, methods and events. This prevents typos and unnecessary scrolling to find the names of existing variables, methods and properties. Visual Studio also supports the use of external plug-ins. This allows programmers to download small pieces of code that can be used as a framework for further development. As we shall see later, we applied a plug-in that eased the development of modules for the web portal framework DotNetNuke.

While developing, the user can organize the workspace as desired. The workspace is divided into different movable “panels” that all serve different needs. The Solution Explorer (Figure 8, panel 1) shows the current project the developer is working on. The panel is organized as a tree view familiar to most users. The tree displays the files available under the current project. The large area in the middle (Figure 8, panel 2) is the work area for the current file. At the top of the panel there are some tabs. These tabs allow the user to have multiple files open at the same time. At the bottom there is a button for Design and one for Source. These buttons allow the user to switch back and forth between Design mode and Source code mode. When developing web applications this is a huge advantage as the user can choose between editing the interface and the code behind. Because this panel is most frequently used, it can be expanded to cover the entire screen. As mentioned above the Toolbox panel contains the most common tools related to the current project (Figure 8, panel 3). When working with web based development these are typically content related to forms, but also database connections or other features that can be placed within a web site. The last panel is used to display errors when debugging (Figure 8, panel 4).

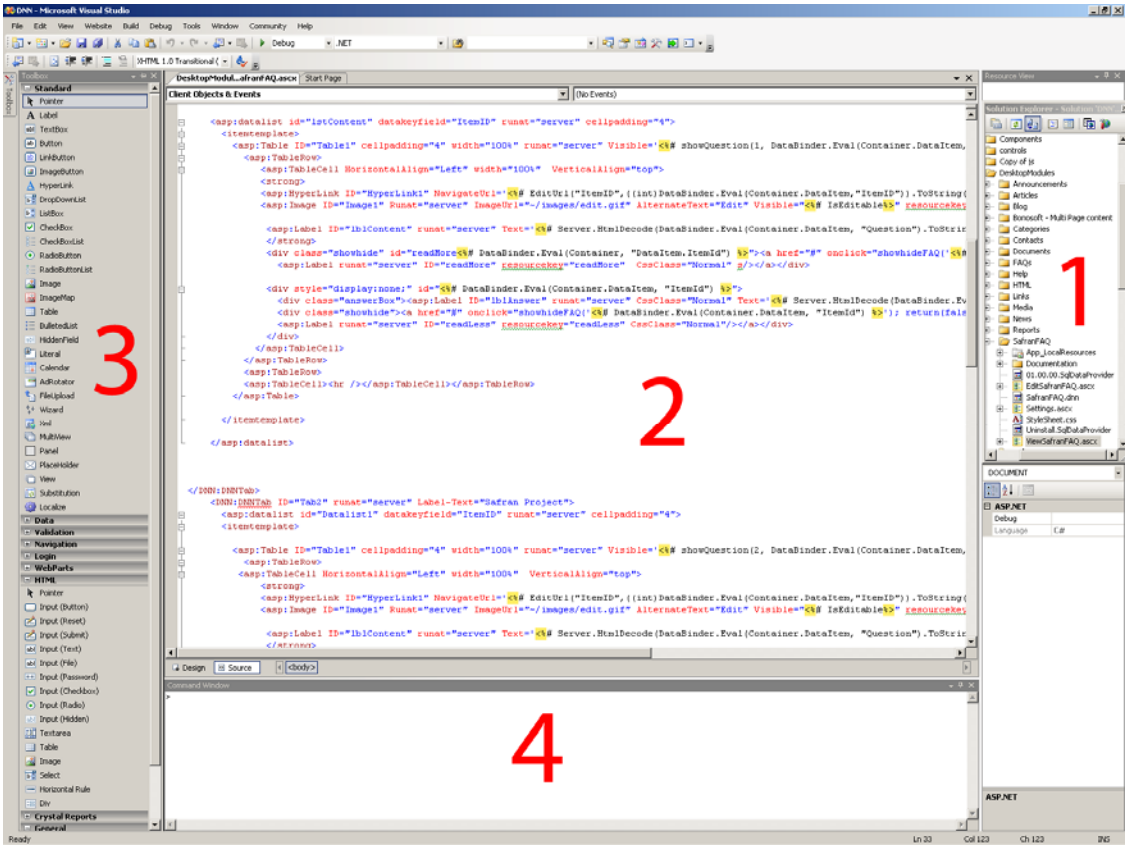

Figure 8: Screenshot from Visual Studio

5.4 Web 2.0

During the 1990's when internet became popularized, web sites typically consisted of static information controlled by the publisher. Although some web sites grew into pages with dynamically generated content loaded from a database, the communication was usually from server to client without much response required from the user. However, as the technology and tools became more advanced, as well as a demand for something more sophisticated, the term Web 2.0 was coined.

The pioneers of web design can only agree on one thing when trying to define Web 2.0; it is not possible to pin it down to a single definition (O'Reilly, 2005; Treese, 2006). However, there are some keywords seems to be common when explaining the success of Web 2.0. The most dominant factor is *interactivity*. Web 1.0 was based on one-way interaction between the

publisher and the reader. For instance, a news site would published news that the visitors to the site would read. With Web 2.0, users are allowed to interact with the web sites. Some of the most popular web sites today are based on user generated content. This means that multiple users are interacting with a web site as well as with other users, i.e. it's a 2 way communication where both sides are equally important. For instance, on a Wiki, multiple users collaborate in publishing and updating social content.

By using advanced technologies such as JavaScript, AJAX (Asynchronous JavaScript and XML) and Flash, web developers are taking the interaction one step further. A well known example is Google Maps. Previous web maps only allowed the user to access one piece of the map at the time. Google maps allow the user to interact with map by sliding it around without having to reload the page. The asynchronous part of AJAX works in the background. In Google Maps, AJAX technology automatically downloads the areas surrounding the current map on the screen. This allows the user to visualize the map as one big image and not small, separate pieces of information.

Another key factor to the success of Web 2.0 is *social networking*. Although this is not a technology, it refers to how the web can be used to tie together groups of people. The currently most popular example is Facebook[3]. Facebook is a web service that lets users connect by sharing personal information, photo albums and future plans. Web site built around social networking is radically different from classic web sites as the content in these sites are built on the information every user contribute with.

The third reason for the popularity of Web 2.0 is the use of metadata and *tagging*. By textually labeling information such as pictures or text, the data can be organized and users can easily browse the content of a web site in a different way than previous. When tags are used, similar information can be group together and finding interesting data is now a more achievable task than before (Treese, 2006).

[3] www.facebook.com

5.5 Web portal frameworks

Lately there has been an enormous demand for web based information sites. Many companies seek a solution where they can post information to customers and employees. This can be achieved through the use of a web portal, also known as a Content Management System (CMS). A CMS is a web based portal where content is automatically organized to fit into a given design. This allows anyone to post news and other content into the web portal without accessing the underlying code. Instead the user is presented with a human friendly interface where posting an article can be done through a number of easy understandable steps and then posted directly onto the web portal.

Until recently companies had to hire professionals to create such web portals. This has created a demand for free open source alternatives. These alternatives are often preferred as they generally are more flexible and offer more possibilities. The open source alternatives are so sophisticated that they can be referred to as complete frameworks. This means that the web portal consists of a core with all the main functionality needed, as well as the possibility to add more components when needed. A component in the world of web portals is related to the components we saw in the chapter on Component Based Development. They are small piece of standalone functionality that can be put into a web portal as an independent element.

The open source web portal frameworks can be downloaded and installed on a local server even by users with limited programming skills. Even though they are open source, there is no need to change the code unless you want to insert new components into the framework that doesn't already exist. Even this task can be achieved relatively easy by downloading open source components from the selected framework's web site and installing them directly into the web portal through a human friendly interface.

5.5.1 Web portal alternatives

Because the demand for open source web portal frameworks has had such an enormous growth over the last few years, there has popped up quite a few alternatives. When choosing a framework, the user has to carefully examine the alternatives. Different frameworks may

support various needs and it is vital to pick the right solution for the developer's need. Below are some criteria that should be taken into consideration when choosing a framework.

- Platform
- Community
- Security

Choosing a platform is usually not that hard. The two main options are frameworks that run on the open source web server Apache and frameworks that run on Microsoft's web server Internet Information Services (IIS). The frameworks that run on Apache are most commonly written in the programming language PHP. The IIS frameworks are based on .NET, which supports different kinds of programming languages (See chapter on Visual Studio). Ultimately the choice boils down to whether the user prefer to work with PHP or .NET, or even simpler, the web server the user is already relying on today. PHP and .NET offers a lot of the same possibilities. They are object oriented programming languages, and they are both well supported with tons of books and web sites dedicated to the subjects. Nevertheless, there are some differences. PHP is a programming language made for server side scripting. This means that it can receive requests from the user (the client) and handle them on the server for further computing.

.NET is a programming framework that supports a more elegant infrastructure for developers. It uses layers that separate the user interface (UI) from the underlying functionality (More on this later). However, the principle is still the same as for the PHP language. A user interacts with the UI (the client), and requests are sent to the server side to be processed.

When the choice of platform has been made, the search for the most suitable web portal framework may begin. An open source CMS should have a large group of users. This will ensure a highly stable CMS and a large community to back it up. Below is a list of advantages of CMSes that have a massive user group. The list is inspired by Hanseth and Lyytinen's article on using bootstrapping in designing information infrastructures (Hanseth and Lyytinen, 2004).

- *Future updates* - A large group of users will encourage the producers of the software to make updates. The fact that it is open source also makes it possible for the users to contribute to the software and publish their work for others to use freely.
- *Beta testers* - When a piece of software has many users it is a very high chance that someone runs into problems before you do. This means that most problems probably are solved before you encounter them.
- *Community* - A big community means that you can always go somewhere if you encounter problems. As mentioned above, most of the time your problem is not a new one. Someone has probably solved it already and posted the solution online.
- *Documentation* - An open source CMS with a large community is often well documented. This makes it easier to get started and the most common questions are usually answered here.

Another aspect when choosing a web portal might be security. This is an important issue which should not be neglected, and most of the large open source web portal framework offers at least a minimum of security. With log in functionality, administrator roles and basic prevention against hacking, the web portals are considered safe at least in an everyday setting. Nevertheless, there are some breaches that cannot be covered by the web portal framework no matter how robust it is.

- A web portal runs on a server and in some cases relies on communication with other software. If the server itself or the communication with other software is not secure, the web portal may be vulnerable to outside threats.
- Different types of services open possibility for different types of vulnerabilities. If users are granted access to extra services such as file uploading or running of external scripts, the web portal cannot be held responsible for security breaches.
- Role based security. Usually an administrator role user has little or none hindrances when administrating the web portal. In on other words, the administrator has to know what he or she is doing when administrating the web portal. Keeping the number of administrator users as low as possible means a reduction of the security issue.

These are some of the points that should be taken into consideration when considering security. Notice that these concerns are related to the inclusion of third party software or human mistakes. In our opinion, the largest open source web portal frameworks available on the market today are secure enough for commercial use as long as the administrators are properly trained and act according to the specifications of the software.

5.6 DotNetNuke

Our choice of an open source web portal framework fell on DotNetNuke (DNN) in the work with the master thesis for several reasons:

- The customer had already laid eyes on DNN before we got involved in the project.
- The customer were already hosting an IIS server
- DNN is without a doubt the best supported web portal framework for this platform.

It also complies with the criteria we listed above (Hanseth and Lyytinen, 2004). It has a large community that will ensure future updates as well as documentation and support if the developer gets lost.

5.6.1 The concept of DotNetNuke

DNN is a good tool for creating websites for commercial use. It is open source, and licensed under a BSD agreement (Or Berkeley Software Distribution)[4]. In general, this license grants the general public permission to obtain the software free-of-charge. The framework is built on the .NET platform, and has to be installed and hosted on Microsoft's web server, IIS.

DNN grew out of a project called the IBuySpy portal (IBS) in the early 2000's. The goal of IBS was to demonstrate how .NET could be used to create a dynamic web portal application. At that point, applications like those were only available for the Apache server. In December 2002, Shaun Walker of Perpetual Motion Interactive Systems Inc., released a modified version of the

[4] http://opensource.org/licenses/bsd-license.php

IBS portal (Walker, Brinkman et al., 2006). His version was known as the IBS workshop and was soon picked up by a large amount of interested developers. After a number of rapid releases, the application was renamed DotNetNuke. The name DotNetNuke is a merger between "DotNet", which symbolizes the programming language .NET, and "Nuke", which in the world of CMS'es describes how components can be imported into a preset number of columns in the web portal (Figure 9).

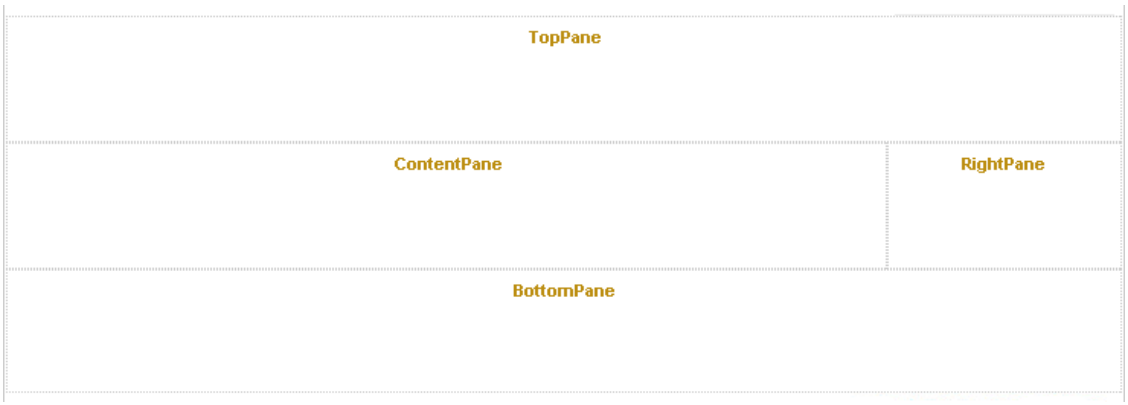

Figure 9: Nuke - Arrangement of columns where the developer can place components

5.6.2 DotNetNuke features

DNN is a feature rich framework that met our requirement when looking for a suitable web portal framework. Below is a list of key features as described on the web site of DNN[5]:

- *Easy to install and to host* - The installation of DNN is done in a web browser. The process guides the user through several steps. These steps do not require any prior knowledge of web portals and is done in a few minutes. The user also can choose to store the data in the various database platforms.

[5] www.dotnetnuke.com

- *Extensible and scalable* - DNN can be suitable for a variety of projects. Because of its structure and reliance on database storage, a web portal will meet the requirements regardless of the size of the project or the number of users. DNN's group policy makes it possible build up either Internet or Intranet sites.
- *Localization* - DNN supports multiple languages, which makes it possible for an administrator to adapt the software to the local language.
- *Community* - The DNN web site has a large community of active users. These are either professional programmers or just simple users. Questions are answered frequently, often in a matter of hours, by DNN enthusiasts.
- *Extension through components* - The DNN framework allows the user to add components (in DNN referred to as modules) (Figure 10). The various modules give the administrator privileges to offer unique web sites that serve different needs. The user interface can be customized by downloading different skins. Skins can be applied to the entire site or to a single module. However, none of these interface changes influence the functionality.
- *User Friendly* - DNN focuses on making the experience as user friendly as possible. Each module has its own help function that describes its use.

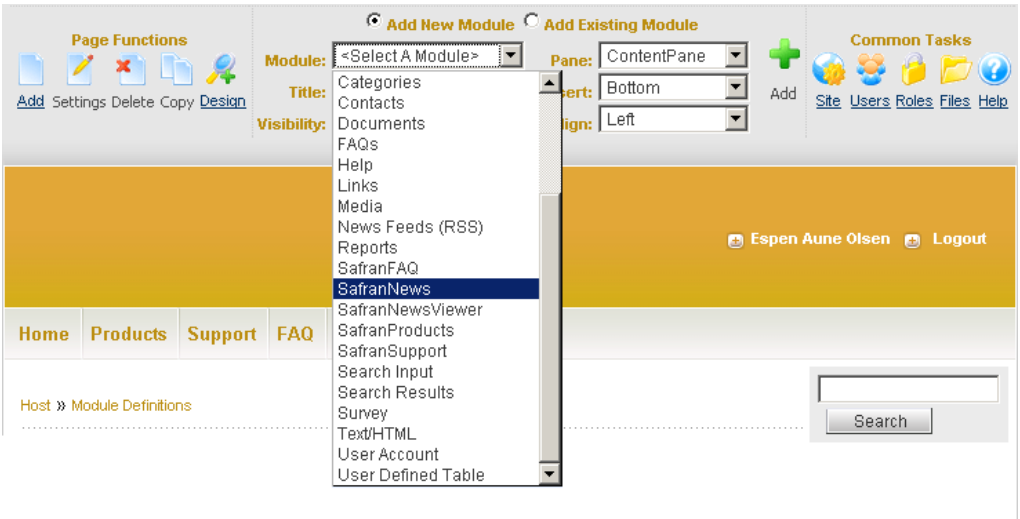

Figure 10: Example of the possibility to extend the functionality of DNN by adding modules

5.6.3 The DotNetNuke Architecture

The DotNetNuke Architecture is traditionally represented using the following graphic (Figure 11).

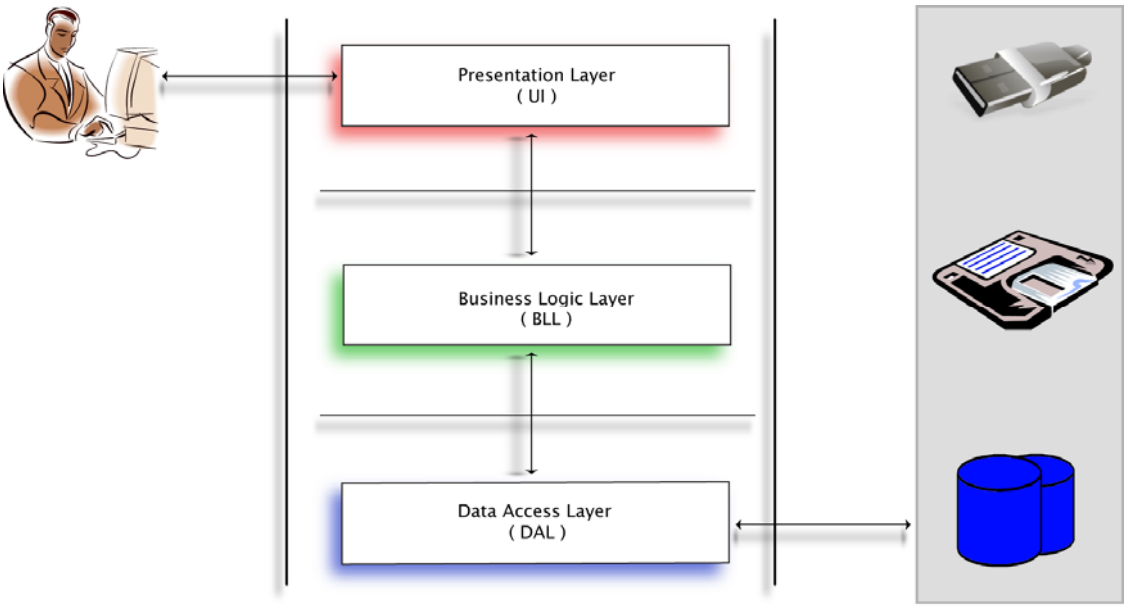

Figure 11: The architecture of the DNN framework

A module consists of 3 layers that each serves different needs. On the top is the presentation layer, also known as the user interface (UI). This layer defines how the module looks through skinning and placement of objects in the interface and how the user can interact with it. When a user does an action in the UI layer, the needed data is transmitted to the Business Logic Layer (BLL). The BLL is the layer that handles the functionality within the module. For instance, when a user decides to click a button or follow a link, the BLL receives a request and determines how to handle it. If the request requires an inquiry to stored data, the Data Access Layer (DAL) is invoked by the BLL. The DAL handles the communication between the module and the data storage. When data is retrieved from the database or some other data storage, it's sent back to

the BLL for further processing. After that it's presented in the UI, and displayed to the user (Walker, Brinkman et al., 2006).

A similar architecture pattern is seen in the Model-View-Controller (MVC). The model is data (for instance, an object) which can be communicated or manipulated by some process. The view transforms the data into a form readable and recognizable by humans, typically a User Interface, while the Controller listens to the user action and access the model (Reenskaug, 1979).

There are several advantages of using this structure. Even though the layers are dependent of each other they don't have to "know" about the other layers' functionality or content as long as they know how to communicate with each other. This increases flexibility as one layer can be modified without having to do any changes to the other layers. In other words, it separates the functionality, i.e. changes in one layer do not influence the other layers. Hence, all three layers of a module can be developed simultaneously.

5.6.4 Modules

As mentioned before, modules are single components that can be put into a framework. The purpose of modules is to implement a piece of functionality into the web portal framework. The modules are not meant to work as a standalone application. However, they are meant to be standalone in the sense that they can be added into and removed from any web portal supported by the DNN framework.

Each module implements the layers described in the chapter on the DNN architecture. This means that each module has a UI, a BLL and a DAL. Hence, a single module consists of all the functionality required to perform a given task. However, it's dependent on the framework to get environment variables and settings that influence the module. Advanced customization also allows two or more modules to communicate with each other. The advantage of using modules in this fashion is the customizability when it comes to the UI. The placement of the functionality is set inside each module, while overall design decisions are done in the DNN framework. With a template for the DNN framework the user can set a specific look that is automatically applied to all the modules.

Content of a module

A module consists of three different views. *View*, *Edit* and *Settings*. Most users will only see the *View* mode of the modules, but as an administrator it is possible to access the other modes through a menu in the module. The *View* mode is the front end of the module. This is where the information of the module is displayed. A news module will typically list the latest news in the *View* mode. The *Edit* mode allows the person who writes the news to access a second view of the module. In this mode the *News writer* can fill out the fields required to insert a news article into the *View* mode of the module. When saved, the article is automatically displayed so regular visitors to the web portal can read the latest news. The *Settings* mode allows an administrator to change the setting of the module. Here it's possible to change the look of the module or set user permissions, i.e. set which users that are allowed to view or edit the module.

Creating a module

There are several ways to attain a module that serves a particular need. It can be downloaded from an online community, build from scratch or customized by editing an existing module. There are several online communities that offer readymade modules. These modules are either free (open source) while others requires you to pay a fee. The size of the fee is often dependent of whether you would like to have the source code included or not. When the source code is included, it means that further customization is possible. As mentioned in the chapter on Component based development, downloading readymade components often require tradeoffs between requirement and what is actually available. In our experience it proved virtually impossible to find the module that fitted our needs exactly. Hence, we often found our self in a situation where we needed to do some customization in order to meet the requirements.

When we were unable to find a readymade module that met our requirements we usually decided to build one from scratch. Building a module entirely from scratch would be tedious and time consuming task. However, a plug-in created by the people behind DNN, gives the developer a full framework required to build a module. The plug-in creates a standard module with a UI, a BLL and a DAL. It also provides a simple example of functionality that allows an inexperienced developer to see how the information flows through the layers of the modules.

Module development in Visual Studio

A plug-in created by the developers of DNN is available from their website. The plug-in is recommended as the best way to get started with module development. It contains all the files required to create a module and it also provides an example for inexperienced developer to get started.

The plug-in, *DotNetNuke_4.3.0_StarterKit.vs*, is automatically installed as a template in the programming environment in Visual Studio. This template can be found by right-clicking on the folder where you want to place the new module in the Solutions Explorer. Selecting “Add New Item” from the pop-up menu (Figure 12) will open another menu.

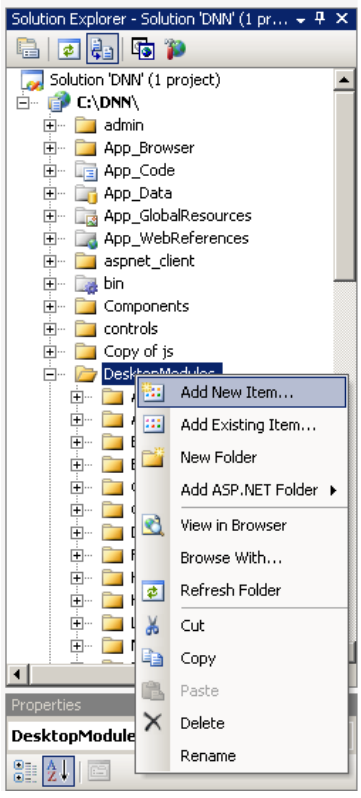

Figure 12: Click Add New Item to add a new module

This window allows the developer to choose a template from “My Templates”. The downloaded template is labeled “DotNetNuke Dynamic Module”. The module template supports Visual Basic,

Visual C# and Visual J#. A drop down menu lets the developer choose his or hers preferred programming language (in our case *C#*) (Figure 13).

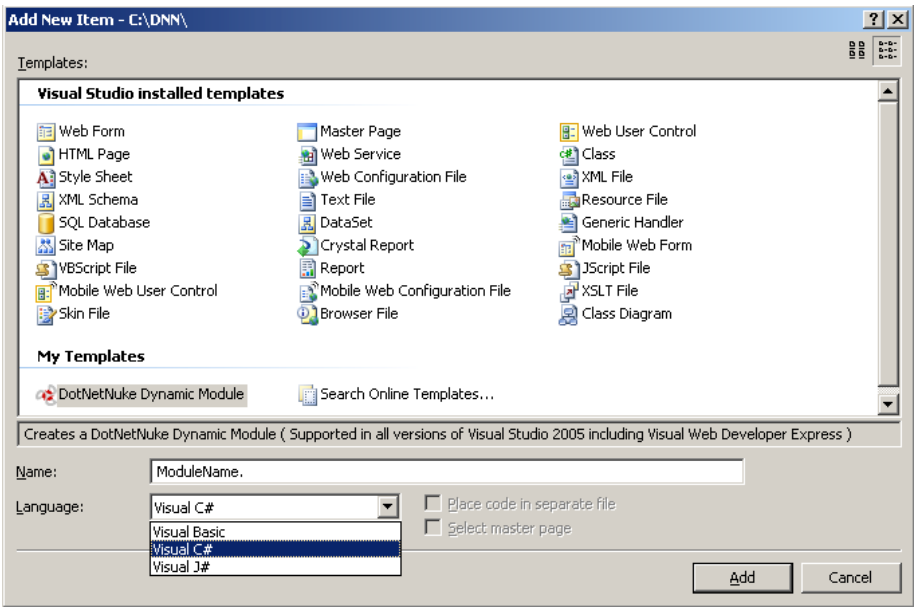

Figure 13: DotNetNuke Dynamic Module template available in Visual Studio

After that it's a straight forward process where the developer has to choose a name for the module and edit a few custom fields in order to get started with the development of a module. As mentioned before, the template provides the developer with an example of use. The easiest way to get started with development is to customize this example into whatever functionality that is needed.

5.6.5 Security roles

DNN offers a fairly robust method for dealing with permissions and controlling the tasks a particular user is allowed to perform (Walker, Brinkman et al., 2006). This is done through role-based security. A module available in administrator mode allows the administrator of the web portal to assign users to certain security roles. These roles can then be used to set permission levels for either entire pages or on single modules. Below is an example of the security role module (Figure 14).

Figure 14: Security roles in DNN

DNN allows the administrator to create as many security roles as desired. When this is done, the administrator can organize the web portal's users into the roles. An example is the role of a *News writer*. First the administrator can create role named News writer. Then the news writers of the web portal can be organized into this security role.

Each module and page of DNN has its own *Setting* mode where the administrator can set the permissions for the available security roles. The view consists of a table of the available security roles. Next to each item in the list, there is a checkbox for *View* and one for *Edit*. Checking these boxes will allow that security role to either view or edit (or both) the module or page (Figure 15).

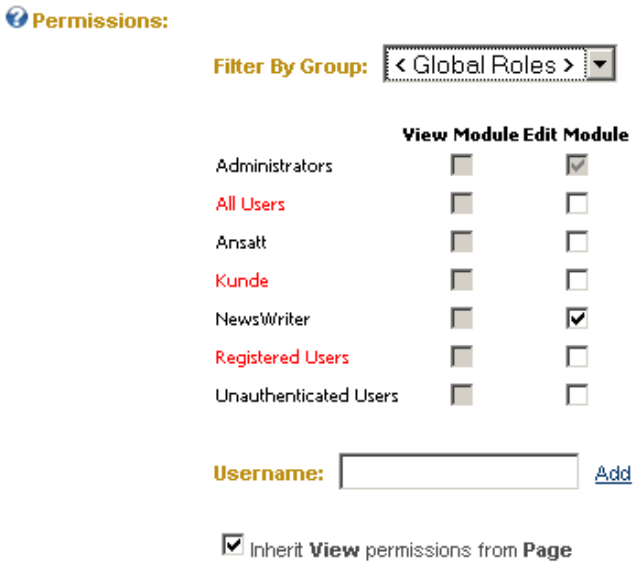

Figure 15: Permission setting in a module in DNN

5.6.6 Support for multiple languages

DNN supports the use of multiple languages. Installation of a second (or as many as desired) language is quite simple. The language configuration module in DNN allows an administrator to add another language by picking it from a drop down list. When chosen, the language is automatically installed and ready to use in the web portal (Figure 16).

Figure 16: Making another language available to the users of the web portal

This means that a user visiting the web portal can choose whatever language he or she prefers from a drop down list of the installed languages. When a different language is chosen, the menus of the web portal are translated to a second language (Figure 17). However, the content of the module still remains unchanged unless it's programmed to display different languages.

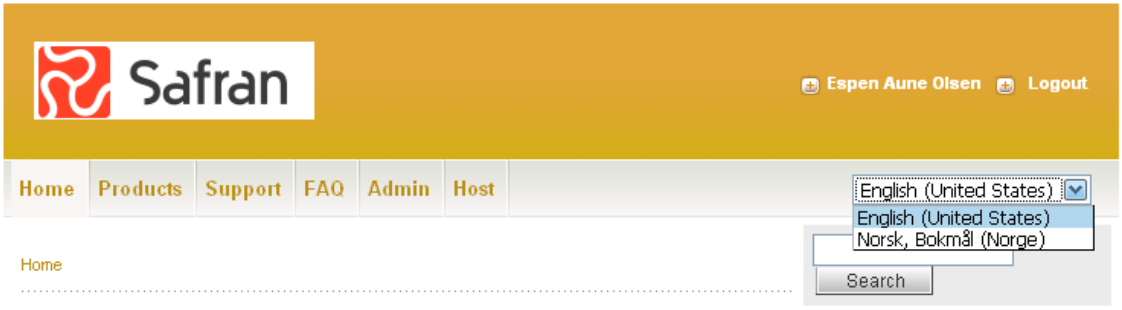

Figure 17: The user can choose between the installed languages

The translation of the web portal framework is done through the use of text files. All that is needed to do a complete translation of all the standard features is to go through a page in the DNN framework and translate word by word.

6 The prototype

In this chapter we present the prototype developed while collaborating with the company. The next chapter is devoted to the design and programming, and to give the reader a better understanding of the various examples and excerpts presented there, we choose to present the prototype in its entirety here. The chapter on design and programming focuses on the process and methods we applied to reach the solution presented in this chapter.

We created four prototype of modules (After this simply referred to as “modules”). In each section we describe a module and its basic functionality, as well as screen shots and an example of use through scenarios. By carefully choosing the scenarios of the modules, we will try to provide a deeper understanding of the user experience related to the modules

6.1 Common settings for modules

6.1.1 Security roles

What a user can do in DNN is decided by security roles. In the following we outline scenarios that describe typical use of the modules. However, we have chosen to focus on the administrator role of the modules rather than on the common user. This will give a better overview of the full range of functionality behind the modules. In the different modules we have given certain user groups permission to be an administrator. For instance, users with the *News writer*-role are allowed to add, edit and delete articles in the news module (Figure 18). Even though the user is granted certain privileges it doesn’t mean that they should be considered to be an administrator of the entire web portal (See Security roles in the chapter on DNN). Remember that a single user can be affiliated with several security roles.

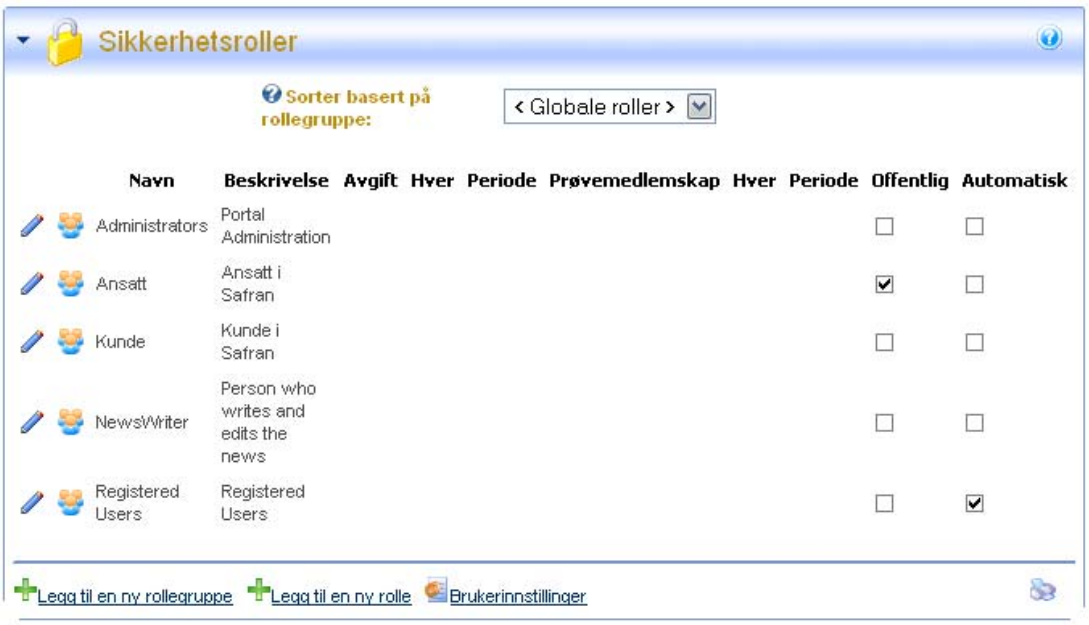

Figure 18: Security roles in DNN

6.1.2 Multiple languages

In meetings with the customer it was decided that the web portal should support two languages: Norwegian and English. This decision was based on the wish to communicate not only with the Norwegian market, but also internationally. Language installation is a simple process (See chapter on DNN) and we decided to support two languages: Norwegian and English. This installation meant that the visiting users could choose between the two languages from a drop down menu.

6.2 The News module

The purpose of the news module was to display a list of news articles for either the employees or the customers. The front-end of the module should display the five latest news articles chronologically by date (Figure 19). The news articles can sometimes be very long and to make it easier for the user to browse through them, we decided to implement an interface consisting of “read more” links after the introduction text (Figure 19). Clicking this link will automatically expand the article so the user can read the entire article on the same page. In that way, we can

present all the news in one page without scrolling or having to browse through multiple pages to read the articles. After the expiry date, the article will not be shown in the list of news.

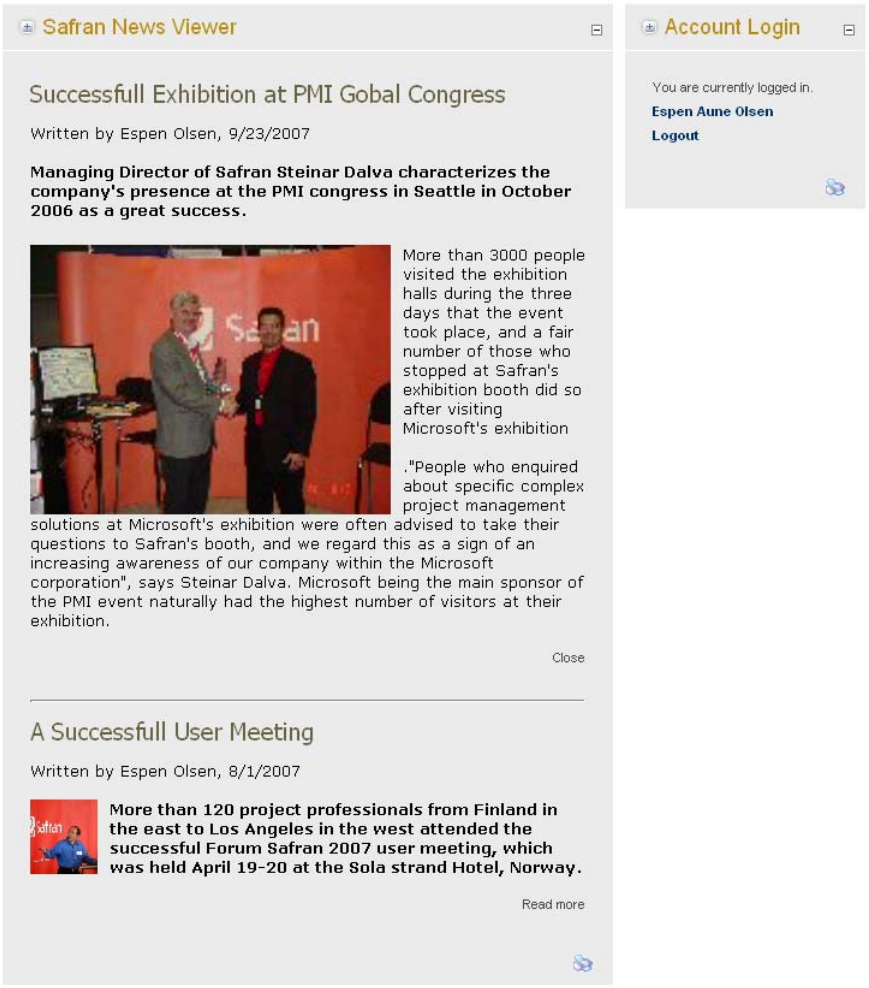

Figure 19: The front-end of the News module

A separate interface allows a user with the *News writer* security role to submit news in both English and Norwegian. It also provides functionality to choose whether the article should be displayed for customers or employees. If an employee wants to read the news published for employees only, the user has to log in before reading it. His or hers security role decides whether the article will be shown or not. However, news articles written for customers are considered as general news and will be displayed to registered and none-registered customers.

The *News writer* has the privileges to add, edit or delete news articles. However, we will only present a scenario of how to add a new article.

6.2.1 Scenario: Add new article

In collaboration with the customer we created a user scenario of how to publish a news article inside the module. The module requires the user to have the appropriate security role, in this case the status as a *News writer* (Figure 18). If the user has this status the module displays a link titled "Add new article". This link leads to a second window where the required fields are shown (Figure 20). First of all, the news writer has to choose the appropriate user group for the article, i.e. customers or employees. After that, he or she has to fill out the article information in Norwegian, and then in English. The remaining fields, publish date, expiry date and article image, are optional. If the user chooses to leave the fields for publish date blank, the system will automatically set the publish date till today's date. The same action on the expiry date field means that the article will remain in the system indefinitely. When the news writer clicks on "Publish article", the system validates each input field. If required fields are missing, the module displays the error(s) where it occurred.

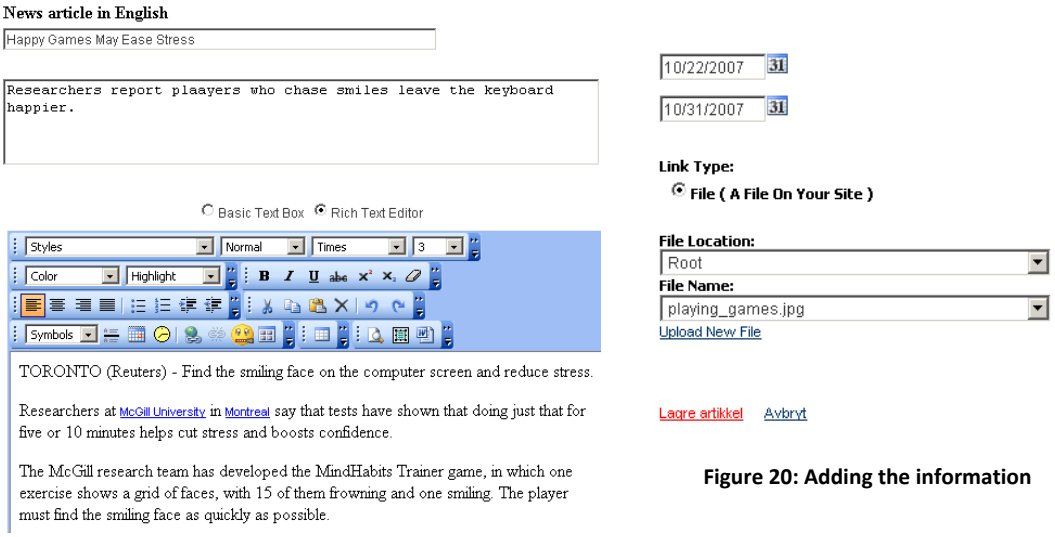

Figure 20: Adding the information

If there are no errors, the article is stored in the database and the *View* mode is displayed (Figure 21).

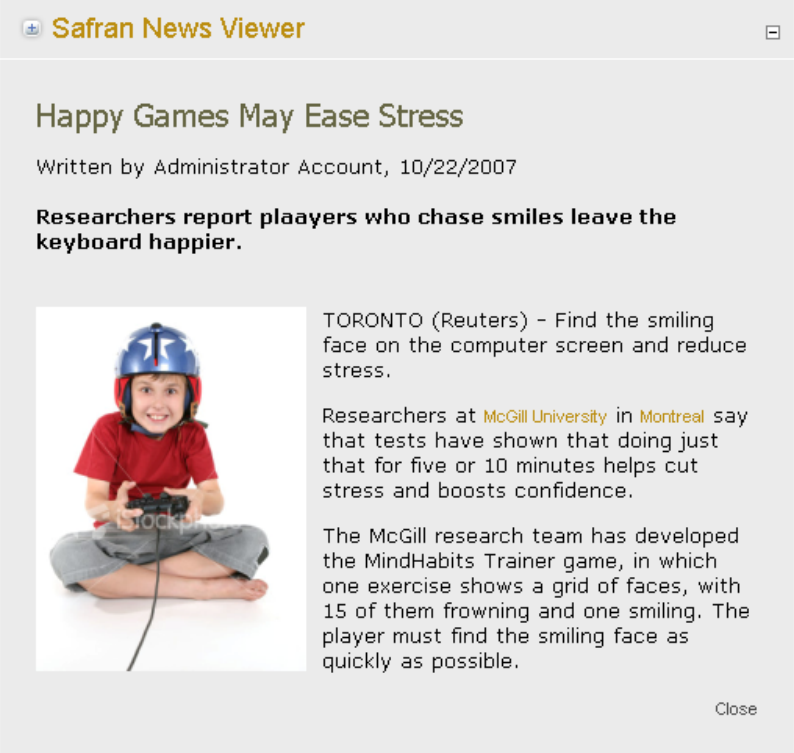

Figure 21: The article is published

6.3 The Product module

The purpose of the product module is to give the customers an overview of the company's products. It consists of two parts that communicate with each other; a *sender* and a *receiver*. The *sender* (The top module in Figure 22), is a tree menu that allows the user to browse the products. The reason we decided to implementing a tree menu is because it looks like the structure in Windows Explorer. In other words, it's recognizable for the users. Another reason is that it's easy to navigate. When clicking on a product, the tree menu will expand and display the sub categories for that product. In our module, these categories are "About" and

"Documentation". When a selection is made, the *listener* (The bottom module in Figure 22) displays the desired information.

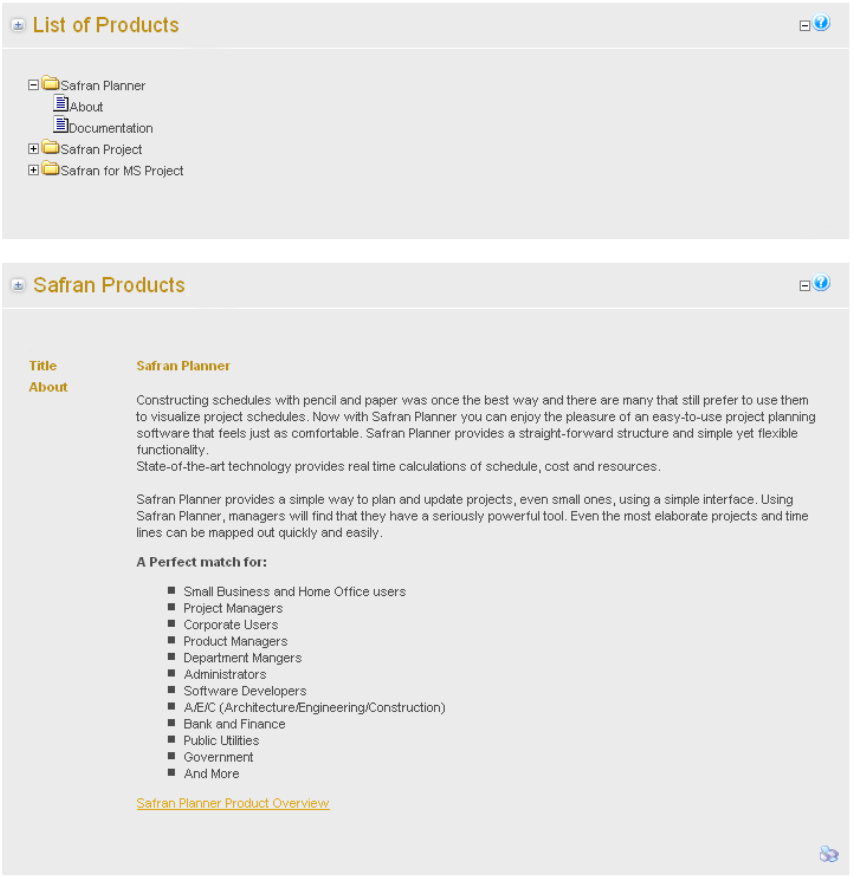

Figure 22: The product module

The task of the *listener* is to constantly listen to calls made from the sender. When a call is made, the *listener* displays the relevant information to the user. Below is a scenario that explains the steps required to edit the information about a product.

6.3.1 Scenario: Edit information about a product

To be able to edit the information about a product the user must have administrator privileges (See chapter on Security roles). In this module we did not create a separate administrator group responsible for moderating the content. Instead, we used DNN's pre defined administrator

account for the task. In order to edit either the “About” or “Documentation” categories, the administrator must choose the desired product in the menu and then click on the pen icon in the display module. The stored information will be displayed in editable text boxes and the administrator can edit the text or add attachments such as *pdf* manuals or images. Clicking on “update” button will store the edited text in database and display the view mode of that category.

6.4 The FAQ module

The FAQ module is a list of frequently asked questions related to the company’s products. By using tabs at the top of the module, the user is able to browse through the information related to three products in a single module. The goal of the FAQ list is to answer common questions about the company’s products that the customer might have. The purpose of the module is to invite the customers to use the portal to find the answers they are looking for rather than contacting the company directly through e-mail or telephone. This is time consuming for both the company and their customers. The tabs allow the user to easily jump between the different products. The list is ordered by question followed by the answer with the newest on the top. The module also features a function that displays the questions only at first and then shows the answers the user wants to see by clicking on it (Figure 23). Using tabs is a good way to separate different categories of related information that should be organized in one place. In the next section we will look at a scenario related to how a user with the right security role can add a new question and answer.

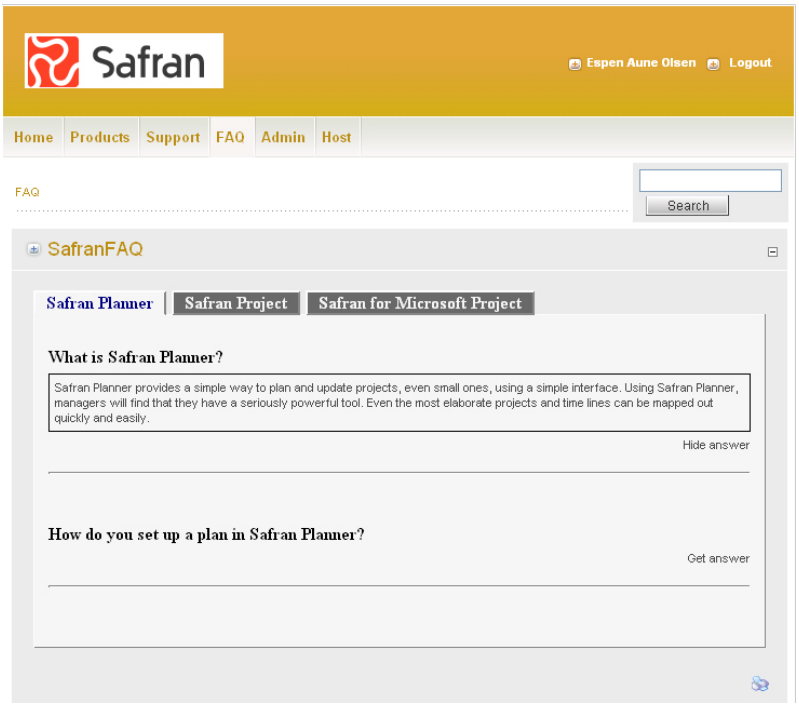

Figure 23: The FAQ module

6.4.1 Scenario: Add new question and answer

Like we saw in the scenario related to the News module, administrating the FAQ module requires the user to be affiliated with a certain security role, in this case the status as a *FAQ writer*. If the user has the correct privileges, the module will display a link at the bottom that says “Add question”. When the link is clicked the module will display the interface for adding a new question. The *FAQ writer* must first choose what product the questions is related to. This is done by selecting the appropriate product from a drop down menu. Furthermore, the *FAQ writer* must fill in a question and then the answer. As with the previous modules, these text boxes can contain text as well as multimedia. When the “Add”-button is clicked, the module will validate the fields and store the information in the database.

6.5 The Support module

The purpose of this module was to allow the customer to request support directly through the web portal. The web portal was supposed to communicate with the company’s internal

Customer Relationship Manager (CRM) system and store the information there. The module was also supposed to get information from the CRM system and display it to the customer. The goal of the module was to function as a communication channel between the company and its customers. In a meeting with the company we outlined a possible strategy for organizing this communication.

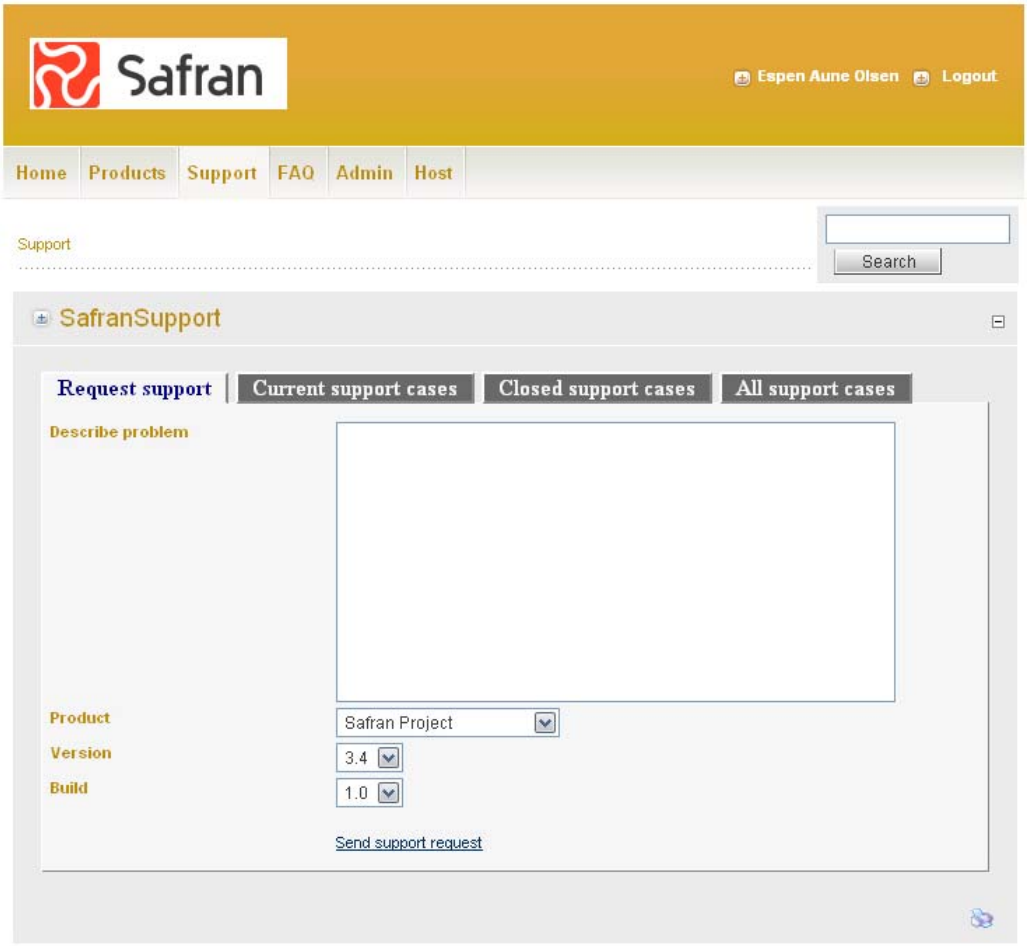

Figure 24: Support module

Notice that this module was created as a "dummy" module (Figure 24). By dummy we mean that the module itself do not serve any purpose except that its interacting UI gives an insight of possible functionalities and their use. Unfortunately we were not able to deliver a working

prototype of this module due to a number of reasons. These reasons are mainly related to the time aspect of our involvement in the project and will be elaborated further in the next chapter.

Although we did not get very far with the planning of this module, we had a vision of what the module was supposed to do. In the following we outline a possible scenario for adding a request for support from a customer's perspective.

6.5.1 Scenario: Add a request for support

In order to add a request for support, the user would have to log in to be identified as a customer. The idea behind the requirement of log in was to identify the customer and the related support agreement. After logging in the user would be able go to the support page (Figure 24). In order to add a support request, the customer would have to fill out the field titled "Describe problem". This is a plain text field where the customer would be able to describe the problem related to a product. After that the customer would have to choose product, version and build from drop down menus. The information is then sent to the CRM system where it would be displayed for the person responsible for responding to inquires within the company.

7 Design and programming process

In this chapter we will describe the development of the modules outlined in the prototype chapter. As we employed both Participatory design and the Agile methodology, we relied on a high degree of user participation. We also describe the customization needed to create the modules. By giving detailed examples of the programming process we also want to highlight some of the challenges related to customizing modules according to specific needs.

7.1 The customer's motivation for building the web portal

Representatives from the company suggested that they needed a new channel for communicating with their customers. They also realized that a lot of time was wasted by spending time on finding solutions to problems that were already solved by someone else. However, as most of the support was provided from person to person either on e-mail, on the phone or in face to face meetings, the knowledge gained was lost between sessions and other employees were unable to learn and reuse previous support sessions. After analyzing their challenges and needs, the company looked at what could be done to improve the situation. At that point, the company's online support was limited to a simple web site and a help desk function that was little used. The web site was static and because it only consisted of simple content written in HTML, the customers were not allowed to interact with the web pages. The help desk was considered to be too impractical and relied too much on outdated technology, to solve the support problems the company was facing. Adding support requests was tedious and challenging and the help desk was hence, little used.

In an internal meeting the company decided that something had to be done. In this meeting it was decided that the solution was to build a new web portal that could handle marketing, customer support and also work as a foundation for an internal network, all in one system. Needless to say, this decision was quite optimistic, as functions like these usually are implemented separately. Furthermore it would require more than two developers with limited time to manage this task.

By analyzing their requirements list, they decided that a web portal would solve most of their challenges. At this point the KIKK project was initialized and our first task in the project was to analyze the various open source frameworks for building web portals that existed online. In addition, the company had already done some research on this and recommended DotNetNuke (DNN) as the most promising alternative.

7.2 Development period

In this chapter we present relevant findings in relation to the research questions of this thesis. The findings address in some detail how we applied the theory described in the first part of the thesis to get the result we presented in the chapter on the prototype.

Our contribution to the KIKK project was to develop a web portal in a real setting, with an actual company. The web portal started as an idea. The starting point for this idea was to improve communication, both externally and internally (See chapter on the KIKK project). Through discussion and exploration with the research group in cooperation with the customer this idea would grow into a working product. This approach, which integrate Participatory Design (PD) with software development, has been explored in another similar project (Mørch, 2005). We approached the project with an open mind knowing that to reach an appropriate solution we were dependent on getting input from the company and collaborate with the involved parts. During the development process we met several challenges. Some of them were technical, others where human, and we needed a way to handle them. Development with the Plan-driven methodology has its advantages, but it also requires planning in advance (See chapter on Plan-driven methodology). However, we realized that the Plan-driven methodology could not be applied in this project for two reasons:

1. The customer's requirements had to be clarified before further planning and development, and
2. the technological aspect contained many uncertainties.

These factors meant that changes were likely to occur after the development phase started. According to the Extreme Programming (XP) method (Beck, 2004), changes are inevitable,

especially in a setting where the goal of the project is to experiment with requirements and unknown technologies.

In the first meeting with the company's representatives the project was still on the drawing board. The representatives we met with realized that they needed a solution to their problem, but they did not know exactly how to specify the requirements, except the wish to get a running web portal. Our assignment was to support them in making the right decisions and simultaneously contribute to the development process. Hence, the most natural way to approach the project was to apply the Agile methodology with a focus on the XP method. In the following sections we will describe how we applied the processes within the Agile methodology, as described in the previous chapters, and then point out how the different Agile and PD methods and techniques supported us during the development phase.

7.3 Work situation

During the development phase of the project we collaborated closely with representatives from the company. We were offered an office twice a week. In order to be able to follow and coordinate the project progress, without project documentation, we needed at least one status meeting every week. During the meetings we gave a description of the work progress and the challenges ahead.

In the work with customizing the web portal we relied on XP as a development method. This meant that we focused on creating working prototypes and maintaining communication channels rather than written documentation. The development was to a large degree dependent on *user stories* (See chapter on XP). However, we based the *user stories* on verbal reports rather than written notes. We relied on this type of work for two main reasons. First, we were situated in the company's office. Hence, we were able to simply walk across the hall when we encountered problems or details that needed to be clarified. Second, at the early stages of the development period we were still exploring technical possibilities of the DNN framework. This meant that it was hard for us to agree upon written *user stories* because we were uncertain of what we could actually achieve in the given time.

Every time we finished an iteration of a module we did a presentation of it and received feedback from a company representative. As XP claims, frequent meetings are important in order to get feedback that encourages further improvement. Sometimes we worked on multiple modules at the same time, and to keep things organized we decided to keep a log that we updated at least once a day (See Appendix A – Daily log for Damir Nedic and Espen Olsen).

During the development of the modules we worked in a Pair Programming (PP) style (Figure 25).

Figure 25: Pair Programming in progress

However, we did not always comply with the “rules” commonly associated with PP. For instance, PP was originally meant for a team of programmers where the pairs rotate every week or month. In our case, we were the only programmers involved and rotation was not an option. Another example of how we deviated from PP is the use of multiple computers. Instead of relying on just one machine, one mouse, one keyboard and one screen, we were sitting on two computers. Although we did most of the programming collectively we discovered that using two computers was more efficient when researching online. This allowed us to find relevant

examples and quickly compare them to our current code. Nevertheless, we believe that our work should be referred to as PP because we have followed the main principles of the method. As we will see later, we experienced many of the benefits of PP such as fewer errors and a more enjoyable working situation.

7.3.1 Cooperative prototyping

In the project we also applied Participatory Design (PD) techniques. Not only did we have representative(s) from the company to review our work. In some of the status meetings we also did code modification while getting feedback from the customer, also known as *cooperative prototyping* (Bødker and Grønbæk, 1992). Often, the presentations were done on a big screen with a projector and when we did modifications, we were able to show the result immediately. In this way the strength of PD can be proved. By including the end users as active participants we ensured that the "right" product was developed. If relying solely on XP while developing the modules, we would not have been able to do modification directly in front of the end users. Instead, we probably would rely more on the *user stories* produced by the company representative. In the end of the thesis, we will discuss some of the similarities and differences between the Agile methodology and PD. Including the end users gave another positive effect: Their enthusiasm for the web portal seemed to rise for every presentation we did, and since their opinions were taken into consideration during further development, each participant felt an increasing ownership to the product.

7.4 Design workshop

After choosing a platform for our project, we needed to gather some initial requirements. In the spirit of Participatory design, we invited some representatives from the company to a design workshop. The goal of this workshop was to clarify how the alternative communication channels between the actors (The employees and their customers). We also wanted to analyze the involved artifacts (computers, available technologies and means of communication) and how these could support communication as needed.

To get the brainstorming going, we presented a scenario of possible problems the company was facing with communication based on the current situation. The scenarios were based on our understanding of the company and its needs. After that, we followed Kensing and Madsen's (Kensing and Madsen, 1992) suggestion of *Future Workshop*. Because some of the other participants of the KIKK project already had been to Stavanger to interview the employees, and therefore, had already discovered most of the problems with the situation today, the *critique phase* was cut short. Instead, some of the problems suggested in the scenarios were corrected and after a brief session we continued. In the *fantasy phase*, we divided the participants into two groups. In this phase we participated along with representatives from the company. One group focused on the problems facing the company's customers, while the other focused on the problems of the company's employees. The main problem to be solved was how support could be done in other ways than through direct contact between a customer and a single employee in the company. The suggestions were put on small post-it notes that could be arranged to simulate the communication flow.

The implementation phase was done by each of the groups presenting their solution (Figure 26). As seen in this picture, we recorded the entire design workshop session with a video camera to make sure that we got a record of everything that happened. These were later translated and used in related reports (Nygård and Mørch, 2007).

Figure 26: Design workshop

At this point the groups' suggestions were put on the white board in separate areas. These areas represented the customer's view and the employees' view. Our goal was to create a complete overview of the desired communication flow that would be supported by the web portal. By removing the line between the separate solutions, we were able to put the two together in a way that opened for discussion among the invited participants. The result was a white board full of post-it notes with directed lines between them (Figure 27). These lines simulated the flow of information necessary between the separate steps of the support.

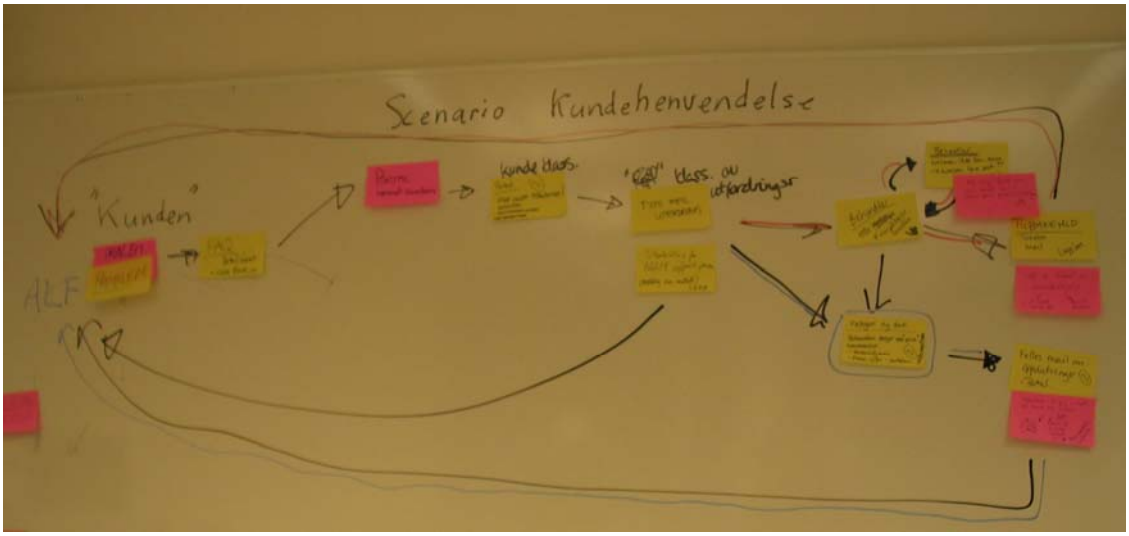

Figure 27: One of the results from the design workshop

Although the results from design workshop outlined a solution beyond the functionality of the web portal, we will describe the basic flow of communication and events here:

1. The customer has a problem or question that needs to be answered.
2. The customer visits the web portal where he or she can either read a list of frequently asked questions or add a request for support.
3. If the customer decides to leave a request for support, information about the problem is entered and sent to the company.
4. The support request is classified in order to delegate it to the right person.
5. The customer gets a notice saying that the request has been received and is currently in a queue.
6. The request is handled by the support personnel.
7. An answer to the request is communicated back to the customer, either through the web portal, by telephone or e-mail. Either way, the support request and the reply is stored in a database, in order to reuse the knowledge next time a similar request is received from a customer.

Even though the results from the design workshop did not reveal any revolutionary new ideas, it enabled us to put the thoughts of the participants into a design everybody could relate to. In the

next sections we will describe how some of these thoughts led to the implementation of several modules.

7.5 Development

While explaining how we applied the XP method, we will also look closer at another aspect of the project; namely the customization of the modules in an open source framework (DNN) within a given context (the KIKK project). While pointing to some examples of how we applied the Agile methodology, we will elaborate on how the programming was done. Our goal is to demonstrate the process of adaptation through *transformation of an idea into a working product*. In collaboration with the customer we went through the steps of brainstorming, requirements gathering, development and feedback in an iterative manner. In the following examples we will not describe how our process relates to these steps, but rather elaborate on our experience with the creation of the modules.

We have chosen to present the development progress in two parts. First, for each module we will examine the process that led to the product. Secondly, we will present adaptation from a programming perspective. Notice that these two parts are not independent of each other. Rather, they are part of an iterative process where decisions made in one area have an impact on the other. By following this structure, we aim to make the distinct parts of the iterative process we went through easier to understand (Figure 28).

Figure 28: The iterative process of creating a module with feedback (F) and prototype (P) development

The figure illustrates how we used an iterative process while developing the News module. Through cycles of creating prototypes (P) and receiving the feedback (F) we gradually got closer to the solution.

7.5.1 The News module

Discussions related to the News module

The initial requirement for the news module was that it should support publishing of news in the web portal. The procedure for writing and publishing news should be simple, so that a non-technical employee would be able to accomplish it. Each article should consist of a title, an introduction, the body, an image and publish and expiry date. The news module should also support news in two languages (English and Norwegian) and news for different user groups

(Employees and customer). Beyond this there were not given any requirements as to what the module should look like or how it should interact with the user. This was probably the result of a discussion around the possibility of changing the user interface (UI) later on in the process. As we saw in the chapter on DNN's architecture, the use of layers separates functionality from the UI. Hence, it was a common conception that user interaction could evolve out of the process of user feedback.

During the development of the module we ran into several technical challenges. These are outlined in the next section. However, we also experienced episodes where close collaboration with the customer helped us in customizing the module closer to the specific needs.

When we started the work with developing the news module, we had little experience with the design and construction of modules. Initially we thought that the fastest way to get started was to build the module from scratch. However, we soon discovered that building up a module consisting of several layers and advanced functionality was a challenging task. In discussions with a representative from the company we were advised to use a plug-in for Visual Studio that would help us with standard functionality. We started by creating a module that published news without being concerned about user permissions or two languages. The goal was to create a prototype that could visualize the solution we had in mind after the initial meetings with the customer, as suggested in the chapter on software prototyping.

We thought that the easiest way to solve the problem would be to split the task into two separate modules. Therefore, we decided to start working on the module that added data to the database. This module was given the name *News module*. Although this was quite a challenge at first we managed to overcome the technical difficulties and the result was a module meant for administrators only. After creating this module, we created a second module that would display the news added to the database. This module was given the name *News viewer*. At first, the requirements were quite modest. What we wanted to achieve was to get the data stored through the *News module* and display it on the screen. The purpose was to learn how objects and variable were sent through the layers of modules. After succeeding in creating two separate modules, we looked closer at the plug-in. We realized that the two tasks of administrating and

displaying news could be done in one module. As mentioned in the chapter on DNN, each module has a View mode and an Edit mode (as well as a mode for changing settings). Hence, our next goal was to merge the modules into one piece. This would make it easier for a non-technical person to administer everything related to the news articles because the *Edit* mode would be available directly from the *Display* section of the module (Mørch, 2005).

This proved to be a feasible task as the plug-in consisted of an example that already had built in the separate modes in one module. After merging the two modules into a common *News module*, we started to work on the requirements from the customer. Specifically we worked on two tasks.

1. Support for Norwegian and English versions of the same article.
2. Support for adding articles for either the company's customers or employees.

In order to add support for adding the news articles in two languages we created fields in the user interface (UI) that allowed the administrator to write the same article twice. We organized the UI in a way that allowed the administrator to first write the title of the article in Norwegian, then in English. We did the same for the other fields (Introduction and body).

During our work with the News module we experienced how the Agile methodology and particularly Extreme programming's mantra: *Embrace change*, was relevant. When presenting the initial News module for the end users (Figure 29), we witnessed an unexpected desire to change previously stated requirements of how the module should interact with the user.

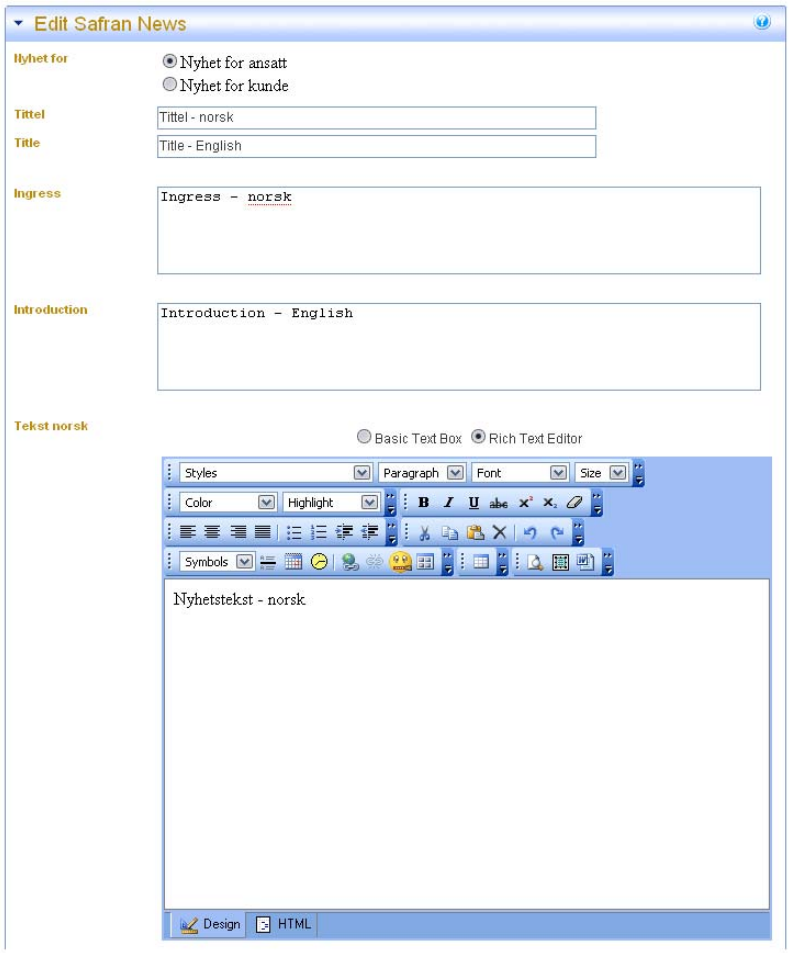

Figure 29: Initial news module

The customer argued that it would be easier to write news in a different order. All the fields for information in Norwegian should be entered first. Then the equivalent information should be entered in English.

During the iterative process of development and feedback (Beck, 2002), we uncovered an important finding. Even though the customer initially had a vision of what the module should look like, creating a prototype that visualized the possible outcome made the users change their minds. After testing the module, the end users realized that it would be quite demanding to keep track of both Norwegian and English at the same time. One user suggested it would be easier to write the entire article in one language, and then translate it afterwards. As Visual

Studio supports changes in the UI through drag and drop, rearranging the order of the input fields was a simple task. The result can be seen below (Figure 30).

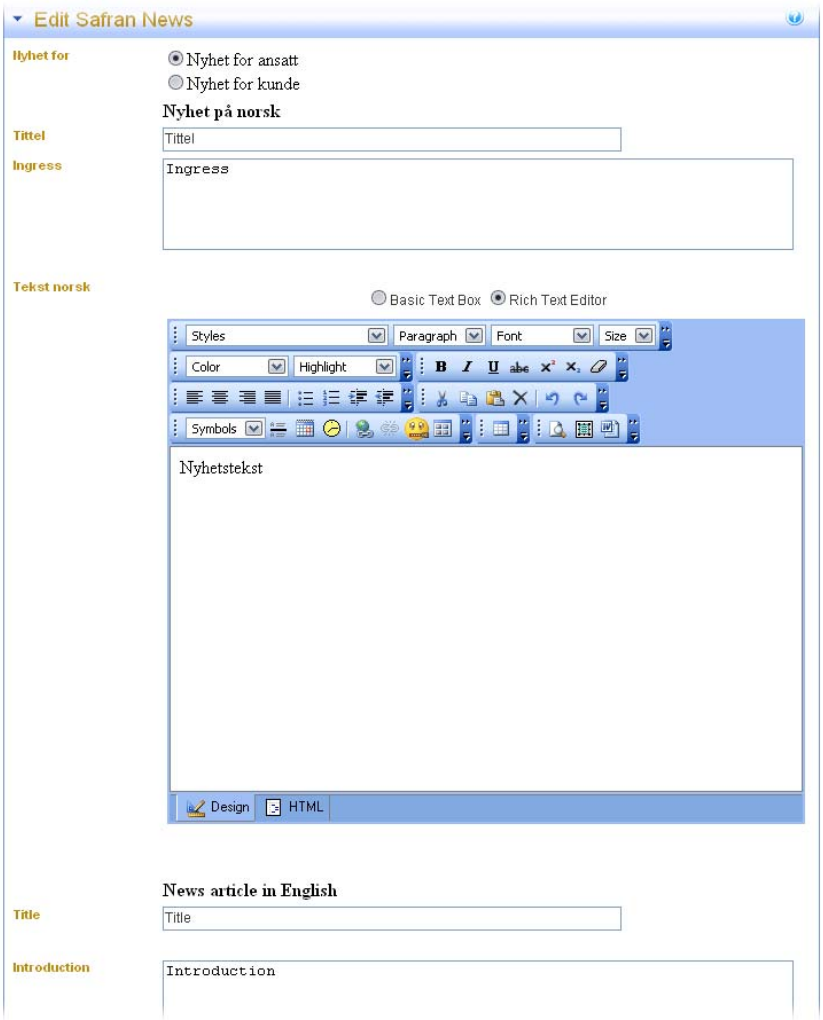

Figure 30: News module after changes

In collaboration with the company, it was decided that the news articles should be divided into categories. It was discussed whether the News module should support three categories: News for employees, news for logged in customers and news for visitors to the web portal, i.e. people visiting the site without logging in. However, it was decided that the module initially should only contain news for customers and news for employees, i.e. both logged in customers and by passing visitors. As mentioned in the chapter on DNN, the DNN framework supports the concept

of security roles. The administrator interface of DNN allowed us to create a security role for employees and a role for customers. The possibility to choose whether the news article should be displayed for customers or employees was implemented by forcing the administrator to choose one of the two options when publishing a news article.

After the prototype was presented and tested by the future users, another requirement was discovered. When a user is given administrator rights in DNN, he or she can edit the content of a single module. However, the same person is also able to edit the contents of the entire web portal. This includes not only editing the overall look of the portal, but also adding and removing modules, as well as administration of the user accounts. The solution to this problem is to keep the number of administrators as low as possible. However, as we saw in the chapter on DNN, different security roles can get different levels of administration rights. In this case we created a new security role labeled *News writer*. A user with the permissions of a *News writer* would be able to administer the news without being able to change anything but the News module.

During the development of the News module discovered that close collaboration with the customer can clarify and even uncover previously unknown requirements. User participation and iterative work with repetitive requirements gathering, development, presentation and feedback does lead to the creation of software that incrementally gets closer to what the customer actually needs. In the next section we will look at the technical aspect behind the programming.

Program construction

The News module was the first module we tried to implement. In order to keep things organized we divided the development into two parts. This was the obvious solution as we were to implement two separate processes: Insert news into the database and retrieve news from the database and display it in the UI.

The first step to creating the part of the module that would submit news to the database was to implement a table in the database that would store the article information. This table was

changed several times during the development, and we present the final version here (Figure 31).

NewsId		int		Auto Increment		Not Null	
ModuleId		int				Not Null	
UserId		int				Not Null	
Title		Varchar(200)				Not Null	
Ingress		text				Not Null	
NewsText		text				Not Null	
TitleEng		Varchar(200)				Not Null	
IngressEng		text				Not Null	
NewsTextEng		text				Not Null	
DateAdded		datetime				Not Null	
DateExpired		datetime				Null	
Picture		Varchar(50)				Null	
ForEmployee		tinyint				0 or 1	

Figure 31: Table News from the database

NewsId is the id of the news article. This Id is used every time an article is retrieved from the database. *ModuleId* denotes the Id of the module. This is always the same as long as the news is submitted through the News module. The next fields are obvious, but the field *ForEmployee* requires an explanation. In order to separate the news for employees and customers we have made a field that stores either 0 or 1 (*tinyint*). When this variable is set to 0, it means false or not for employees, i.e. for customers. In the case of 1 it means that the article is indeed for employees.

In order to create a table in the database through DNN, the developer has to write a *stored procedure*, which is a command to the SQL database. This procedure is called and run upon the initial installation of the module (Figure 32).

```
-- Creating table for news -------------------------
if not exists (select * from dbo.sysobjects where id = object_id(N'com{objectQualifier}News') and
OBJECTPROPERTY(id, N'IsTable') = 1)
CREATE TABLE {databaseOwner}{objectQualifier}News(
      [NewsId] [int] IDENTITY(1,1) NOT NULL,
      [ModuleId] [int] NOT NULL,
      [UserId] [int] NOT NULL,
      [Title] [varchar](200) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
      [Ingress] [text] COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
      [NewsText] [text] COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
      [Title_eng] [varchar](200) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
      [Ingress_eng] [text] COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
      [NewsText_eng] [text] COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
      [PublishDate] [datetime] NOT NULL,
      [ExpiryDate] [datetime] NULL,
      [ForEmployee] [tinyint] NOT NULL,
      [ImageId] [int] NULL
) ON [PRIMARY]
```

Figure 32: Stored procedure for creating the News module

The first line checks if the table already exists in the database. The rest creates the table News and sets the proper fields.

After creating the table in the database we started implementing the user interface (UI). We created fields where the *News writer* could write the article information. These fields were closely related to the fields in the News table. In order to store the news article in two languages we created text boxes where the *News writer* could write the same article twice but in a different language.

Below is an example of the code created to display the fields for writing the news article in English (Figure 33). This code is part of a form where the written input is sent to the Business Logic Layer (BLL). Here the information is transmitted further.

```
<asp:TableRow VerticalAlign='Top'>
        <asp:TableCell ID="TableCell19" runat="server" width="200" CssClass="subhead">
                Title
        </asp:TableCell>
        <asp:TableCell ID="TableCell20" runat="server">
                <asp:TextBox runat="server" ID="Title_eng" cssclass="NormalTextBox"
                width="400px" Columns="30" maxlength="200"></asp:TextBox>
                <asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server"
                ErrorMessage="Vennligst fyll ut en engelsk tittel" ControlToValidate="Title_eng"
                />
        </asp:TableCell>
</asp:TableRow>

<asp:TableRow VerticalAlign='Top'>
        <asp:TableCell ID="TableCell21" runat="server" width="200" CssClass="subhead">
                Introduction
        </asp:TableCell>
        <asp:TableCell ID="TableCell22" runat="server">
                <asp:TextBox runat="server" ID="Ingress_eng" TextMode="MultiLine"
                Height="80px" Width="550px"></asp:TextBox>
                <asp:RequiredFieldValidator ID="RequiredFieldValidator4" runat="server"
                ErrorMessage="<br />Vennligst fyll ut en engelsk ingress"
                ControlToValidate="Ingress_eng" />
        </asp:TableCell>
</asp:TableRow>

 <asp:TableRow VerticalAlign="top">
        <asp:TableCell ID="TableCell7" runat="server" width="200" CssClass="subhead">
                News text
        </asp:TableCell>
        <asp:TableCell ID="TableCell8" runat="server">
                <dnn:TextEditor runat="server" ID="NewsText_eng" TextMode="MultiLine"
                Height="300px" Width="550px" />
                <asp:RequiredFieldValidator ID="RequiredFieldValidator6" runat="server"
                ErrorMessage="<br />Vennligst fyll ut en engelsk nyhetstekst"
                ControlToValidate="NewsText_eng" />
         </asp:TableCell>
</asp:TableRow>
```

Figure 33: Code example of UI

In the example (Figure 33) each table row (asp:TableRow) denotes a field required to publish the news article. The first row is the title. This is a simple textbox with a max length of 200 characters. The second row is the introduction. This is a field that expands over several lines. The third row is the main text of the article. Here we have applied the textbox included in the

DNN framework. The difference between this textbox and a regular textbox is that it supports multiple fonts and insertion of multimedia files along with the text (Figure 34).

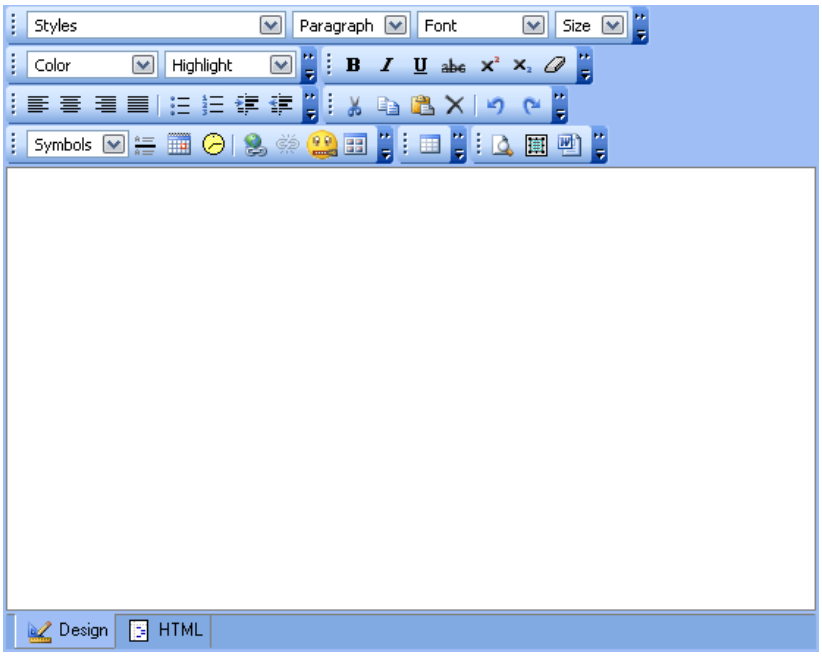

Figure 34: Example of the textbox provided by the DNN framework

When the *News writer* has entered all the fields, the system validates that the required fields has been filled out. Below is an example of the code required to validate a field (Figure 35)

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator6" runat="server" ErrorMessage="<br
/>Vennligst fyll ut en engelsk nyhetstekst" ControlToValidate="NewsText_eng" />
```

Figure 35: Required field validator

This code automatically checks if the field has some entered text when the submit button is pushed. After the required fields are validated the BLL receives the entered data in order to pass it on to the Data Access Layer (DAL), where the data can be inserted into the database. The BLL creates a News-object that can store the entered information. Below is the declaration of the News class (Figure 36).

```
public class News
{
    private int _NewsId;
    private int _ModuleId;
    private int _UserId;

    private string _Title;
    private string _Ingress;
    private string _NewsText;

    private string _Title_eng;
    private string _Ingress_eng;
    private string _NewsText_eng;

    private DateTime _PublishDate;
    private DateTime _ExpiryDate;
    private int _ForEmployee;
    private int _ImageId;

    private int _PortalId;
    private string _FileName;
}
```

Figure 36: Declaration of the News class

The information from the entered fields are then stored in the object. Below is the code necessary to achieve this for the basic text fields (Figure 37):

```
oNews. NewsId = getNewsId();
oNews.ModuleId = this.ModuleId;
oNews.UserId = getUserId();

oNews.Title = Title.Text;
oNews.Ingress = Ingress.Text;
oNews.NewsText = NewsText.Text;

oNews.Title_eng = Title_eng.Text;
oNews.Ingress_eng = Ingress_eng.Text;
oNews.NewsText_eng = NewsText_eng.Text;
```

Figure 37: Insertion of data into the object

The method, *getNewsId()*, finds the Id of the news article if it's an update rather than the insertion of a new article. ModuleId is set to the Id of the current module. This is a variable available from the DNN framework. The *getUserId()* method gets the current user's Id. The rest

is pure text and is inserted directly from the entered text in the UI. The other fields require some adoption and we will look at some examples of this later.

When the variables of the News object have been set, it's sent to the DAL. In the DAL an appropriate method gets the variables of the object and sends them over to a *stored procedure*. In this example the object is received by the method *InsertNews(News oNews)* (Figure 38).

```
cmd.Parameters.AddWithValue("@ModuleId", oNews.ModuleId);
cmd.Parameters.AddWithValue("@UserId", oNews.UserId);

cmd.Parameters.AddWithValue("@Title", oNews.Title);
cmd.Parameters.AddWithValue("@Ingress", oNews.Ingress);
cmd.Parameters.AddWithValue("@NewsText", oNews.NewsText);

cmd.Parameters.AddWithValue("@Title_eng", oNews.Title_eng);
cmd.Parameters.AddWithValue("@Ingress_eng", oNews.Ingress_eng);
cmd.Parameters.AddWithValue("@NewsText_eng", oNews.NewsText_eng);
```

Figure 38: The InsertNews(News oNews) method

The method *AddWithValue()* sends a variable from the News object along with a string, preceded with @, that lets the *stored procedure* know the type of variable that is sent. Below is the *stored procedure* required to insert a news article into the news table in the database (Figure 39).

```
-- Insert article into table -----------------------------------------

if exists (select * from dbo.sysobjects where id =
object_id(N'{databaseOwner}{objectQualifier}InsertNews') and OBJECTPROPERTY(id,
N'IsProcedure') = 1)
 DROP PROCEDURE {databaseOwner}{objectQualifier}InsertNews
GO

CREATE PROCEDURE {databaseOwner}{objectQualifier}InsertNews
@ModuleId int,
@UserId int,

@Title varchar(200),
@Ingress text,
@NewsText text,

@Title_eng varchar(200),
@Ingress_eng text,
@NewsText_eng text,

@PublishDate datetime,
@ExpiryDate datetime,
@ImageId int,
@ForEmployee tinyint
AS
INSERT INTO News (ModuleId, UserId, Title, Ingress, NewsText, Title_eng, Ingress_eng,
NewsText_eng, PublishDate, ExpiryDate, ForEmployee, ImageId) VALUES (@ModuleId, @UserId,
@Title, @Ingress, @NewsText, @Title_eng, @Ingress_eng, @NewsText_eng, @PublishDate,
@ExpiryDate, @ForEmployee, @ImageId)
```

Figure 39: Stored procedure for inserting news articles into the News table

After creating the basic functionality for inserting a news article into the database, we started to look at the functionality required to display the news in the module for users to see. This process is very much related to the first process. However, this time we retrieved data from database rather than inserting data.

In order to retrieve an article from the database a method in the DAL is called from the BLL. This method takes an empty News object created in BLL and fills it with variable from the database. This is done with the help of a *stored procedure*. Below is an example of one of the most common stored procedure for retrieving data from the database, namely the one that gets the 5 latest articles in the database (Figure 40).

```
-- Get all news articles(gets news either for employee or customer) ---
CREATE PROCEDURE {databaseOwner}{objectQualifier}GetNews
@ForEmployee tinyint
AS
SELECT TOP 5 NewsId, PortalId, FileName, UserId, Title, Ingress, PublishDate, ExpiryDate, NewsText
FROM Files, News WHERE (FileId=ImageId) AND (ExpiryDate IS NULL OR GETDATE()<ExpiryDate)
AND GETDATE()>PublishDate AND (ForEmployee=@ForEmployee) ORDER BY PublishDate DESC
```

Figure 40: Stored procedure that gets 5 latest news articles

Notice that the *Select* query in this example only gets the articles in Norwegian. Later we will see how a variable in the DNN framework allowed us to get the current user's preferred language.

When the data is retrieved from the database it is inserted into the object in the DAL before it's sent back to the BLL. This process is quite similar to the process of inserting data into the database, except that it's done in the opposite order. In the BLL, a *DataBinder* stores all the retrieved objects from the DAL. A *DataBinder* is an object that can store the information from a data source (in this case, data from a database). In the UI, a *Repeater* goes through the *DataBinder* object by object (Figure 41).

```
<asp:Repeater ID="NewsList" runat="server">
        <ItemTemplate>
                <asp:label id="title" runat="server" Text='<%#      DataBinder.Eval(Container,
                "DataItem.Title") %>'/>
                <asp:label id="ingress" runat="server" Text='<%# DataBinder.Eval(Container,
                "DataItem.Ingress") %>'/>
                <asp:label id="NewsText" runat="server" Text='<%#
                Server.HtmlDecode(DataBinder.Eval(Container, "DataItem.NewsText").ToString())
                %>' />
        </ItemTemplate>
</asp:Repeater>
```

Figure 41: Excerpt of the code needed to display the vital information of a news article (Title, introduction and body)

The example above illustrates the power of the *Repeater*. The process illustrated above is repeated as long as there are objects connected to the *DataBinder*. Notice that the example only illustrates how the data from database is displayed in the module. It is also possible to add syntax that describes of the element should be placed within the module.

The process covered in this section this far describes how data is put into the database and then how the data is retrieved. This process is very common and we have applied it in all the modules. Hence, we choose to only present it once. However, the News module also contained other variables and concerns than just displaying pure text. In the following we will elaborate on how the requirements of multiple languages and user groups were implemented.

In order to decide whether the article should be displayed for customers or employees, we let the *News writer* choose between two options when writing the article. These options were implemented in the UI (Figure 42).

```
<asp:TableRow VerticalAlign='Top'>
        <asp:TableCell ID="TableCell15" runat="server" width="200" CssClass="subhead">
                Nyhet for
        </asp:TableCell>
        <asp:TableCell ID="TableCell16" runat="server">
                <asp:RadioButton ID="employee" GroupName="NewsFor" runat="server"
                Text="Nyhet for ansatt"/><br />
                <asp:RadioButton ID="customer" GroupName="NewsFor" runat="server"
                Text="Nyhet for kunde"/>
        </asp:TableCell>
</asp:TableRow>
```

Figure 42: Radio buttons for employees or customers

Two *Radio buttons* gives the *News writer* the option of whether the article is intended for customers or the employees. In BLL a variable in the object of the News class will describe who the article is for (Figure 43).

```
if (employee.Checked)
        oNews.ForEmployee = 1;
else
        oNews.ForEmployee = 0;
```

Figure 43: Checking who the article is for in the BLL

This is related to the News table in the database. If the *ForEmployee* variable is checked, the *tinyint* is set to 1, otherwise it is set to 0. This variable is then sent to the DAL and inserted to the database as described earlier in this section.

When displaying the news in the database we had to check what security role the user belonged to in order to display the correct news. The solution was to get the relevant security role from the DNN framework. As we can see in the code below, we can get a user's security roles by calling DNN's *GetRolesByUser()* method (Figure 44).

```
foreach (string i in mUserRoles.GetRolesByUser(mUser.UserID, mRoles.PortalID)) {
        if (i.ToString().Equals("Administrators") || i.ToString().Equals("NewsWriter")) {
        userRole = 2;
            break;
        }
        else if (i.ToString().Equals("Ansatt")) {
                userRole = 1;
                break;
        }
        else if (i.ToString().Equals("Kunde")) {
                userRole = 0;
                break;
        }
}
```

Figure 44: Get the appropriate security role

If the user who is logged in is an "Ansatt" (Employee), the *userRole* variable is set to 1. If the user is a "Kunde" (Customer) or the user is not logged in, the *userRole* variable is set to 0. In the case of the user being an "*Administrator*" or a "*NewsWriter*" the *userRole* is set to 2 which give him or her permission to view all news articles. Remember that the *stored procedure* for getting news articles receives the *ForEmployee* variable from the DAL (See figure Figure x: Stored procedure that gets 5 latest news articles). In the *Select* query it is used to get the latest 5 articles for either employees or customers. In the case of the user being logged in as an administrator or *News writer*, a different *stored procedure* is called. This is similar to the previous mentioned *stored procedure* except that it gets all the news articles from the database, as administrators should be able to edit all the articles regardless of who it was originally intended for.

Another example of customization was the need to display the articles in both Norwegian and English. As described in the Prototype chapter and in previous examples this is done by making the *News writer* the news in two languages when publishing an article. However, when

displaying the *View* mode of the News module, the language is decided by the user's preferred option of language. This meant that we had to get the selected language at run time before calling the *stored procedure* that would get the articles in the appropriate language. In order to get the user's selected language, we had to use a variable available from the DNN framework. The example below illustrates how we did this (Figure 45).

```
String language = System.Threading.Thread.CurrentThread.CurrentCulture.Name;
```

Figure 45: Command to get the currently selected language

This variable contains the chosen language. In the case of English it would contain "en-US", in the case of Norwegian, "No-nb".

7.5.2 The Product module

Discussions related to the Product module

Another finding related to the agile methodology, was the use of XP to experiment when developing the Product module. After an initial brainstorming it was decided that the company's products were to be presented in a tree structure. The purpose of arranging the product in a tree structure was to present the products at a glance at first, and then allow the visitor to choose a product in order to get more in-depth information. Initially, the tree structure would list company's products. When clicking on a product, the tree expands, showing the categories related to the product. These sub categories were information about the product and the product versions and relevant documentation.

Due to technical limitations we decided to place the tree menu and the display in two separate modules. However, the two modules would still be placed in a package and hence, could be installed as one module. We wanted the tree structure to communicate with the content part that displayed the chosen information. This meant that the menu module had to be considered to be a sender, while the display module was the receiver. In the previous modules we only had to be concerned with one module at the time, but now we had to deal with multiple parts,

which required advanced programming. However, using XP's pair programming method and prototyping helped us to manage these tasks.

DNN is an open source framework that has a large community that cooperates in creating different kinds of plug-ins. In order to implement a tree structure in a module, we had to download a plug-in from an online community[6] and customize it to our needs. Although this seems to be an achievable task, it was not that simple. The plug-in was poorly documented and we had problems while doing the advanced programming required customizing the module. First of all, the code was written in Visual Basic rather than C#. This was a problem as we were unfamiliar with the syntax and unsure of how to convert the code. Secondly, there was no documentation except one simple example. This example provided a simple tree structure but we also had to be concerned with how the two parts of the Product module should communicate with each other. This meant that we had to modify the example to fit our more advanced needs. However, modifying poorly documented code can be frustrating unless the code is well organized and easy to read. One advantage of pair programming is to reduce that frustration when meeting obstacles (See chapter on PP). Furthermore, experimenting with unfamiliar code will automatically be error-prone. Pair programming ensures that the number of errors is reduced (See chapter on PP). And that is what we experienced when working on this module. When finished, the module worked almost perfectly, with only minor flaws. When working in a PP style while trying to solve problems, we generally felt that the process was eased. Building prototypes and doing customization often requires trying and failing, especially when experimenting with an unfamiliar programming environment. Another advantage of PP is that it leads to more effective learning and building of common knowledge among the programmers. For instance, whenever one of us was absent for a day or a few hours, the other was able to continue the work nevertheless. In our experience, PP also reduced the stress because we were able to discuss all the problems we met directly in front of the computer, without having to schedule meetings or even leave the office.

---

[6] http://webcontrols.dotnetnuke.com

While customizing the tree structure to our needs, we noticed that the nodes of the tree could be implemented in two ways. They could either be hard coded into the code file or be dynamically retrieved from the database. We choose to implement the static version as would save us a lot of time. The company's representative also argued that their range of products were unlikely to expand in the near future. According to the principles of Extreme programming (Beck, 2004), *simplicity* allows the developers to only be concerned about today's challenges.

Program construction

The Product module uses a tree structure that allows the user to navigate through the information related to each product. Like mentioned in the section above, the tree structure can be applied by downloading a plug-in from DNN's web site.

In DNN, these libraries are referred to as web controls. By downloading and connecting a DLL-file to Visual Studio we got an extra toolbox that allowed us to add the web controls to our modules. DLL-files (Dynamic-link library) are shared files that can be used by programs under runtime. Basically this means that instead of having several processes calling multiple procedures that do the exact same thing; one procedure is written and stored in a DLL-file. In this way the procedure is loaded only once in the memory of the computer and it can be used by different processes.

When the DLL-file is downloaded and the path to it is correct, the tree structure can be implemented in the module. Visual Studio's IntelliSense (see chapter on Visual Studio) can also be used to simply access the code needed to implement the tree structure in the User Interface (UI) (Figure 46).

```
<DNN:dnntree id="MyDNNTree" runat="server" DefaultNodeCssClassOver="Normal"
DefaultNodeCssClass="Normal" cssClass="Normal"
OnPopulateOnDemand="MyDNNTree_PopulateOnDemand"
OnNodeClick="MyDNNTree_NodeClick" CollapsedNodeImage="../../../Images/Plus.gif"
ExpandedNodeImage="../../../Images/Minus.gif" />
```

Figure 46: The code needed to implement the tree structure

The DNNTree has an Id that has to be unique (“MyDNNTree”). When populating the nodes related to the tree, the Id is used to link the nodes to the correct tree. The variable *OnPopulateOnDemand* on line 4 allows the nodes in the tree to get populated at run time when the node is expanded. This means that the content of the nodes can be populated dynamically from for instance a database, rather than being predefined in the programming code. In the BLL, the *PopulateTree()* method (Figure 47) is invoked. This method fills in nodes for that tree. Notice that we decided to initialize the nodes of the tree statically in the code file, rather than getting data from the database.

```
private void PopulateTree()
{
    DotNetNuke.UI.WebControls.TreeNode objNode = new
    DotNetNuke.UI.WebControls.TreeNode("Planner");
    objNode.ToolTip = "Information about Planner";
    objNode.ImageIndex = (int)eImageType.Folder;
    objNode.ClickAction = DotNetNuke.UI.WebControls.eClickAction.Expand;
    objNode.HasNodes = true;
    MyDNNTree.TreeNodes.Add(objNode);
    PopulateChildrenTreeNodes(objNode, 1);
}
```

Figure 47: The PopulateTree() method

Line 1 creates an object of the node with the name “Planner”. The variable *ToolTip* on line two sets the alternative alt tag. This is the text which will be showed when user points a mouse over the node. Line three sets the icon of the node. This should be set to “folder” if the node is a parent node or a “page” if the node is a children node. The *ClickAction* variable describes the action that should be applied if the node is clicked. The options are to expand in order to display the children nodes or go to the desired target, i.e. display the appropriate information in the display module. *HasNodes* sets whether the node has children nodes or not. The *Add()* method adds the current node to a tree. In this case it’s added to the previously implemented tree, *MyDNNTree*. The last line sends a call to a second method that populates the current node with children nodes.

PopulateChildrenTreeNodes(objNode, 1) sends the current node and an index that will be used in the next method (Figure 48).

```
private void PopulateChildrenTreeNodes(DotNetNuke.UI.WebControls.TreeNode objParent, int
key)
{
      int index = 0;
      DotNetNuke.UI.WebControls.TreeNode objTreeNode;
      index = objParent.TreeNodes.Add();
      objTreeNode = objParent.TreeNodes[index];
      objTreeNode.Text = "About";
      objTreeNode.ToolTip = "About this product";
      objTreeNode.ImageIndex = (int)eImageType.Page;
      objTreeNode.ClickAction = eClickAction.PostBack;
      objTreeNode.Key = key + "+about";
}
```

Figure 48: The PopulateChildrenTreeNode() method

PopulateChildrenTreeNode() takes the parent object node and an integer value key as parameters. In line 4, another node object is initialized. This node will eventually become the children of the node that was sent into the method. Line 5 gets the index of the next children nodes of parent object node and line number 6 associates the children object node (line 4) to the parent node. In this way, the parent node always knows how many children it has. The *Text*, *ToolTip*, *ImageIndex* and *ClickAction* are initialized as mentioned in the previous example. The variable *objTreeNode.Key* is sent to the display module (The receiver). For instance, sending the code “3about” to the receiver, means that the receiver should display the stored about-text for product three.

We have mentioned that one module is used to list the products with sub categories in a tree structure, while the other which displays the chosen information. In order to do that the Product list module is initialized as the “sender” and Products as the “target”. When the user clicks on a node in the tree structure, a key value for that node is send to the target module. By looking at that key, for instance “3about”, the module can easily determine for which of the products it should display information about and get the appropriate information from the database.

The DAL gets the appropriate product from the database by the help a *stored procedure*. When the object is returned to the BLL, an if-statement determines whether the module should the display information in the *About*-string or the *Documentation*-string (Figure 49).

```
if (tmp[1].Equals("about"))
            {
               AboutRow.Visible = true;
               DocumentationRow.Visible = false;
                     AboutText.Text = Server.HtmlDecode(objProducts.About);

else if (tmp[1].Equals("docs"))
            {
               DocumentationRow.Visible = true;
               AboutRow.Visible = false;
               Documentation.Text = Server.HtmlDecode(objProducts.Documentation);
            }
```

Figure 49: Display the correct information in the module

The if-sentences determine the sub category to be displayed. Notice that this code excerpt can be extended to support more nodes in the tree structure. Creating support for dynamic population of the tree should also be a manageable task.

7.5.3 The FAQ module

Discussions related to the FAQ module

The development of the FAQ module was based on a wish to have a module that could give users of the companies' product a list of frequently asked questions. While discussing the Support module (Described in the next section) with the end users, we agreed that developing that module would be time consuming. At that time, the company was still in the process of choosing an appropriate CRM solution, which the support module depended on (More on this in the chapter on the support module). Again, we relied on Extreme Programming's advice to strive for *simplicity*. In collaboration with the customers, it was decided to start the development of another support function, namely a list of frequently asked questions. This module was independent of third party software, which meant we could get started right away.

Since the FAQ module the question answers for all of the company's products, the natural requirements were:

1. Place all the information in one module
2. Easy to navigate and find the desired information.
3. The questions answers should be stored in the database.

A representative from the customer expressed that the desired solution should be in the form of tabs within the module. Tabs make it easy to navigate and give a good overview of the available options as long as there are a limited number of tabs (Figure 50).

Figure 50: Tabs in a DNN module

After creating the prototype that supported the initial requirements, we had a second meeting with the customer where the module was presented. The demonstration version of the module only contained one question and one answer. After the presentation, the customer expressed a concern implying that when the list of questions and answers eventually would get longer, it would also be harder to find the desired information the user is looking for. As a result, it was decided that initially only the questions should be displayed in the list. Next to each question, there should be a link titled "Get answer". When clicked, the answer is displayed below the question. The link text is also changed to "Hide answer" which obviously hides the answer when clicked.

Program construction

The FAQ module uses a navigation menu at the top. This menu is formed as tabs that allow a user to browse through several windows without having to reload the page. In order to use tabs in DNN, we had to download a library from DNN's web site (Like described in the section on the Product module).

The FAQ module contains three tabs - one for each of the company's products. Inserting a tab is done by first initializing a tab strip and then implementing the tabs needed in the UI layer of the module (Figure 51).

```
<DNN:DNNTabStrip ID="MyDNNTabStrip" runat="server"  DefaultContainerCssClass="tabcontainer"
DefaultLabel-CssClassHover="tablabelhover" DefaultLabel-CssClassSelected="tablabelselected"
DefaultLabel-CssClass="tablabel" DefaultLabel-CssClassDisabled="tablabeldisabled"
SelectedIndex="0" TabRenderMode="All">
        <DNN:DNNTab ID="Tab1" runat="server" Label-Text="Text of tab label">
                <!-- Content inside the tab page -->
        </DNN:DNNTab>
```

Figure 51: Example of how to insert a tab into a module

The first line initializes the DNNTabStrip. The tab strip must have an exclusive Id (*MyDNNTabStrip*). Line two implements a tab. Each tab must also have an Id and a label text. The variable *Label-Text* is text that will be presented in the UI of the module.

In order to keep the module organized, we implemented a table that would list the question and answers. Remember that the data containing the questions and answers are stored in a database. We retrieve them by calling on *stored procedure* as mentioned in the section on the News module. Inside the table we display a question and then the answer. Below is an example of how this was implemented in the UI layer (Figure 52).

```
<asp:Label ID="lblContent" runat="server" Text='<%#
Server.HtmlDecode(DataBinder.Eval(Container.DataItem, "Question").ToString()) %>'/>
<div class="showhide" id="readMore<%# DataBinder.Eval(Container, "DataItem.ItemId") %>">
        <a href="#" onclick="showhideFAQ('<%# DataBinder.Eval(Container.DataItem, "ItemId")
        %>'); return(false);">
        <asp:Label runat="server" ID="readMore" resourcekey="readMore"  CssClass="Normal"
        s/></a>
</div>

<div style="display:none;" id="<%# DataBinder.Eval(Container.DataItem, "ItemId") %>">
        <div class="answerBox">
                <asp:Label ID="lblAnswer" runat="server" CssClass="Normal" Text='<%#
                Server.HtmlDecode(DataBinder.Eval(Container.DataItem, "Answer").ToString())
                %>' />
        </div>
        <div class="showhide"><a href="#" onclick="showhideFAQ('<%#
        DataBinder.Eval(Container.DataItem, "ItemId") %>'); return(false);">
                <asp:Label runat="server" ID="readLess" resourcekey="readLess"
                CssClass="Normal"/></a>
        </div>
</div>
```

Figure 52: Displaying question and answer with JavaScript

Like mentioned in the Prototype chapter, we have implemented a *JavaScript* function to hide the answer at first to save space and to make it easier for the user to find the appropriate question. Each question has a ''Get answer''-link (Label “ReadMore” on line 4). Clicking on this link will invoke the *JavaScript* function ''showHideFAQ'' (Line 3). This function will display the answer if hidden. The “showHideFAQ” function (Figure 53) takes one parameter, which is the question Id from the database.

```
function showhideFAQ(id){
        if (document.getElementById){
                obj = document.getElementById(id);
                 if (obj.style.display == "none"){
                        obj.style.display = "inline";
                 } else {
                        obj.style.display = "none";
                }
                obj2 = document.getElementById("readMore"+id)
                 if (obj2.style.display == "none"){
                         obj2.style.display = "inline";
                } else {
                        obj2.style.display = "none";
                }
        }
}
```

Figure 53: showHideFaq JavaScript

The function checks if the answer's html div tag for that Id is set to hidden. If not, the display value is set to *inline* (Line 5 and 11). Inline means that the element will be displayed with no line break before or after the element. When the answer is set to be *inline*, the link next to it also changes from "Get answer" to "Hide answer". Clicking this link will invoke the *JavaScript* "showHideFAQ", which will hide the answer and restoring the link to "Get Answer".

7.5.4 The Support module

Discussions related to the Support module

One crucial module that apparently will increase communication with the customers is the Support module. By using two types of prototyping techniques (Floyd, 1984), we were able to solve two challenges:

1. *Exploratory prototyping* helped us to create a visualization of the suggested functionality.
2. *Experimental prototyping* allowed us to explore technical limitations of involved systems surrounding the module.

Since the Support module requirement gathering involved several people, including those in Stavanger we decided to arrange a meeting face to face. By using exploratory prototyping we made mock-up of the module's UI (Figure 24). This module should serve as a fertilizer for further discussion. In the meeting, the prototype was very valuable. By using the prototype we were also able to predict needed data and input fields for further exploration of the technical requirements (Appendix B – Fields required in the Support module).

Like mentioned in the Prototype chapter, this module will serve as a mediator between the employees of the company and their customers, and the Customer Relationship Manager (CRM). By using experimental prototyping, we were able to explore the technical challenges of the module. Combining several technologies (CRM, Forms, Sharepoint, IFrames) were supposed to develop that bridge that support module will provide. We needed to look closer at each one of them to get an idea of how the data would be handled and transmitted. Furthermore, this gave us an insight of how the Support module should be designed and developed. However, as the project was on a tight time schedule and we were unable to finish the module on time.

8 Discussion

In this chapter, we will compare existing theories on Agile methodology (Chapter 4.2) and Participatory Design (Chapter 3) with our work (Chapter 6). To support the discussion we will refer to some of the empirical data (Chapter 7). We will also discuss the research question that was raised in the beginning of the thesis (Chapter 1).

We have shown how the Agile methodology, and especially Extreme Programming, helped us during the development of the prototype (Chapter 6). In order to implement the separate modules, we relied on the user stories expressed by the company representative. However, (Rittenbruch, McEwan et al., 2002) argues that one of the weaknesses of applying XP as a development method, is that the user stories may just be written by a company representative. If the representative who writes the user story is not representative for the general users of the system, the developers might create the "wrong" system. The process of selecting the user representatives is not well articulated in XP (Rittenbruch, McEwan et al., 2002). Often, and as we experienced, the selection is done by the company itself. Due to a limitation in available human resources and tight time schedules, the company may designate an individual(s) who is not optimal for this task.

In our case this lead to the development of a module that did not meet the users' expectations. During the development of the News module (Chapter 7.5.1), we were asked to do post-feedback modifications on the module because the requirements were gathered from a representative, rather than from future users. We interpreted the company representative as the representative for the actual end user of the system. Initially, we suggested that there should be three user groups; employees, customers and random visitors. Nevertheless, in meetings with the representative from the company, it was decided to initially only implement support for two groups: employees and customers. After the completion of the module however, we learned through meetings with actual users that the optimal solution would be the support of multiple user groups. One even suggested the support for extending the number of user groups dynamically through a separate user interface within the News module.

(Rittenbruch, McEwan et al., 2002) also argues that scenarios of PD are created in cooperation between the users and the designers, while as we saw; user stories are utterances of a representative without direct contribution from the designers.

In the design workshop we arranged, where multiple users and participants were involved, we used scenarios to express our understanding of how the company should solve a key communication problem. In collaboration with the participants from the company we created a new scenario of a proposed flow of communication (Chapter 7.4). We experienced that discussing the solution and creating a scenario had obvious advantages. By the time we started to discuss the Support module, which was the main topic of the design workshop, we already had a notion of both functionality and the user interaction within the module. Although we don't have any data to show that the design workshop started a process of thoughts among the participants, it seemed like the company's representatives were more certain in their decision-making at later stages of the process when discussing the Support module that lead us to believe that this originated in the workshop. Nor did the company representatives express any hesitation while making decisions related to the module. Their initial requirements also uncovered the fact that they had been giving this a lot of thought. In (Appendix B – Fields required in the Support module) we have listed the required fields in the module and in the CRM system that were given from the company's representatives in the first meeting scheduled to discuss the Support module. The list shows the determination of the users while presenting their suggested solution (Appendix B – Fields required in the Support module).

8.1 Future workshop as a method of clarifying requirements

Generally, PD focuses on the processes and means for creating a shared understanding between users and developers. Furthermore, the *future workshop* (Kensing and Madsen, 1992) has a goal of helping the participants in generating ideas of how the work situation should be in the future. During the design workshop, we experienced that by letting the users turn today's problem with the support situation into tomorrow's solution, the requirements were clarified in the process. When designing the communication flow for a proposed support solution, one of the users

stated a possible need for a FAQ list that could work as a first step in answering the most common problems of the company's customers.

> *"To maintain a – a FAQ – with the 10 most frequent support* enquires *– we have to log the* enquires *– that comes in – and then it would be nice to have a list where you can insert the questions into a given area"*

Company representative (Excerpt from the design workshop)

As a result, we already had a notion of the intension behind the module, as well as a concrete example of the how it could be used. In addition, generating the basic idea of a required function at this early stage meant that by the time we were to develop it, the company representative writing the user story already had a vision of how the module should be implemented. The representative may even have discussed the solution with other people in the company before suggesting it to us. Hence, a clarification of the requirements happened before we started planning the development related to the FAQ module, and the implementation did not require as much iteration as we experienced when working with the previous modules.

On the other hand, while working with the News module, we did not have the chance to arrange a design workshop directly related to the requirements of this module. Rather, we based the development on user stories stated by a single company representative, as suggested by (Beck, 2004). As a result, the ideas behind the initial requirements were not worked out in advance and we had to do several modifications in order to meet the company's needs. For instance, the decision of rearranging the fields of the article in the order of English first and then Norwegian (Chapter 7.5.1) might have been avoided if we arranged a design workshop prior to the development. Hence, our research suggests that arranging a future workshop in the early stages of development, may lead to less iteration during the development of a module.

8.2 Different levels of user participation

One of the key practices of applying XP is to have an on-site customer during the development phase (Deursen, 2001). In other words, the customer should come to the developers. The goal

of this is for the developers to understand customer wishes, maintain contact with the end users and clarify feature requests as well as technical aspects. PD, on the other hand, requires the developers to visit the workplace of the customer at the early stage of the process (Bødker, Grønbæk et al., 1995). (Deursen, 2001) describes how a single on-site customer can be problematic. In his work he discovered that the customer of the developers most of the time consist of a broader aspect of end users than it's possible to represent through a single on-site customer. Furthermore he argues that it involved a lot of creativity to make the groups of various interests "speak with one voice", i.e. through a single customer representative (Deursen, 2001).

During the development phase we did not only visit the workplace before getting started with the development. We were also offered a desk at the company's offices in order to collaborate closely with the company's representatives. As a result, we constantly received feedback and were able to get our questions answered from multiple sources. This was positive, as we did not have to rely on a single on-site customer to have the answers to all our questions. However, we experienced that having multiple sources of information to rely on also lead to insecurities. Because we received feedback from various representatives, it was challenging to decide what to prioritize in the next version of a module.

8.3 Demanding user participation in a real life setting

It's important to notice that this project did not consider the economic aspect, as no money was involved in the development process. The company did not pay us or our fellow researchers for being involved in the project. As we mentioned in the chapter on PD, it's unlikely to practice a high degree of user participation in a business context without some form of incentive to stimulate participation (as a minimum to cover for lost work time). Furthermore, applying these two approaches in a "real life" setting could be very demanding especially for the company who are dependent of taking their employees off the job to be a part of design workshops, interviews and other user centered approaches. Hence, the context of the situation may not be completely realistic. PD requires a huge amount of human resources and investment of time. We think that in a real life setting it might be hard to advocate these methods economically for a company

depending on making profit to survive. This is mainly because it requires the participants to spend time and resources on tasks that does not directly give the impression of being relevant to the progress of the project. Nor does it produce a single line of code. However, PD has been practiced successfully even outside the academic world. In(Hansson, Dittrich et al., 2006; Mørch and Skaanes, 2007), user participation is practiced through yearly meetings where the developers' customers are invited to share their opinions, improvement requests and general sharing of knowledge. (Hansson, Dittrich et al., 2006) also describe how the developers successfully have combined PD with agile methods in order to closely collaborate with the customer and implement the desired improvements.

8.4 Pair Programming and personality traits

In Chapter 4.3.3., we argued based on previous research that the developers' personality traits must be satisfied in order to take the full advantage of PP. It is argued that PP has two main drawbacks. First, the roles of "driver" and "navigator" may not always be appropriate. This is due to the fact that the while the "driver" is producing the code, the "navigator" attention may drift away. Second, pairing individuals without the proper personality alignment may lead to a breakdown in the interaction between the programmers. As a result, the knowledge transfer within the pairs might cease, leading to slower development than expected (Dick and Zarnett, 2002). Furthermore, it's stated that in order to exercise PP successfully, the individuals within the pairs should meet certain personality traits (Chapter 4.3.3).

Our developer team only consisted of two people. One of the reasons we think PP was successful in our case is that we can identify ourselves with the personality traits that (Dick and Zarnett, 2002) refer to as important. During the process we realized that communication between us, the developers, was just as important as the communication between us and company.

As our programming skills were approximately on the same level, we felt comfortable in expressing our ideas, suggestions and strategies without being concerned that the other part would either think that we were being too complicated or too basic. As a result, we were able to

quickly reach agreements even though there was no upper hand or organizational structure involved in the collaboration between us.

In our case, applying PP was never a big issue. We already knew each other from previous projects. Nevertheless, we have often discussed the following: What if we were not comfortable with working in team, or, what if started to argue without being able to agree on a solution? We feel that the answer to this is: In this project, where communication and collaboration was very important, we would eventually fail if we were unable to reach common agreements on vital points of the process.

8.5 Combining the methods

Notice that it is not common to compare XP and PD side by side (Rittenbruch, McEwan et al., 2002). XP is a development method consisting of several processes focusing on development, while PD is a research field with focus on learning and understanding between the system developer and the user. In other words one is more product-oriented, whereas the other is more process-oriented. Nevertheless, there are similarities that can be compared. Beside its prototype-based approach, both of these methods are very people centric and rely on a high degree of user and customer involvement. However, there are significant differences when considering in which way and how the users are integrated in the development process. We think that these methods are not directly supplementary, but rather that applying PD while working with the XP method, results in a better outcome, for both product and process.

In conclusion, XP with the support of PD enabled us get a broader perspective of the company and their current situation. In particular, this broadened perspective for not only us as developers, but also the company's representative, allowed us to uncover the requirements through close collaboration.

The active involvement of the users recommended by PD gave us a different, "extended" kind of information than what XP's requirements to user participation would do on its own. It's our firm belief that the prototype presented in this thesis, is more complete than it would have been if we decided to apply the XP method alone, without being accompanied by PD. Furthermore, we

believe that combining these two approaches in our context, where the requirements were not settled at the outset and a gap between the developers and user representatives were present; gave us a broader understanding of the context where the system was made.

9 Further development

In this chapter we will describe possible extensions and improvements of the web portal.

The web portal as it is today does not communicate with any other third-party systems. However, it's worth noticing that in today's world of computer systems it's a tendency that most software should be able to communicate with each other. In the KIKK project we worked on a support solution that was supposed to function as a mediator between the web portal and a CRM system. The idea was that the CRM system should hold all the necessary information about the company's customers, such as name, customer status and other relevant data for the customers (See Appendix B – Fields required in the Support module). In addition, the CRM was supposed to store the support requests made by the customers.

The goal of the Support module was that it should function as a mediator between the web portal's interface and a CRM system. In other words, a communication channel between two important customer-centered systems. By interacting with the Support module, the company's customers should be able to send a support request, check status for previous requests or be presented the solution to previously solved support issues.

On the other side, the Support module would pass these requests on to the CRM system, where the company's employees should receive requests, answer them or search in a database of previously solved problems. A requirement to achieve this communication would be a constant synchronization of data between the Support module and the CRM system.

Another requirement to the Support module was worked out during the design workshop in collaboration with the company's representatives. The request was that the Support module should be able to determine the importance of the customer's request. This was going to be done by analyzing the customer's status as well as the support request itself. Every customer of the company has a certain status. Some customers even have a deal that guarantees support within 24 hours. Naturally their requests must be considered first. Secondly, by dividing the support requests into sub categories, such as installation problem, general questions and ideas for further development, the Support module should be able to channelize the requests and

pass them on the right people within the company. The goal of this structure would be that the support requests were to be handled by the employees with the best knowledge of the different domains.

Remember that today the Support module only exists as a prototype (Chapter 6). In order to achieve these improvements, an appropriate technical solution has to be worked out.

10 Summary and Conclusions

In the research process we have investigated how Extreme Programming and Participatory Design have been used for the design and development of modules to be part of a web portal employed in a business context.

We started the thesis with an introduction to the context (Chapter 2). The company involved in the KIKK project experienced that the knowledge kept by individuals was not shared among the employees and the company's customers. The company expressed a desire to improve the flow of communication both internally and externally. The goal of the KIKK project was to help the company in that process. Developing the web portal, which was our main task, was one the suggested solutions.

The theoretical part of the thesis (Chapter 3 and 4) explained the details behind PD and the Agile Methodology. In the chapter on PD we showed how involving the users at all stages of the development process, allowed them to contribute in a way that makes it possible for users and developers to learn from each other. We also looked at how different techniques, for instance design workshop, can clarify the requirements needed to develop the appropriate system.

In the chapter on software engineering, we compared the "traditional" approach of Plan-driven with the Agile approach. Our conclusion was to take advantage of one of the agile methods as the requirements were vague and likely to change during the process. In this sense, XP seemed promising due to its flexibility in handling changes. We learned that XP's four principles of communication, simplicity, feedback and courage (Beck, 2004) were not just a set guidelines but rather a must in order to succeed in developing a web portal.

In the chapter on Programming framework (Chapter 5), we explained the techniques and tools required to implement an open source web portal framework. We briefly looked at the advantages of using open source applications and relying on component based development. In this chapter, we also the framework we utilized in our thesis, namely DotNetNuke.

We have, besides setting up the web portal, developed four modules that met the specific needs of the company (Chapter 6). Additionally, we have given detailed examples of how these modules were developed through close collaboration with the customer. We have also looked at how we applied the techniques of XP while going through the different phases of the development of each module (Chapter 7).

In the discussion chapter (Chapter 8), we looked at the overall process of development while answering the research questions by exploring their pros and cons. We described how implementing a web portal with the use of XP and PD leads to the development of a system more likely to be accepted by the users. We also argued that by supplementing XP with the techniques of PD, all the involved participants got a broadened understanding of the context we were working in. We discovered that when we applied these techniques less post-feedback alterations were necessary.

We find our exploration and customization of the DNN web portal in a business context to be a success. We went from being totally new to the programming environment to become expert programmers. Through the development of the modules, we also conquered the challenge of communication with various stakeholders and participants. Even though we sometimes met conflicts of interests, we feel like we were able to analyze the requirements and create a product closely linked to what the users had in mind.

The development itself has at times been very demanding, and DNN is not recommended for those unwilling to try and fail in order to reach an appropriate result. However, after finishing the first module, we noticed that things went smoother. We no longer had to search the net for hours to find the answers we were looking for. Also, as we became more comfortable with the architecture of DNN, we realized that the organization in layers is quite convenient.

Although the web portal we developed is not finished and should be considered a prototype, we still believe the company finds our work to be worth waiting for. As we mentioned in the beginning of this thesis, the notion of creating a web portal started as a general idea. We believe that the prototype presented in thesis will give the company's representatives a good way to visualize what their future Support system should be implemented. At the same time, our

project has answered many questions, and hopefully our work in the KIKK project can reduce misunderstandings, especially related to communication, in similar projects in the future.

11 Bibliography

Ambler, S. W. (2004). The Object Primer : Agile Modeling-Driven Development with UML 2.0, Cambridge University Press.

Ambler, S. W. and R. Jeffries (2002). Agile modeling: effective practices for extreme programming and the unified process, John Wiley \& Sons, Inc.

Bauer, F. L., L. Bolliet, et al. (1968). Software Engineering - Report on a Conference Sponsored by the NATO Science Committee, Garmisch, Germany.

Beck, E. (2002). "P for Political: Participation is Not Enough " Scandinavian journal of information systems **14**(1): 77.

Beck, K. (2004). Extreme Programming Explained: Embrace Change (2nd Edition), Addison-Wesley Professional.

Bjerknes, G. and T. Bratteteig (1995). "User Participation and Democracy: A Discussion of Scandinavian Research on System Development." Scandinavian Journal of Information Systems **7**(1): 73.

Boehm, B. (1986). "A spiral model of software development and enhancement." SIGSOFT Softw. Eng. Notes **11**(4): 14-24.

Bossen, C. (2006). Participation, power, critique: constructing a standard for electronic patient records. Proceedings of the ninth conference on Participatory design: Expanding boundaries in design - Volume 1. Trento, Italy, ACM Press.

Bratteteig, T. (1997). Mutual learning: enabling cooperation in systems design. Proceedings of the 20th information systems research seminar in Scandinavia (IRIS'20), Hankø, Norway.

Bravo, E. (1993). Participatory Design: Principles and Practices. D. Schuler and A. Namioka, Lawrence Erlbaum Associates, Inc.**:** 3-12.

Brynhildsen, C. and A. Mørch (2005). Brukermedvirkning i design av e-læringsportal. Integrert e-læring i bedriften: Pedagogikk, teknologi, organisasjon. A. Mørch and I. Solheim, Unipubforlag**:** 81-94.

Budde, R., K. Kuhlenkamp, et al. (1992). Prototyping: An Approach to Evolutionary System Development, Springer-Verlag New York, Inc.

Bødker, S. and K. Grønbæk (1992). Design in action: from prototyping by demonstration to cooperative prototyping. Design at work: cooperative design of computer systems, Lawrence Erlbaum Associates, Inc.**:** 197-218.

Bødker, S., K. Grønbæk, et al. (1995). Cooperative Design: Techniques and Experiences From the Scandinavian Scene. Human-Computer Interaction: Toward the Year 2000, Morgan Kaufmann Publishers Inc.**:** 215-224.

Bødker, S. and O. S. Iversen (2002). Staging a professional participatory design practice: moving PD beyond the initial fascination of user involvement. Proceedings of the second Nordic conference on Human-computer interaction. Aarhus, Denmark, ACM Press.

Carter, R. A., A. I. Ant, et al. (2001). Evolving Beyond Requirements Creep: A Risk-Based Evolutionary Prototyping Model. Proceedings of the Fifth IEEE International Symposium on Requirements Engineering (RE '01), IEEE Computer Society.

Clement, A. (2005). "Participatory Design." Retrieved October 28th, 2007, from http://cpsr.org/issues/pd/.

Cockburn, A. (2002). Agile software development, Addison-Wesley Longman Publishing Co., Inc.

Dahl, O.-J. and K. Nygaard (1966). "SIMULA: An Algol-based Simulation Language." Communications or the ACM **9**(9): 671-678.

Deursen, A. v. (2001). "Customer involvement in extreme programming: XP2001 workshop report." SIGSOFT Softw. Eng. Notes **26**(6): 70-73.

Dick, A. J. and B. Zarnett (2002). Paired Programming and Personality Traits. Third International Conference on eXtreme Programming and Agile Processes in Software Engineering, Alghero, Sardinia, Italy.

Ehn, P. and M. Kyng (1992). Cardboard computers: mocking-it-up or hands-on the future. Design at work: cooperative design of computer systems, Lawrence Erlbaum Associates, Inc.**:** 169-196.

Floyd, C. (1984). A Systematic Look at Prototyping. Approaches to Prototyping. R. Budde, K. Kuhlenkamp, L. Mathassen and H. Züllighoven. Berlin, Spirnger-Verlag**:** 1-19.

Fowler, M. (2000). "The New Methodology." Wuhan University Journal of Natural Sciences **6**(1): 12-24.

Greenbaum, J. and M. Kyng (1992). Introduction: situated design. Design at work: cooperative design of computer systems, Lawrence Erlbaum Associates, Inc.**:** 1-24.

Griss, M. L. and G. Pour (2001). "Accelerating development with agent components." Computer **34**(5): 37-43.

Hanseth, O. and K. Lyytinen (2004). "Theorizing about the Design of Information Infrastructures: Design Kernel Theories and Principles." Sprouts: Working Papers of Information Environments Systems and Organizations **4**(4): 207-241.

Hansson, C., Y. Dittrich, et al. (2006). How to Include Users in the Development of Off-the-Shelf Software: A Case for Complementing Participatory Design with Agile Development. Proceedings of the 39th Annual Hawaii International Conference on System Sciences - Volume 08, IEEE Computer Society.

Hayes, S. and M. Andrews (2004) "An Introduction to Agile Methods." **Volume**, DOI:

Herzog, C. J. (2005). Pair programming and learning. Institutt for Informatikk. Oslo, Univeristy of Olso. **Master:** 135.

Highsmith, J. (2002). Agile software development ecosystems, Addison-Wesley Longman Publishing Co., Inc.

IEEE, I. o. E. a. E. E. (1983). "Standard Glossary of Software Engineering Terminology " Standard **729-1983**.

Kensing, F. and J. Blomberg (1998). Participatory Design: Issues and Concerns, Kluwer Academic Publishers. **7:** 167-185.

Kensing, F. and K. H. Madsen (1992). Generating Visions: Future Workshops and Metaphorical Design. Design at Work: Cooperative Design of Computer Systems. J. Greenbaum and M. Kyng, Lawrence Erlbaum Associates, Inc.**:** 155-168.

Kuutti, K. (1995). Work processes: scenarios as a preliminary vocabulary. Scenario-based design: envisioning work and technology in system development, John Wiley \& Sons, Inc.**:** 19-36.

Mayr, H. C., M. Bever, et al. (1984). Prototyping Interactive Application Systems. Approaches to Prototyping. R. Budde, K. Kuhlenkamp, L. Mathassen and H. Züllighoven. Berlin, Spirnger-Verlag**:** 105-121.

Mørch, A. (1995). Application Units: Basic Building Blocks of Tailorable Applications. Proceedings of the 5th Int'l East-West Conf. on HCI, Moscow.

Mørch, A. (2005). Integrert e-læring i bedriften: Pedagogikk, teknologi, organisasjon, Unipubforlag.

Mørch, A. and M. A. Skaanes (2007). Design and Use of an Integrated Work and Learning System: Information Seeking as Critical Function. Learning Across Sites: New Tools, Infrastructures and Practices. S. Ludvigsen, A. Lund, I. Rasmussen and R. Säljö, Pergamon Press.

Mørch, A., H.-R. H. Åsand, et al. (2007). The Organization of End User Development in an Accounting Company. End User Computing Challenges and Technologies: Emerging Tools and Applications. S. Clarke**:** 102-123.

Netcraft. (2007). "Netcraft Web Server Survey." Retrieved October 28th, 2007, from http://news.netcraft.com/archives/web_server_survey.html.

Nygaard, K. (1979). The iron and Metal Project: Trade Union Participation. Computers Dividing Man and Work - Recent Scandinavian Research on Planning and COmputers from a Trade Union Perspective. Å. Sandberg. Malmø, Sweden, Swedish Center for Working Life.

Nygård, K. and A. Mørch (2007). From knowledge management to collective knowledge advancement: product evolution as boundary-crossing in mentoring and user participation. ICCE 2007.

O'Reilly, T. (2005) "What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software." **Volume**, DOI:

Parnas, D. L. (1979). On the criteria to be used in decomposing systems into modules. Classics in software engineering, Yourdon Press**:** 139-150.

Pree, W. (1997). Component-based software development-a new paradigm in software engineering? Software Engineering Conference, 1997. Asia Pacific ... and International Computer Science Conference 1997. APSEC '97 and ICSC '97. Proceedings.

Ravichandran, T. and M. A. Rothenberger (2003). "Software reuse strategies and component markets." Commun. ACM **46**(8): 109-114.

Raymond, E. S. (2001). Cathedral \& the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary, O'Reilly \& Associates, Inc.

Reenskaug, T. (1979). Models - Views - Controllers, Xerox PARC.

Rittenbruch, M., G. McEwan, et al. (2002). Extreme Participation - Moving Extreme Programming Towards Participatory Design. Seventh Biennial Participatory Design Conference, Sidney, Australia, CRC for Enterprise Distributed Systems Technology.

Rittenbruch, M., G. McEwan, et al. (2002). Extreme Participation - Moving Extreme Programming Towards Participatory Design. Seventh Biennial Participatory Design Conference, Sydney, Australia, CRC for Enterprise Distributed Systems Technology.

Sametinger, J. (1997). Software engineering with reusable components, Springer-Verlag New York, Inc.

Schuler, D. and A. Namioka (1993). Participatory Design: Principles and Practices, Lawrence Erlbaum Associates, Inc.

Silva, M. and A. Breuleux (1994). "The Use of Participatory Design in the Implementation of Internet-based Collaborative Learning Activities in K-12 Classrooms." Interpersonal Computing and Technology: An Electronic Journal for the 21st Century **2**(3): 99-128.

Sommerville, I. (2004). Software Engineering, Addison-Wesley Longman Publishing Co., Inc.

Treese, W. (2006). "Web 2.0: is it really different?" netWorker **10**(2): 15-17.

Tsai, J. J. P. and T. Weigert (1989). Exploratory prototyping through the use of frames and production rules. Computer Software and Applications Conference, 1989. COMPSAC 89., Proceedings of the 13th Annual International.

Walker, S., J. Brinkman, et al. (2006). Professional DotNetNuke 4.0: Open Source Web Application Framework for ASP.NET 2.0 Wrox Press

Weber, S. (2004). The Success of Open Source, Harvard University Press.

Wells, D. (1999). "The Rules and Practices of Extreme Programming." Retrieved October 28th, 2007, from http://www.extremeprogramming.org/rules.html.

Wheeler, D. (2004). "Why Open Source Software / Free Software (OSS/FS)? Look at the Numbers!" 112.

Williams, J., P. Clegg, et al. (2005). The Advantages of Adopting Open Source Software. Expanding Choice: Moving to Linux and Open Source with Novell Open Enterprise Server**:** 264.

Williams, L., R. R. Kessler, et al. (2000). "Strengthening the Case for Pair Programming." IEEE Softw. **17**(4): 19-25.

12 Appendix A – Daily log for Damir Nedic and Espen Olsen

22. januar

Karl var syk

23. januar

Startet dagen med et møte der Karl fortalte om hvordan kommunikasjonen fungerer i Selskapet per i dag (med servere, firewalls osv) og hvordan han ønsker at det skal bli i fremtiden.

Karl foreslo at vi startet med å lage en modul for DotNetNuke (DNN) der en ikke-teknisk sekretær kan legge inn en nyhet. Denne nyheten skal skrives både på norsk og engelsk. Det skal også være mulig å velge om nyheten skal vises for kunder (innlogget/ikke innlogget) og for ansatte.

Før vi kunne begynne utviklingen av denne måtte vi sette opp den virtuelle serveren vi skal kommunisere mot frem til det blir kjøpt inn en ny server. Pga av noen tekniske problemer og en del installeringsnødvendigheter tok dette arbeidet hele dagen.

29. januar

Startet arbeidet med å utvikle nyhetsmodulen. I første omgang prøvde vi å lage en frittstående ASP.NET-applikasjon som tar i mot data og putter disse i en database. Ettersom vi ikke har så mye erfaring med ASP.NET gikk dette arbeidet ganske langsomt. Hadde også noen problemer med å kommunisere med SQL-databasen på serveren. Dette løste vi til slutt da vi endret connection-stringen med et portnummer.

30. januar

Vi fortsatte arbeidet med nyhetsmodulen. Karl ville gjerne ha en statusrapport på fremgangen ettersom han skulle rapportere videre til "høvdingen". Usikker på hvem dette er.

5. Februar

Vi fortsatte arbeidet med nyhetsmodulen. Espen forsøkte å lage en modul ut i fra DNNs starter kit. Damir fortsatte med utviklingen av nyhetsmodulen. Mot slutten av dagen laget vi en modul av vår frittstående ASP.NET. Dette fungerte bra, men vi fikk problemer da vi prøvde å legge til modulen som skal vise nyhetene. Dette var fordi vi brukte de samme namespacene i de to modulene. Måten vi jobber på er at vi har alle filene lokalt på pc, hvor vi implementerer, kjører, tester og debugger. Når modulen er ferdig overføres alle filene til virtuelle pc-en, zippes de sammen og importeres det i dnn.

6. februar

I dag klarte vi å hente brukernavn fra DNN og bruke den i News modulen. Vi også prøvde å skjønne/lage funksjonen for å hente rå image data uten å lykkes. Damir måtte gå etter lunsj og Espen lagde en alternativ løsning til bilder henting, som er å lagre bilder i filesystemet i stedet for databasen. Dette vises seg å være bedre og enklere løsning. Bedre på den måten at serveren vil bruke mindre tid for parsing av rå data og klienten vil bruke mindre tid for konvertering rå data til bildet. Isteden lager vi bare path-en til bildet i databasen.

12. februar

I dag presenterte vi News og NewsViewer modulene. Karl og Jens var veldig fornøyd. Resten av dagen brukte vi på å lage screenshots som Karl kommer til diskutere med Selskapet i Stavanger i morgen(13.02).

13. februar

I dag begynte vi arbeidet med å gjøre det mulig å slette og endre en bestemt nyhetsartikkel. Vi bestemte oss for å slå sammen modulene til en, slik at man kan endre, slette eller legge til en nyhetsartikkel i en modul. I tillegg har vi tenkt til å beholde modulen som kun lar deg legge til en nyhetsartikkel. Denne modulen kan legges på en side som kun er tilgjengelig for administratorer.

19.februar

I dag prøvde vi å skjønne hvordan vi henter ting fra databasen. Vi klarte det til slutt men sliten med kalenderen. Vi prøver å bruke Pop-up Calendar men det går ikke. Espen har laget mulighet til å lege nyhet for kunder eller ansatte. Vi venter på svar fra forumet til DNN angående pop-up calendar. Ellers en ganske lite produktiv dag.

20.februar

I dag fikk vi gjort mye. Både kalenderen og brukergrupper funker. La også inn en sperre som gjør at kun utvalgte brukergrupper kan endre nyheter. Vi har bruker ikke lenger News tabell. Isteden har vi laget to helt like tabeller. NewsCustomer og NewsEmployee. Det som skiller de fra det opprinelige News tabellen er at image feltet tar imot en Varchar(50) og ikke rå data. Ellers en del endringen på stored procedures.

22. februar

Har nå sittet en lang kveld og prøvd å finne ut hvordan vi skal ordne delt visning av informasjon (kunde/ansatt), samtidig som alt skal være tilgjengelig for administrator. I tillegg skal det være søkbart. Løsningen først var å dele opp informasjonen i to separate tabeller med helt like felter. Dette virker dumt. Så vidt jeg har skjønt nå er det vi selv som bestemmer hva som skal kunne søkes på i modulen. Da må det være letter å bare finne ut hvilken brukergruppe brukeren tilhører og hente informasjon fra en felles tabell. Vi må bare merke nyheten med enten for kunde eller ansatt. Admin kan naturligvis se alle nyheter. Har lagt inn mulighet for å slette nyhet (Man må gå inn på edit og deretter velge slett).

- Lagt inn sjekk på at felter er fyllt ut.
- lagt inn mulighet for å velge publish dato, samt oppdatert spørringer, så de henter rette artikler
- lagt inn mulighet for å trykke 'legg til ny artikkel' fra News viewer

26.februar

I dag hadde vi møte med Karl. Vi presenterte resten av Nyhets modulen vi har laget. Vi fikk ordnet "Read more.." funksjonalitet. Vi har også fikset at når vi leser en nyhet fra DB så legges den i et objekt. På denne måten kan vi enklere skrive ut / til dataene til nyheten. Problemet til nå er at vi ikke klarer å hente nye verdier når en nyhet oppdateres. (Hehehe: dette har vi fikset nå :)

27.februar

I dag ble vi ferdig med NewsViewer modulen. Ordnet også opp slik at brukergruppen *News writer* kan laste opp bilder.

5.mars

I dag begynte vi på en annen modul "ProductList". Den skal inneholde en liste over alle produktene selskapet lager i form av linker. Når man klikker på en av de linkene åpnes det mer utfyllende info om det produktet i en ny modul. (Dette er spennende for oss siden vi må lage slik at de to modulene kommuniserer med hverandre). Vi måtte gå kl. 13 pga møte med KIKK.

6.mars

Idag har vi hatt et lite møte med Karl. Han informerte oss om at server-pc er kjøpt. Videre sa han at vi skulle bruke litt tid for å sette oss inn i CRM og hvordan vi skal kunne kommunisere mot den. Damir gikk kl. 12:30 og Espen brukte resten av dagen for å finne ut om SDK-plug in og CRM. Han prøvde å kommunisere med et CRM-image men fikk det ikke helt til. (Det gikk som en "stand alone" webside - altså ikke via DNN-portalen)

12. mars

I dag møtte vi Michael og fortalte litt om våre erfaringer med DNN så langt. Vi fikk ikke gjort noe på utviklingen siden vi måtte være på møtet med KIKK og Selskapet.

13.mars

Møte med Karl: kanskje reise til stavanger for å snakke med Ivar og Michael. Han har laget support system (forrige). Den kan si noe om hva man ikke skal gjøre.

Fane system. Lage til mandag. Dummy module. Trenger ikke å gjøre så mye (som å hente ting fra DB osv videre). Bare for å ha noe å presentere i Stavanger til Ivar og Michael.

Case attributes

- Description : beskrivelse av casen
- Type: radio button eller felt.
- Solution: Løsning
- Case type: kanskje ikke nedvendig
- Stryke: due date, raised by, priority, status, created date, og resten (forecast)
- Vedlegg skal inn..

~~Sjekke faner med DotNetNuke om det går an å lage.~~

Spørsmål og svar. Spørsmålet vises også trykker man på + og åpnes svaret.

Mest mulig fram til 19. april. Karl sier at kanskje juni skal alt lanseres.

Resten av dagen: Espen forsøkte å implementere tabber i en modul, men lyktes ikke særlig med det. All koden som var gitt var i VB. Forsøkte å oversette, men det var lite vellykket.

Laget en dummy-versjon av Support-modulen, som vi kan vise frem i Stavanger.
Eksperimenterte også med resource-filer, istedenfor å skrive tekst direkte i modulen. Dette vil gjøre det enkelt å oversette modulen til flere språk. Alt man trenger å gjøre er å lage en resourcefil for hvert språk man ønsker å benytte. Dette er ganske enkelt, men det vil ta en del tid å gjøre det.

Karl bestilte billett til oss slik at vi drar til Stavanger på tirsdag. Her skal vi prate med Ivar og Michael.

19.mars

I dag har vi laget "dummy" Products og Support modul. Product viser treeview og når man klikker på linkene kommer de opp i vinduet ved siden av. Support er en modul hvor det skal være mulig for innloggede brukere å stille spørsmål. Nå inneholder den 4 faner.(Faner er ikke helt sånn som vi vil ha det, men vi har ikke brukt så mye tid på det.)

20.mars

Stavanger. Yeah. I dag har vi pratet med Ivar, Michael og Julie. Her kommer en oppsummering.

Tilstede: Ivar, Karl, Damir, Espen, Michael, Julie

- Felles forståelse hva som er viktig og ikke viktig. Organisering av data. (Vi har ikke noe med det men vi kan komme med forslag).
- Kvalitetssikring – en kunde skal aldri vente mer 48 timer, sølvkunde skal kanskje vente maks 6 timer og gullkunde 2 timer.
- Portalen: Skal settes i gang (hvis ikke for hele verden, så internt).
- Supportdelen skal kunne huske profilen til kunden. Dette kan gjøres i form av en cookie (Når man vil sende en request så legges det alt ”det” unødvendige).
- Finne en metode som gjør at vi kommuniserer med CRM. Søke på muligheter med share-point. (webparts må lages for share-point – iframe må ha.) Dette må vi sjekke.
- FAQ – separert modul. Først velger du og da listes spørsmål og svar.
- Nyhetsarkiv - lage en modul / eller modifisere News slik at nyhetsarkivet vises.
- Sjekke ut om det går an å legge til brukergruppe for forskjellige brukergrupper (Hierarki av brukere. Brukergruppe arver egenskaper av andre brukergrupper).
- Sjekke rettighetene på filer. (At noen brukergruppen har tilgang til filene).
- Filopplasting på support modulen.
    - Lytteknapp på om kunde har lastet ned software.
- Vise bilde med ingress. (To typer bilder eller samme bildet bare resized).
- Support modul (Enkel kontakt) – hvis bruker er innlogget så sender den bare, hvis ikke innlogget så fylle ut noen enkle spørsmål også sende.
- "Nett analyser" -system. Hvem som er vært på nettstedet. Sjekke om deres N???(husker ikke navnet på hva de bruke nå) kan importeres i DNN.
- Hvis vi ser noen moduler som er nyttige er det bare å begrunne og da vurderes den for innkjøp.

Support module

- show current support cases.
    - vises bare titles som en liste også når man klikker på title utvide den seg

"*SharePoint* - portal i seg selv. Og du kan ha iFrame i DNN som speiler det SharePoints-webparts viser"

26.mars

I dag begynte vi å lage en ny modul (enda en) som heter FAQ. VI klarte å fikse det med tabstripes. Det skal lages 1 tab for hver solution og under dem skal de ha sine Questions og Answers. Vi må også lage taber for forskjellige versjoner. Ole påpekte også at vi ikke skal lage "den" midlertidig løsningen sånn som Michael foreslå men fokusere å kode direkte mot CRM.

Forsøkte å lage et system der vi delte informasjonen inn i 3 tabeller. En tabell som holder oversikt over spm/svar, en som holder oversikt over spm/svar og de forskjellige produkter med versjoner og en tabell som holder oversikt over produktene med versjoner.

Dette viste seg som svært krevende å holde oppdatert når det skal kjøres.

Slik som vi ser det har vi nå 2 muligheter:

1. Implementere det slik vi har gjort nå. Det vil ta tid ettersom hvert kall til databasen krever en del (men veldig enkel) programmering.
2. Blåse i å ha 3 tabeller og heller legge all informasjon inn i en tabell. Jeg ser for meg at den kan være slik:

ItemId | SolutionName | SolutionVersion (denne kan være en enum('3.4','3.5','Alle') der alle sier at spørsmålet gjelder for alle versjoner) | Spm | Svar | ... osv

2.april

Først hadde vi møte med Karl. Han påpekte at alt vi lurer på kan / burde tas opp med Ivar. Videre har Michael satt opp server til oss men vi vet ikke hvordan bruker vi den. Karl skal sette

opp image for Microsoft Forms slik at vi kan begynne å kommunisere med CRM. Forms skal brukes som mellomledd mellom CRM og DNN. (Da er alt lovlig?)

10.april

I dag har vi ordnet det med Stavanger - Remote Desktop. Vi prøvde å installere DNN (4.5) på dnn-serveren men fikk ikke det helt til. Vi klarte ikke å få den nye DNN til å kommunisere med databasen.

Vi har fått tilgang til en image av server med Microsoft Forms installert. Det skal vi se nærmere på neste mandag. Ellers har vi også ordnet nyhetsmodul med tanke på resizing av bildet og noen andre små ting. Ellers har Espen pratet med Ole som sa at hvis vi trengte hjelp angående masteren så er det bare å spøre.

16.april

Møte med Karl og Jens. Vi har endelig fått opp DNN på StavangerIp'en. Vi har også overført 2 moduler (News og Faq). De fungerer veldig bra. Videre har vi prøvd å skjønne hvordan vi skal kommunisere mellom CRM, sharepoint og Microsoft Forms. Vi tror at vi må lese litt mer om det før vi kan gjøre noe som helst.

23.april

Idag har vi hatt møte med Ole og Karl angående vår fremtid i selskapet. Vi ble enige om at vi tar en pause fram til 1.juni. Etter det skal vi jobbe mer med dette (en slags sommerjobb). I denne perioden vil Selskapet legge til rette at det skal bli enklere for oss når vi kommer tilbake. Karl skal ha et møte med www.gasta.no som vi skal kanskje være med på. Videre påpekte Ole at vi må dokumentere mest mulig (i form av manualer og kildekode). Ole også sa at det skal lages to forskjellige portaler. En for europiske-asiatiske markedet og andre for USA. Da skal man velge hvilke side man ønsker å gå til fra hovedsiden som vil antakelig bare inneholde 2 flagg.

Timeliste

Dato	Damir	Espen
22.januar	0	0
23.januar	8	8
29.januar	8	8
30.januar	8	8
5.februar	8	8
6.februar	3,5	8
12.februar	8	8
13.februar	3	7
19.februar	9	9
20.februar	4	8
26.februar	9	9
27.februar	4	8
5.mars	4	4
6.mars	4	8
12.mars	8	8
13.mars	3	7
19.mars	10	10
26.mars	10	10
27. mars	4	8
2.april	10	10
3.april(påskeferie)	0	0
9.april(påskeferie)	0	0
10.april	4	8
16.april	9	9
23.april	6	6
Totalt	144,5	177

ToDo-liste

News module

- ~~News modulen mangler tekstfelt for engelsk~~
- ~~Databasen trenger modifisering med tanke på bilde path og engelsk felt~~
- ~~Vi trenger også å finne en løsning på sletting og oppdatering av nyheter (det jeg tenker er å ha en update eller delete link slik at når man trykker på den åpnes nyheten for redigering eller slettes den)~~
- ~~Mulighet til å lage nyhet for enten kunder eller ansatte~~
- ~~Skalere bildet til en fast størrelse~~
- ~~Viser brødtekst når man trykker på "Read more..."~~
- ~~Kikke på om modulen kan vises kun for enkelte brukere.~~
- ~~DatoAdded skal skrives inn - per default skal det blir dagens dato også mulighet til å endres~~
- ~~Sjekke om tekst eksisterer - hvis ikke gi beskjeden til brukeren~~
- ~~Skille mellom norsk og engelsk felter~~
- ~~Norsk språk for alle modulene - sjekke hvor mye den koster og om den finnes i det hele tatt~~
- ~~Det skal lages norsk engelsk (først skal det skrives på norsk også på engelsk)~~
- ~~Oppdatering av nyhet~~
- ~~Innloging modul (med tanke på 3 brukergrupper - forskjellig database)~~
- Søk funksjon (søke i nyheter)
- Opplasting av bilder skal være optional
- ~~Bilde resizes i det brukeren laster opp~~
- ~~Gi privlegier til brukergruppen "newswriter" til å legge inn bilder~~

Søk

- Den funker ikke.. den gi resultat kun når søkeordet er "nyhetstekst"

Products module

- ~~Modul: Products (Modul med linker til en annen modul (Produktdetaljer))~~
- ~~Modul: ProductDetails (Modul som viser detaljer om et produkt valgt i Products-modulen (Denne kan vi lage ved å endre på newsviewer))~~
- ~~Hvordan strukturere Products (foldere med underpunkter)~~

- Hvis kunde velger å sende skjema på et produkt om eventuelt kjøp, mulighet for å sende linken fra hvilken side det kommer fra og hvilket produkt det dreier som
- Skal listen over produkter/versjoner være statisk? Vil ta tid å utvikle full støtte for oppdatering av database.

Support module

- Kontakt med database
- Kontakt med CRM
- ~~Legge til ordentlige faner~~
- Cookies
- Velg produkt - automatisk visning av versjoner - autovisning av build (settes i cookie)
- Build må lagres i database
- Vedlegg - uansett form/størrelse
- Velg om supportsaken skal være offentlig eller skjult

FAQ modul

- ~~Modul som separat fra CRM-systemet gjør det mulig å legge til spørsmål og svar.~~
- Spørsmål og svar linkes til de forskjellige programmene selskapet tilbyr
- Skal tabber leses fra database eller være statisk (Kommer det flere produkter/versjoner)?

Easy contact

- Modul som gjør at man en hvilken som helst side kan sende en kort beskjed til selskapet. Denne skal finne ut av hvilken side brukeren var på da han klikket på kontaktboksen for å spørre om noe.

Generelt

- Se på flere brukergrupper. Lage en tabell til med gruppe-tilhørighet
- forandre tekst (engelsk/norsk) til flagg/ikoner
- Til google: sjekke om det går an når bildet sendes at linkes til selskapetshjemmesiden - forbedre treff.
- Skrive dokumentasjon
- Pakke/zippe de ferdige modulene
- ~~Kommentere kildekoden~~

- Skrive evn. merknader - hva som burde gjøres, være obs på osv.
- ~~Lage to faner → Contact og About~~
- Kanskje fjerne den obligatoriske funksjonen at man må skrive inn nyheten både for norsk og engelsk → må undersøkes

13 Appendix B – Fields required in the Support module

Portal

•Number	auto	
•Description	kort tekst	
•Product		cookie
•Version		cookie
•Build		cookie
•Solution (Suggested/Actual/Workaround)	tekst	
•Case type (Question, Improvement Request, Complaint, Support Request)		
•Company Name	input til CRM	
•Raised by (Person)	input til CRM	
•Client acceptance flag	integer	
•Created date	datetime	
•History & Activities as in standard CRM solution!)		
•Attachments	blob	

CRM

•Type (Development vs. Support)
•Due date
•Case type (Question, Improvement Request, Complaint, Support Request)
•Priority
•Status (Recorded, In Progress, Pending Information, Rejected, Approved)
•Forecast start date
•Forecast finish date
•Estimated Hours
•Actual Hours
•Modul (innenfor produkt)
•Funksjonsområde (i arbeidsprosessen)