UPTEC IT 09 006
Examensarbete 30 hp
Februari 2009

Evaluation of Exor´s Work Process:

Methods and Models of Usage-Centered Design

Petter Höglund

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Evaluation of Exor's Work Process: Methods and Models of Usage-Centered Design

Petter Höglund

These days the usability of software has risen to become one of the most importan aspects of a system. The importance of software usability will grow since more and more laymen will use software in any form in their everyday life. The functionality of the software will still be as important, but what good is functionality without people being able to use it easily?

A lot of software clearly has usability issues which, in business, cost money both in terms of time inefficiency and health related problems due to inefficient tools.

Usage-centered design aims to solve these issues by applying a systematic model driven design approach. The usage-centered design approach is driven by three simple and highly related abstract models: the role model, the task model and the content model.

This thesis is an evaluation of the work process of a small IT company called Exor located in Uppsala, Sweden. The criteria for the evaluation are taken from the methods and models of usage-centered design. The evaluation was done to see if, and in that case what Exor could do differently to improve usability in their software.

It was found that Exor possibly could improve the way they incorporate the real users in the design process. Furthermore Exor should allow more time to be spent on constructing models before programming starts. Another big factor is the way Exor uses prototypes, a different approach would be beneficial according to usage-centered design.

Handledare: Jörgen Enström
Ämnesgranskare: Iordanis Kavathatzopoulos
Examinator: Anders Jansson
ISSN: 1401-5749, UPTEC IT 09 006
Tryckt av: ITC

Sammanfattning

Användbarhet av mjukvara är en aspekt som bara blir viktigare och viktigare dessa dagar. Då mjukvara som verktyg blir vanligare är det även viktigt att verktygen är lätta att använda. Användbarhetsproblem kostar pengar i form av tid och hälsoskäl. Om ett verktyg är ineffektivt att använda så går det åt tid att lösa användbarhetsproblemen som uppstår. Om man tänker sig att varje anställd på ett stort företag måste lägga tio minuter per dag på att lösa användbarhetsproblem så inser man att allt sammanlagt blir det en stor mängd tid som går åt, vilket kan översättas i pengar.

Ineffektiva verktyg kostar också pengar i form av att de anställdas hälsa försämras. Ett bra designat verktyg med hög användbarhet motverkar fysiska åkommor såsom musarm, ledvärk och dylikt. Kan man motverka dessa skador sparar man inte bara pengar för företaget utan även för samhället.

Usage-centered design fritt översatt till användningscentrerad design försöker undvika användbarhetsproblem genom att nyttja en systematisk, pragmatisk och modelldriven process. Användningscentrerad design består i huvudsak av tre starkt besläktade modeller. Det är en rollmodell, en uppgiftsmodell och innehållsmodell. Rollmodellen utröner vilken användaren till systemet är och vad dess relation till systemet är. Uppgiftsmodellen täcker vad dessa roller vill uppnå med systemet. Slutligen så täcker innehållsmodellen vilka komponenter användargränssnittet kräver för att kunna stödja de uppgifter användaren vill utföra.

Detta examensarbete är en utvärdering av ett litet IT-företags arbetsprocess. IT-företaget heter Exor och är belägen i Uppsala, Sverige. Exors arbetsprocess utvärderades utefter de metoder och modeller som tas upp i användningscentrerad design. Resultatet av utvärderingen är en summering av vad Exor skulle kunna ändra i sitt sätt att arbeta för att producera mer användbar programvara.

Resultatet av utvärderingen visade vissa brister i Exors arbetsprocess. En tydlig brist var att de riktiga slutanvändarna inte alltid var inblandade i designprocessen, vilket är en viktig del i användningscentrerad design. En annan brist var avsaknaden av modellering i deras arbetsprocess. Om mer tid kunde läggas ner på eftertanke genom modeller innan programmering startar så skulle möjligen goda resultat kunna uppnås. En allvarlig brist i Exors arbete är användandet av prototyper, användningscentrerad design menar att prototyper ska användas på ett helt annat sätt än vad som görs på Exor.

Contents

1 Introduction

1.1 Purpose

These days the usability of software has risen to become one of the most important aspects of a certain system. We can see this by looking at the success of Apple™ in general and the iPod™ and iPhone™ in particular. Apple has been focusing on usability and design. The importance of software usability will grow since more and more laymen will use software in any form in their everyday life. The functionality of the software will still be as important, but what good is functionality without people being able to use it easily?[1]

But there are more important issues concerning usability than designing a user-friendly mp3 player, namely cost and health issues. Some years ago, Cap Gemini did a study of 975 people to see how much time they spent solving computer related problems. The study showed that computer users on average spent two hours and 22 minutes per week with computer problems. That sums up to two and a half week of lost productivity in a year. Then you realize, when put in terms of money, that this is very much a cost issue. A big company with 13000 employees would loose 250000 SEK per day if the employees have to spend just ten minutes a day solving computer related problems.[2]

There is much to gain from improving usability, not just economically but also concerning health. From usability problems follows a whole other range of problems such as: increase in sick leave, the feeling of being inadequate and physical nuisances. By improving the efficiency of the systems users employ, these factors could be reduced which would not only improve overall health for computer users but also save money.[2]

User-centered design of software has been around for a long time and is widely spread in the IT-world. There is an ISO-standard which form the foundation of most user-centered design techniques, namely "ISO 13407 - Human centered design processes for interactive systems". [3]

User-centered design is fore mostly based on three techniques: user studies to map what users want, rapid paper prototyping for fast feedback on user interface design decisions and lastly usability testing to find usability problems in working systems or prototypes.

Usage-centered design (note the difference in perspective) on the other hand is more focused on making usage easier, i.e. the task the user needs done. Where user-centered design incorporates rapid prototyping and testing, usage-centered design is more systematic. It is driven by three simple and highly related abstract models: the role model, the task model and the content model.[4]

The purpose of this thesis is to analyze and map the work processes of a small IT consulting company called Exor located in Uppsala, Sweden. Furthermore

the work processes will be compared and evaluated with respect to the methods and models presented in the book "Software for use - A practical guide to the models and methods of usage-centered design" written by Larry L. Constantine and Lucy A.D. Lockwood. An outcome of the evaluation will be a discussion on what Exor could do differently to make the resulting software more usage-friendly.

1.2 Background

1.2.1 Exor

Exor is a relatively small IT consulting company whose business model is developing IT solutions based on Microsoft™ developed products. Exor is focused on delivering customer accommodated solutions. The systems developed are mostly smaller web-based systems such as web-site booking services, ordering systems and regular web-sites for photographers and such. Bigger projects include a job portal web-site.[5] [11]

They started business some twenty years ago and is run and owned by the founder and chief executive officer Jörgen Enström. Exor currently has eight developers and one marketing director employed. Exor is a certified partner of Microsoft.™ [5]

1.2.2 IT-kompis

Exor's business model is slightly different to what is common in the industry. Exor has come up with a concept that they call "IT-kompis" which is Swedish for "IT friend" which in practice means that the developers at Exor have direct contact with the customer, minimizing misunderstandings and securing safe and sound communication between Exor and the customer. The idea is that the customers get their own friend at the company.[5]

1.2.3 Why usage-centered design?

Putting the user in focus rather than the technology, as user-centered design does, seems like a good incentive. However, really distinct breakthroughs in usability are hard to find. Instead a shift in focus should be made to improve usability. By focusing on what task the user needs to do rather than focusing on the user and user experience will help make the software more usable.

User-centered design is about involving the user in the design process and there is nothing wrong with that. After all, user requirements is the foundation of software development. The problem is that by focusing on what the user wants you might not get the information you need to build a tool that actually supports the user's needs. Users do not always know what they want or what they need.[4]

There are many different techniques within user-centered design but most of them start by gathering user requirements to define scenarios (see Appendix E) that the system could be faced with. After that iterative prototyping of the user interface is done to get feedback from the users.

This could be considered a trial-and-error technique.[4]

As the name suggests, *Usage-centered design,* focuses on the task the user needs done. By investigating the user requirements and trying to find the tasks that need to be supported by the system, you get this shift in focus from user desirables to user needs. After simplifying the requirements as much as possible and finding the users intention with the requirement, complete mapping of the tasks is done. With a model that maps all the user tasks and how they are interconnected, the designer is in a better position to construct a design with features that support the tasks in the appropriate place of the user interface. To find the intention of the user, Constantine and Lockwood propose using a form of use cases called task cases (also called essential use cases) which is more concise than the scenarios and user stories from user-centered design. These task cases are more focused on user intentions and the system's responsibility towards that intention. The format is very much to-the-point which facilitates compact and comprehensive modeling of the task essentials.[7]

Where user-centered design would start prototyping a user interface intended to resemble an actual user interface based on the scenarios or user stories, usage-centered design would keep it as abstract as possible. For as long as possible. Keeping the models abstract and not resembling an actual user interface too much has the advantage of allowing designers to not get preoccupied with details of the user interface too early in the design process. This way designers can stay focused on the essentials, the user's intention. With a good model of the tasks it is easy to derive abstract prototypes for the user interface.[6]

These are the basic reasons to choose a usage-centered design process. The models and methods of usage-centered design will be more thoroughly explained in section 2.

1.3 Problem Formulation

The general problem of this thesis is to evaluate the work processes of an IT company. More specific, the problem lies in evaluating IT consulting company Exor's processes with respect to models and methods from usage-centered design described in the book "Software for use - A practical guide to the models and methods of usage-centered design" written by Larry L. Constantine and Lucy A.D. Lockwood. These models and methods strive to make the end-resulting software as usage-focused as possible. The goal of this thesis is to see whether or not Exor could work more usage-centered and if so, to see what Exor could do differently to this end.

The problem is two-folded, one part of the problem is to investigate and map Exor's work processes. The other part is to analyze and evaluate the processes compared to the methods described in the book mentioned above. The differences

between the processes will be pointed out along with a discussion on what Exor could do to work more usage-centered.

1.4 Delimitation

The evaluation will cover the work processes, not the end result i.e. the software delivered to customer. Furthermore, the discussion will provide suggestions on what Exor could do differently in their processes, but not explicitly say how these changes could be applied in Exor's particular case.

2 Methods and models of usage-centered design

Usage-centered design is about focusing the design process on the task that the user needs to accomplish. It is a model driven approach where the models used in the design process are meant to be as abstract and stripped of details as possible in the early stages of the design process. The rationale for this is to stay focused on satisfying the user's needs, and not stray away thinking about visual gadgets and widgets too early in the process as this hinder designers from designing software that is centered on the tasks that need to be accomplished. The key elements of usage-centered design are:[7]

- Pragmatic design guidelines
- Model driven design process
- Organized development activities
- Iterative improvement
- Measures of quality

A design fueled towards usability should be derived from acknowledged usability rules and user interface principles. Constantine and Lockwood has devised a set of usability rules closely related to Jakob Nielsen's five rules of usability:[8]

- Learnability
- Memorability
- Efficiency of use
- Few and non-catastrophic errors
- Subjective satisfaction

The rules that Constantine and Lockwood devised are meant to guide the designer to build systems that are usable without help or instructions, by a user that is experienced within the application domain but has no prior experience with the system. Furthermore the rules are intended to help design systems that are efficient to use. Another important issue is that the system should support progressive usage i.e. as the user's skill advances, the system should be adjustable to fit a more experienced user and not hinder the user. Also, the rules are there to ensure that designers build systems that support the user, in the way that they make things easier, simpler and faster for the user. Supporting the ability to revert or undo a certain step is an important feature. Finally, supporting Constantine and Lockwood's rule of context will ensure that the system is suited to the real working conditions within which it will be deployed.[7]

As said, usage-centered design is model driven. We will briefly go through the different methods and models of the design process. Figure 1 shows an overview of the usage-centered design process.

2.1 Role model

An important step in usage-centered design is to sit down with users and have a thorough discussion on the user's needs. This is preferably done early in the process. The goal of this discussion is to model the

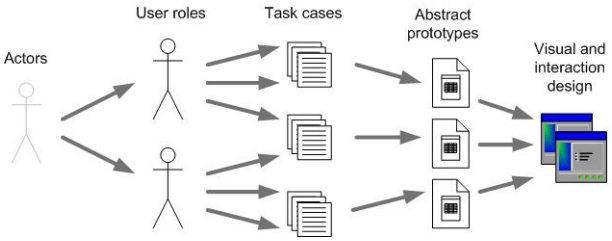

Figure 1: **Overview of the usage-centered design process**

information we get from the users. The outline of the role modeling process is simple: gather information, build model, get feedback on model from users and then refine it.[7]

Users are very important to designers and developers because of the role the play in relation to the system. It is therefor meaningful to collect the needs, expectations, behaviors and responsibilities of users and try to abstract them into user roles. To define these roles it is necessary to ask questions of ourselves or of the users. Typical questions to ask are:

- Who would or could use the system?
- What is the general class or group to which they belong?
- What distinguishes how they would or could use the system?
- What characterizes their relationship to the software?
- What do they typically need from the software?
- How do they behave in relation to the software, and how do they expect the software to behave?

By answering these questions a good understanding of the different roles can be achieved and the modeling can begin. Typically a list of roles will be defined, these are called candidate roles. The candidate roles should be sorted by similarity into groups. Simplification and generalization of roles should be sought after during the whole process. The next step is to find a focal role or focal roles if there are more than one. A focal role is the one that is most common or most important.[7]

2.1.1 User role map

Once the candidate roles and focal role has been defined they should be described and simplified sufficiently.After that the roles are to be mapped by noting the relationships between them. To do that a user role map is created, which will provide a broad vision of how the roles fit together with the system. The roles can be related to each other in different ways, namely: by affinity, by classification, or by composition. If they are related by affinity it means they are similar in some way, if they are related by classification it means that one role is a subtype of another role and if they are related by composition, two or more roles combine certain characteristics and one role is then, in a sense, composed out of the other role. A general user role map can be seen in figure 2.[7]

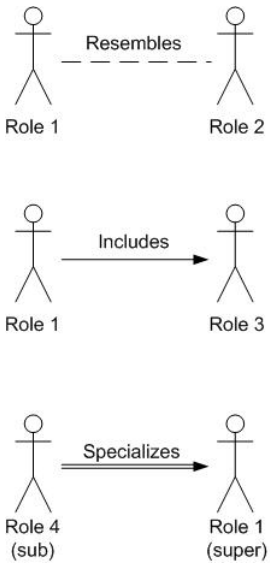

Figure 2: **User role map**

2.1.2 Information sources

To define the user roles we obviously need to involve the users. In usage-centered design it is important to involve the *real* users, i.e the people that will end up using the system as a tool. The end users should be the primary source of information. However, the reality is that often it is not possible to get information from end users. Then other sources of information can be useful. One of them is the customers, the people who buy software but may not necessarily use it. Another one is the managers of the end users, they often have a strong opinion and cannot be ignored. Other sources include: experts on the specific field, and work manuals.[7]

2.2 Task model

The next model in the usage-centered design process is the task model. It comprises two models: the task cases and the task case map. The goal is to model the work done by users when they use the system. There are many ways to model the work such as: using a workflow diagram, writing scenarios to describe the work, writing use cases, or doing a breakdown chart of the work. Constantine and Lockwood opted to use a variant of the use case, which was first described by Ivar Jacobson in his object-oriented engineering method.(See appendix A) Described briefly, a use case is a simplified description of the interactions between the user and the system, showing the user action and the system's response to that action.[7]

Constantine and Lockwood took the simplification of use cases even further because they felt the use case in its original form made to many implicit assumptions about the user interface, which is yet to be designed at this stage of the process. They came up with the essential use case, also called task case, which will be described in more detail in the next section.[7]

2.2.1 Task cases

A task case is a structured description of a task or interaction between a user and a system. The task case case should be written in the language of the user, furthermore it should be simplified, generalized , and technology-free. Another important thing is that no assumptions

are made concerning the implementation of the interaction.[7]

When devising the tack case concept, Constantine and Lockwood made the choice of focusing on the user intention and the system's responsibility towards that intention, rather than focusing on user action and system response, as the original use case did. The reason for this is to achieve a change in perspective to make the task cases better explain *why* the user takes a certain action. This way of doing use cases makes them shorter and simpler because it only includes those steps that are essential to the user. The name of the task case should express the user intention or the purpose of it.[7]

The task cases are derived from the user roles in the role model defined earlier in the usage-centered design process. Starting with the focal user roles and then moving on to the candidate user roles, a brainstorming session will give us a list of candidate task cases. This list should be reviewed and refined to find task cases that are similar, related or composed of each other and thus might be grouped together, just as the user roles in the role model.[7]

Once a refined list of candidate task cases is produced, the next step is to find one or more focal task cases. The focal task case is the most common one and based on which it would be reasonable to organize the user interface. For the most part, the focal task cases are based on the focal user roles. When writing the task cases it is important to identify the essential user purpose. It is desirable to involve the users in the task case brainstorming session or at least have users review the list of task cases to confirm that it represents the user needs. Since the task cases are written in user language and are fairly intuitive it is easy for the users to understand them.[7]

2.2.2 Task case map

The next step is to map the task cases into a model to get a good overview. Depending on the size of the system it may involve tens or hundreds of task cases and as stated previously, task cases are often related, similar or composed of each other. therefor it is favorable to identify how the task cases are interrelated. Once all possible relationships has been found a task case map is produced. A general depiction of what a task case map could look like can be seen in figure 3. Note that the figure shows close affinity among the focal task case and the first candidate task case, hence they are overlapping and grouped together. This is to show that the solution to this design problem should be grouped together later on in user interface design.[7]

2.3 Content model

A user interface should be organized into different and distinct interaction spaces. These interaction spaces should provide the user with the appropriate tools and materials for the task at hand. Thus we need to define different interaction spaces based on the task case map and populate these spaces with contents to support all

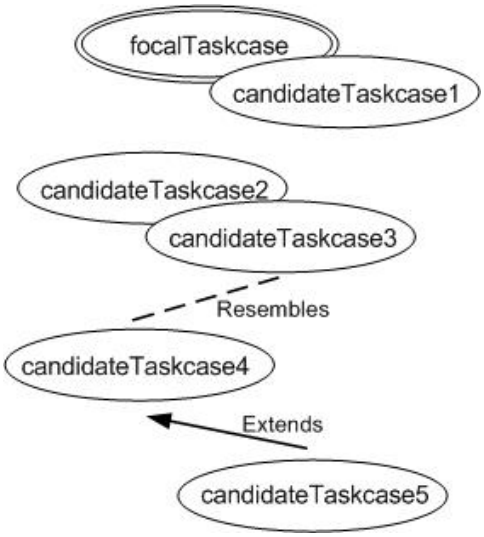

Figure 3: **Task case map**

the different task cases. The content model is an abstract representation of these interaction spaces. One interaction space can support several task cases and on the other end, some complex task cases might require switching from one interaction space (or context) to another to be able to complete the task. It is desirable to keep context switching as low as possible since this is not very usage-friendly.[7]

A widely used method, not only in usage-centered design, to model contents is to use regular paper and Post-it notes. Have separate sheets of paper represent each interaction space and Post-it notes in various colors represent the tools and materials. The names of the interaction spaces should be meaningful and not be to generic. Once the different interaction spaces have been defined they are to be populated by the necessary contents. This is done by posting notes in each of them until all task cases are supported.[7]

When done, the result is a very abstract, low-tech, and passive prototype which can be shown to the users for feedback. The benefit of using paper and Post-it notes is that we can show a very brief layout of the functionality to the users. At the same time the model clearly signals that this by no means is a design written in stone, so the users will not feel as they cannot influence the design. Another advantage of using Post-it notes is that, in case big reorganizations are needed, the process of doing this is not more complex than rearranging the Post-it notes. The ease of reorganizing encourages experimentation and exploration.[7]

2.3.1 Context navigation map

Using sheets of paper and Post-its is a good way to solve design problems but it is impractical for documentation, if that is needed. Therefor, to document the content model, it is better to use a textual form. In this form, the interaction spaces are simply listed along with their contents.[7]

Big systems will be comprised of several interaction spaces and switching between them is inevitable. Hence it is worthwhile to map all the interaction spaces clearly showing the context switching within the system. This is called the context navigation map. The advantage of doing this map is that it can reveal different kinds of problems in how tasks

are distributed over interaction spaces. It will also show how complex the system will be for users. If a task requires a lot of switching between interaction spaces it could possibly be simplified by including functionality from one interaction space into another. Figure 4 shows a general depiction of a context navigation map. The double arrow represents an implied return and the single arrow represents an action. In a more specific navigation map the interaction spaces would naturally have more meaningful names than 'interactionSpace1' etc. The task cases inside the interaction spaces represent the contents needed to support them. As you can see, the task cases with close affinity are covered in the same interaction context.[7]

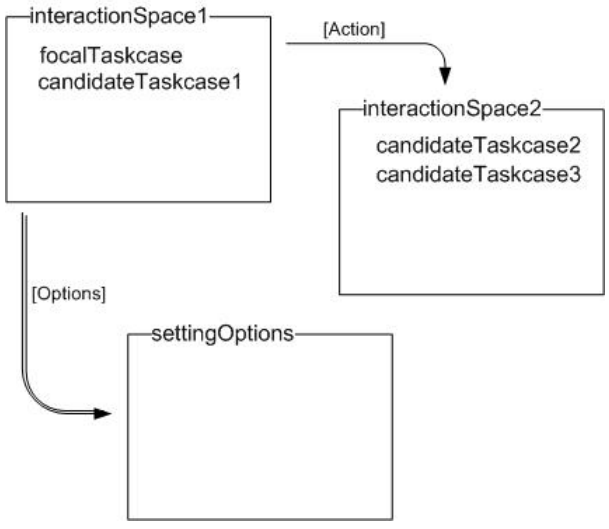

Figure 4: **Context navigation map**

2.4 Model verification

When the role model, the task model and the content model are done, it is evident that they are all closely related. The strong relationships between them make it possible to validate them against one another. This is done by checking the models against a number of consistency criteria. These criteria are:

- Every user role must be supported by one or more task cases.
- Every task case must support one or more user roles.
- Every task case must be fully supported within the content model.
- Every interaction space must be used in some one or more task cases.
- All tools and materials must support some one or more task cases.
- Every switch between interaction spaces should be used in enacting some task case.

By verifying, the models are cleaned up. Roles may appear that are not linked to any task case, it may be that the role was incorrectly identified in the first place or that task cases are missing in the task model. Also, task cases may appear that are not associated with any user role. Such task cases should be reviewed carefully to justify their existence. Task cases that require multiple interaction spaces should be examined to see if they can be simplified in any way, this is especially important for the focal task cases.[7]

2.5 Implementation model

The implementation model deals with, as the name suggests, the implementation. In this stage of the process, prototypes are made which we will go through in the next section. Implementation modeling is about translating the content model into an implementation model, which could be a complex process. It is possible to start programming once the content model is finished, but Constantine and Lockwood's experience is that if the programming is guided by an implementation model, better results are obtained.[7]

In order to complete the implementation model, issues within three different classes need to be resolved. These classes are: the interaction spaces, the components within the interaction spaces, and the composition and layout of the components in the interaction spaces. At this stage of the design process there are many design decisions needed to be made.[7]

The first thing that needs to be resolved is the design of the focal interaction space, which supports the focal task cases and possibly other task cases as well. The focal interaction space will be the 'main screen' that users see. Decisions have to be made about what type of user interface architecture to use, this could, for example, be whether to use a different window for each interaction space or to use a tabbed layout where the different interaction spaces have their own tab. An important goal to strive for when designing is to keep context switching as low as possible, i.e. do not make the user unnecessarily switch between interaction spaces. For instance, an action takes the user to a dialogue box and from there to another one and so on. Eventually the user interface will be all cluttered up with windows and boxes.[7]

In the first draft of the implementation model it is favorable to solve the design issues with fairly standard and conventional solutions. The model should then be reviewed to see if improvements can be made, which is often the case. At this point, more innovative design solutions may be more advantageous. Once all interaction spaces have been populated with appropriate tools and materials to support the task cases a prototype has been produced.[7]

2.5.1 Prototypes

There are several different kinds of prototypes. They could be either active or passive, and high-fidelity or low-fidelity. An active prototype is one that actually functions in some way, either simulated or real functionality. A passive prototype on the other hand, may be just a sketch on a paper or like the sheet of paper and Post-its used in the content modeling. A high-fidelity prototype closely resembles the real user interface, whereas a low-fidelity prototype barely resembles the real user interface.[7]

The benefits of prototypes are that they are easier and quicker to do than programming the user interface. Also, they are a great way to get feedback from the users on

design issues. It is better to use passive, low-fidelity prototypes in the early stages of the design process, so that users do not think that the design decisions have already been made and cannot be changed. Later on in the design process the active and high-fidelity prototypes are more appropriate. The prototypes should be reviewed together with the users and then refined based on the feedback. Though, iteratively changing the prototypes should not be overused, since users could always find something new they want to improve and often it is hard to know whether a change has actually made the design better or if it has merely changed it.[7]

2.6 Programming

In the usage-centered design process Constantine and Lockwood propose to use an object-oriented (See appendix F) programming technique. The most striking benefit of using object-oriented programming when developing a user interface is the object-oriented techniques ability to reuse components. This is good since consistency is important to improve usability. Reusing already familiar, to the user, components will allow the user to more easily understand the user interface.[7]

Also Constantine and Lockwood propose programming iteratively, starting with the focal task cases and continuing with the other prioritized task cases if a priority list task cases has been done.[7]

2.7 Help

An important factor when developing software is including support to help the user. In order to create usable software help should be provided. In perfectly usable software help should not be needed, however this is rarely the case. Helping the user means not only to include a help section but also to design instructive interfaces, to design for error prevention and to provide active on-line help among other factors.[7]

Designing instructive interfaces to help the user is the most cost-efficient way to provide a good help service. The meaning of an instructive interface is that the user interface should be designed in such a way that there should be no doubts to what a function does. Icons used should be well-known standard ones, or at least icons that leave little up for interpretation. Instructive interfaces should provide active help by displaying tips or identification pop-ups on a mouse-over event. Hints and tips can also be communicated to the user using a status bar or a separate window.[7]

Help can be classified into the following categories:

- **Seeking identification** - To find out what something is.
- **Seeking instruction** - To find out how to do something
- **Seeking suggestion** - To get a suggestion on how to resolve something

- **Seeking clarification** - To get an explanation for something.
- **Seeking elaboration** - To get more information about something.
- **Seeking reminder** - To get a reminder about something.
- **Seeking location** - To find the location for something.
- **Seeking features** - To get information about the features of the system.

How to resolve these help categories is best understood by defining use cases (or help cases) for each of them.[7]

Rather than displaying an error message when something has gone wrong, a better approach is to try to prevent errors before they are made. This is most applicable when the user is supposed to input something to the system. For example, if the user needs to input a date it is advisable to construct the user interface to only accept valid dates. This could be done by letting the user select the date from a calendar or separating the input fields for year, month and day. Further, when the user is supposed to select something out of a set, it is favorable to use a drop-down list of some kind.[7]

Documenting all features of the software is recommended. The documentation and the help section should be written by technical writers, not programmers. The text should be written in user language. Also it should be friendly and to the point.[7]

3 Methodology

To be able to solve the problem of evaluating Exor's work processes a set of different methods had to be used. The methodology used in making this thesis can be classified into four different categories. The initial part was a study of the topic in which the thesis would take place. Next part consisted of gathering information about Exor's work processes. Once that was done the information obtained needed to be analyzed and finally evaluated. All methods will be described in more detailed in the following sections.

3.1 Study of topic

The first phase in the process of doing this thesis was a thorough study of the methods and models described in Constantine and Lockwood's book 'Software for use - A practical guide to the models and methods of usage-centered design'. The study was invasive to really understand the whole process and the factors involved:

- The actors.
- The rules and principles of usability.
- The models.
- The logical structure of the process.
- The rationale for all steps and models involved.

Another part of the initial phase in the making of this thesis was investigating more recent research results. Several research articles made by the authors of the book were examined. Also, research and studies made in related fields such as user-centered design were reviewed in order to see what other techniques were doing differently. The initial study of literature and research results was made to get a good understanding of the field in order to be able to analyze Exor's work model properly.

To understand the importance of improving usability and to get solid background knowledge about the topic, articles and papers about the cost and downsides of usability problems were reviewed. This was done to understand why this topic is so important and will continue to be in the future.

3.2 Information gathering

The information gathering process involved using qualitative interviews and a variant of focus groups. The management were interviewed and the developers were divided into focus groups.

I chose qualitative interviews because I wanted to have more of an open conversation rather than having the interview subjects fill a questionnaire, as is common with quantitative methods. The questions asked were a combination of open and close ended. A majority of the questions were open ended because I wanted to get an unrestrained response from the interview subjects. The close ended questions were accompanied by open ended follow-up questions in case of an affirmative

answer. The interviews were recorded in order to get a smooth conversation without interruptions and pauses for notation.

As said, the developers/designers were divided into three different focus groups. The division was made based on the time they had been working at Exor. The groups were, developers employed since:

- 5-6 years.
- 1-2.5 years.
- less than 0.5 years.

The reason they were divided this way is because the management at Exor thought it would be interesting to see if there were any differences between the groups and to see if they were successful in introducing new developers. The rationale behind the decision to have the developers divided into focus groups in the first place was that we thought it would promote an interesting conversation among the group members. Thus, when hearing how others work, making them really think about how they work individually. The conversation in the focus groups was at all times fairly informal and relaxed, all to make the developers not feel awkward. The focus group sessions were recorded as well.

The type of questions asked differed between the management and the developers. The management was asked questions concerning the overall process of the projects as well as questions about the initial phase of projects and how they start up. The questions to the developers were more focused on the actual development and how it was organized.

Part of the information gathering process was also a small course that Exor's founder has for all newly employed. The course explains the parts of the project process and gives advice on how to treat customers.

3.3 Analyzing

Once all interviews and focus group sessions were made the information had to be thoroughly analyzed. The material gathered was reviewed and all the involving stages of Exor's work process were put into a logical order to make the material more structured. This had to be done to be able to evaluate the material with respect to the evaluation criteria; the methods and models of usage-centered design, which is a very structured model driven approach. When an understanding of the logical order of all stages involved in the process, the activities within each stage were analyzed to understand the goals and the reasons for the actions performed.

3.4 Evaluating

The results from the analysis were then compared to the activities in the usage-centered design process described in section 2. This was done to identify similarities and uncover differences. Each stage of Exor's work process was thoroughly evaluated with respect to the corresponding stage in the usage-centered design process.

All activities in both processes were compared to find differences in order to come to a conclusion. The evaluation resulted in a list of activities that could be done differently to comply with the methods of the studied literature.

4 Results

Since the problem in this thesis was to investigate, analyze and evaluate the work processes of Exor, I wanted to get a good overview of the whole project process. Because of that I wanted to understand the underlying reasoning behind Exor's concept IT-kompis. The next section will cover Exor's model behind the perceived quality of service and involving factors to be able to deliver good quality of service. After that the results from investigating the IT-kompis concept will be covered.

The following sections after that cover the work process, from start-up to the end of a project.

4.1 Quality of Service

The founder of Exor, Jörgen Enström realized a long time ago that the perceived quality of service by the customer not only depends on the delivered product. Since almost all IT companies can or will deliver equivalent solutions to the customer's need it is important to stand out in other ways. To make the customer perceive a better quality of service other factors should be dealt with. In Exor's work they always have this list of guidelines, also called quality determinants, in mind to give the customer the best perceived quality of service as possible:

- **Reliability** - to function reliably and as a complete unit.
- **Responsiveness** - the Exor staff's will and ability to serve.
- **Competence** - the appropriate competence required of Exor staff.
- **Availability** - the ease of which to reach Exor.
- **Treatment** - the politeness, respect, consideration and kindness of Exor staff
- **Communication** - the ability to listen to the customer and giving information in a language he/she understands.
- **Trustiness** - the reliability and honesty towards the customer and to always do what is best for the customer.
- **Security** - the absence of danger, risk and doubts.
- **Understanding** - the effort to try to understand the customer's needs and wishes.
- **Concreteness** - the physical aspects such as the locality and proper clothing.

These guidelines serve to help Exor staff give the customer added value to the product delivered. They permeate Exor's work in all aspects to try to satisfy the customer. The underlying model for perceived quality of service can be seen in figure 5. [9]

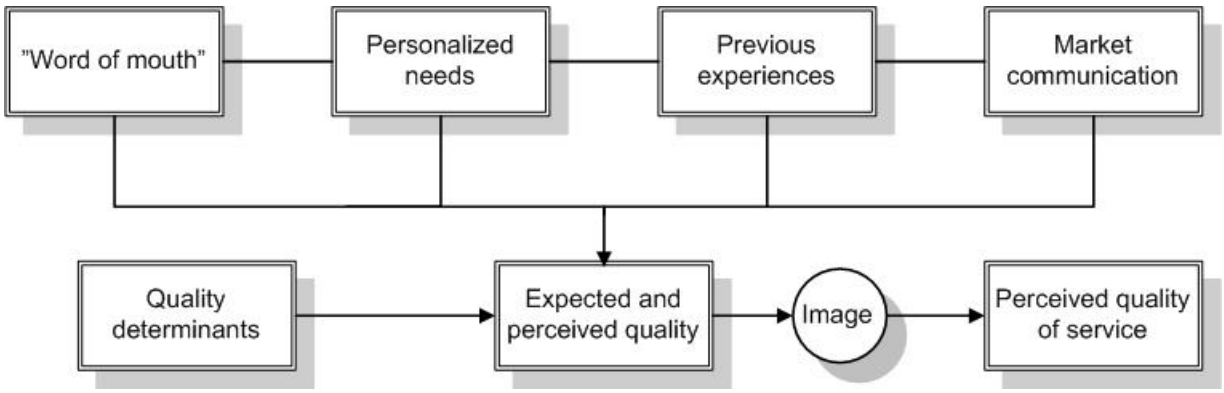

Figure 5: Model of perceived quality of service

4.2 IT-kompis

The concept of IT-kompis means that the customer is supposed to get his very own friend at Exor. At Exor all developers are also project managers. This means that Exor does not work traditionally with a project manager and a project team assigned to him/her. Since most projects at Exor are fairly small (one man projects, 1-2 months) this model is more suitable.[10]

When a new customer arrives he/she is assigned a developer/project manager at Exor. This will be the customer's IT-kompis which will handle most of the communication and also develop the system desired. The developer should be in direct contact with the client.[10]

The name IT-kompis was coined in 2007. The underlying model however, is one that Exor has used since the start, about 20 years ago. The founder of Exor, Jörgen Enström experienced that in the business it is common that there is a certain project manager that handles all communication with the customer. The customer is never in contact with the developers that has the real technical knowledge. The project manager often takes the role of an intermediary. Jörgen Enström felt that, this way, a lot of information gets lost in translation. Thus by skipping the middle-man, it would allow more direct communication, minimizing misunderstandings and enabling the developers to explain problems instantly. The communication is then speeded up and it makes it more clear what the customer and Exor has agreed upon. [11]

The idea of IT-kompis is that the customer is always just a phone call, or an e-mail, away from the person responsible for developing the system. The developers should try to have close and frequent contact with the client so that he/she feels involved in the process and gets updated on the progress or to be able to help solve design issues. The communication range from progress of on-going projects to support for already delivered systems.[12]

4.3 Start-up

New projects are mainly brought in to Exor by Jörgen Enström. When an inquiry for a new system arrives at Exor, the first phase is evaluating whether or not it lies within Exor's segment of expertise, it should be possible to develop the system with the technology that Exor applies. Also, since Exor is focused on delivering customer accommodated solutions, there are some projects that are not suitable such as systems to which there already exist a solution. Once it has been established that the system requires an accommodated solution in Exor's field of technology, an initial meeting is set up.[11]

The next step is a meeting to try to find out what kind of system the customer needs. This meeting typically involves Jörgen Enström and one or more representatives from the client side. When investigating what kind of system that is needed the focus is more on finding the purpose of the system rather than just asking what the customer wants. A part of finding the purpose is simply asking why they want to build a new system or changing from a present solution. At this stage of the project the user is typically not involved, instead the contact is between Exor and the buyer at the client side. At this meeting the economy of the project is also discussed to determine if the project is economically feasible. If it is, the project gets a green light and can move on the next phase.[11]

At this point a second meeting is set up involving one or more developers/project managers from Exor and representatives from the client side including, if possible, actual users of the system to be developed. The first activity in this meeting is mapping the client's business processes that are relevant to the system that is desired. This is done to get an understanding of the problem at hand. Once that is done, there is an open discussion to determine what functionality is needed to deliver a solution.[11]

The results of the discussion is documented into what can be considered a requirements specification, although not very technical. The document typically contains a brief summary of what functionality is required from the system. The document is later presented to the customer so that he/she can confirm that he/she has understood it, thus making sure there are no misunderstandings. Exor does not work with big, invasive and detailed requirements specifications, instead the developer use this brief summary as a specification for the programming. The reason for doing the requirements document is mostly to establish a correct understanding of what type of system that is required.[11]

The amount of documentation before development starts depends on the size of the project. In small projects (200 hours), there is almost no documentation at all, whereas in big projects there has to be more documentation and a requirements specification that is reviewed and refined several times since the problem is much

more complex. Smaller projects are often handled by one developer and bigger ones could involve several developers.[9]

4.4 Development

Once the project has started up and is under way the development begins. Development differs a lot depending on the size of the project. Almost all systems are web-based in the sense that they utilize a web user interface. The fact that the systems are web-based does not necessarily mean that they are open to the public, rather they can be installed to the intranet of the client. The majority of the projects are smaller solutions but there are also bigger projects that can spread over a year or so.[12]

The smaller projects are often one-man projects and in those projects programming starts immediately once the brief requirements document has been agreed upon in the start-up phase. At Exor, modeling and designing is by default low prioritized activities and in the small projects it is often not justifiable, economically nor by the amount of time it consumes. Hence, doing a thorough investigation to map the actual users and describing them with personas (See appendix A) or user roles is not done. Use cases are not utilized explicitly in the smaller projects since the project is typically to small. Rather, use cases are used implicitly by the developer for himself. They are not written down formally but the notion of them is always present.[13]

The technique used at Exor for the smaller projects can be described as code-and-fix. Write code based on the requirements agreed upon earlier and iteratively refine it. The goal is to construct something to show to the customer fairly quickly. Once that is accomplished, it is showed to the customer to get feedback. This can be considered as doing a prototype and then put it up for review. The customer is involved during the whole project by keeping close contact. Since almost all solutions are web-based it is very easy and convenient to just put up a semi-active prototype online for the user to test and give feedback on. Then the prototype is refined based on the feedback. Passive prototypes are also used at times. Whether or not to use an active or a passive prototype is a decision left to be determined by the developer.[14]

The techniques for designing the graphical user interface varies a great deal. First of all it depends on the customers, sometimes they have a very good understanding of what they prefer whereas sometimes they do not have any preferences at all. At times, the customer already has seen a working solution to the problem and wishes something like it, then that could be used as an inspiration for the design.[13]

During the whole process of designing the user interface the customer is frequently involved. The developers are always perceptive to the wishes of the customer. If the customer does not have an idea of a good design, the developer makes suggestions. The design decisions are always based on client wishes. For example choosing between a tabbed layout

or using several windows, is a decision the developer will ask the customer about.[12]

The developers try to design instructive interfaces when possible. Interactive help is provided using pop-up tips and identification names. Should something require a more detailed explanation a frequently used technique is to link to an Adobe™ reader document containing help for the particular problem. Sometimes this document is made by the customer. The customer reviews the system to find obscurities and then document the problem and the solution or procedure to do a specific action. A link to the document is then included in the software by the developer.[13]

For small projects there is minimal documentation and these systems are typically sufficiently intuitive to not need a detailed help section or documentation. Should the developers discover that the user has a problem understanding how to use the system, they can just modify the part that causes confusion. It could be simplifying the interface, adding explanatory text or changing a function name. All web-based solutions are run and maintained at Exor's location, an advantage of this is that should it be necessary to adjust something or fix bugs, it can be done quickly without sending a developer to the client site or build a new release.[13]

Documentation is by default minimal since this is an activity that neither the customer nor Exor wants to pay for if it is not necessary. However some customers require detailed documentation. It all depends on the customer, if they are willing to pay for the documentation, it is done. Generally, if the customer has its own IT-department they often require more thorough documentation.[13]

The developers try to prevent errors as much as possible. For example when the user wants to set a date in some system, both validation of the input and an explanatory text on how to organize the input is used. Another technique used in newer systems to prevent input errors, for the date example, is using a calendar tool where the user simply clicks the date which he/she wishes to input. Also, for selecting from a collection, drop-down lists are often used.[13]

To support the ability to revert a step or undo an action is a somewhat complicated matter when it comes to web-based systems because it is time consuming to implement, thus in the end costly. Whether it is implemented or not is up to the customer if he/she desires it. In general it is more common to not include an easy way to undo an action than to include it. Instead a technique used, to not put the user in these awkward moments when he/she has made an undesired change, is to make the user confirm the action before it is performed by the system. This is used when there is a major change about to be made. Usually a pop-up window is used to confirm the action.[13]

4.4.1 Development methods

The development method used at Exor can best be described as an agile development method. (See appendix B) Coding starts very early in the project to quickly build something useful to show to the customer for feedback. Based on the feedback, improvements can be made. Smaller projects typically involves just one developer whereas bigger projects could involve 2-4 developers. When just one developer is involved it is not really fair to say that a strict development method is used, it is very much up to the developer himself. There are no real guidelines on this topic at Exor but rather ad hoc methods that suit the particular developer are practiced.[12]

When doing bigger projects involving several developers it is advantageous to use a more strict development method to better organize the work. In a recent bigger project, Scrum development (See appendix C) is used.[12]

When coding the most used technique at Exor is object-oriented programming. Since most systems are developed using Microsoft's™ .NET Framework this technique is favorable because the .NET Framework clearly supports object-orientation.[13]

4.5 Closure

Once a system is starting to get finished and the project is reaching its end, acceptance testing is done. Many times there is a demo version online that the customers can test by themselves, this can usually also be tested during the course of the project. If the developer wants feedback on a specific feature he can ask the customer to try it and say what he/she thinks of it. Doing this during the project can resolve a lot of issues before they get to complicated to fix. A demo version put up during the project is considered more of a prototype, thus it does not need to be fully functional nor be visually appealing. When a demo version of a system that is to be released is put up online for testing, it is supposed to support all features agreed upon earlier.[13]

Though, putting up a demo version is not always done, it is up to the customer or the developer which he/she prefers. Sometimes the project is too big to make it useful putting up a demo version, because perhaps different parts of the system that are depending on each other are developed separately and at different times.[13]

Usually all the developers continuously test everything developed and when the system is considered finished and tested, a meeting with the customer is scheduled to do a walk-through of the system and how it works. In that meeting all aspects of the system are covered to clarify obscurities and explain functionality. The reason for this is to check that all requirements that were agreed upon are supported. If the customer has any new additional requirements at this point, this is considered a new project with new requirements and project plan.[13]

4.6 Usability

In their work the developers try to think about usability. They always consider the most usage friendly alternatives to solve a problem. Though all the time the developers has to consider the amount of time a particular solution takes to implement since this is very much a cost issue. If there is not enough time to implement an extra clever solution it will be discarded in favor of another more easily implemented one. With that said, this does not mean it is a bad solution. Of course, should the customer express a wish for a system fueled towards usability, it will be delivered but it may take longer to develop thus costing more in the end. [14] [13] [12]

Generally the developers try to consider what the user's perspective might be and try to see things in different views to discover usability problems. If a developer implements a new solution not used before, often he consults other developers to get a second opinion about it. These consultations are not carried out formally, rather they could be done over a coffee break. Sometimes the developers consult marketing director Ellinor Källholm, who does not have a technical background, to uncover potential usability problems. She usually thinks in a different way than the developers, thus allowing the solution to be reviewed by a different mind set.[10] [12]

Usable and functional solutions are always favored over fancy and visually appealing ones. The Exor guidelines for development states that functionality should be prioritized over design. When all required functionality has been implemented satisfactory the developers can start fine-tuning the design to make it more visually appealing. This does not apply to regular informative web-sites such as web-sites for photographers or just informative sites about a particular company. In the development of these web-sites a visually attractive design can be prioritized since there is no real functionality required to be implemented.[9] [12]

5 Discussion

From the results we can see that Exor's work process differs a great deal from Constantine and Lockwood's suggested design process. In this section we will go through the steps in the usage-centered design process to clarify what Exor does differently.

5.1 Requirements gathering

The requirements for the system is the foundation on which the system relies. It is important that the information gathered is the correct one. Should the system be based on incorrect requirements the results could be devastating.

An important factor in the gathering of requirements of the usage-centered design process is having the right focus. When investigating the requirements, the focus should be on the user's needs rather than on what he/she wants. As we can see from the results Exor has this focus. They try to uncover the real needs instead of just accepting a wish list from the customers. The reason for the initial meeting between Exor and the customer is to find the purpose of the system, which clearly is in compliance with the usage-centered design process. The questions asked to the customer may not be precisely the ones that are presented in section 2.1 but the important thing is that Exor has the right incentive when uncovering the requirements.

Another important thing is to understand the application domain in which the system is meant to function. An initial stage in Exor's work process is mapping the customer's business processes to get a clear understanding of the domain in which the system will be deployed. This activity follows the suggested methods by Constantine and Lockwood.

Involving the real intended users of the system is fundamental in usage-centered design. The real users are the most important information source. Exor's gathering of information is somewhat lacking in this area. The information is for the most part gathered from client representatives which may or may not be actual users of the system. Actual users are involved in the requirements gathering only if possible. However, the systems developed by Exor are generally quite small and from that follows that they are not the most complex systems so one could argue the need for real users. Is it economically justifiable to do a thorough investigation of actual users?

It has become clear that this is very much a cost issue. Often the customers are not willing to pay for the extra effort a thorough investigation requires. Since the systems are quite small they are usually fairly intuitive anyway, hence gathering requirements from the client representative could suffice. It would still be accurate in supporting the needs of the real users.

At Exor, the documentation of requirements is an activity that is not assigned a lot of time. Exor clearly strive to start building as quick as possible. The reason

for this is that it has become clear to Exor that spending time on documentation is something the customers generally are not willing to pay for. Doing a formal requirements specification takes time and the customers generally want to see results fairly quickly to be satisfied, so this is a predicament. Again, remember that often the systems are not that complex, hence creating a formal requirements document may not be needed. It could suffice to just have the brief requirements summary that Exor use as foundation for every project. However, in bigger projects involving several programmers Exor do more extensive documentation since it is hard to organize the development without doing a more formal specification.

To be able to do the models according to the usage-centered design process it is crucial to collect the needs, expectations, behavior and responsibilities. This information is the foundation for creating the user role model which is the next step in the process. In this sense Exor's work process differs a great deal from the usage-centered design process.

As stated in section 2.1, getting feedback on the collected information is important. This is something that Exor is very strict about. They always want the customer to confirm that the information gathered is correct. This is done to be sure that there are no misunderstandings concerning the system requirements. It is crucial that the system is built according to the right requirements. Also, by having the customer confirm that the requirements are satisfactory, possible future contractual disagreements can be avoided.

5.2 Modeling

Once the correct information is gathered and requirements have been defined, the next step is the modeling activities. Since usage-centered design is a model driven approach it is obvious that these activities are fundamental.

The first model to create is the user role model which is based on the requirements gathered in the initial phase of the process. According to the usage-centered design process the requirements should be abstracted into user roles. Since Exor generally utilize agile development methods, what the developers do when the requirements have been agreed upon is that they, for smaller projects, start coding immediately. No significant work is put in developing personas or roles to map the actual users of the system. Doing this would be good to understand the needs of the users. Also, defining user roles is good to be able to do accurate use cases. Since the developers at Exor start programming right away this step is skipped, thus the work processes differ in this aspect. The underlying explanation for this is, again, the cost issue and that it might not be needed for the kind of projects that Exor deal with.

In the bigger projects however more work is put in doing a thorough investigation and mapping of the users since the bigger projects generally involve more

complex systems. The conclusion is that for the most part the systems are not big enough to justify doing the role modeling, except for the more complex systems.

The role model consists of two parts, the user roles and the user role map. Since Exor do not define user roles, subsequently they do not map the relationships between the roles. Not even for more complex systems. For the less complex systems this is understandable, there would not be that much of a difference between the roles hence mapping the relationships between them would possibly be unnecessary. It would, however, be a good idea for the more complex systems. The roles would differ more for these systems and a consequence of that would be the need to map the candidate roles and the focal role(s) by affinity, composition and classification. The creating of the user role map could potentially provide a better understanding of the complexity of the system. The user role map is the foundation for the next step in the usage-centered design process, the task modeling.

The task modeling is done to better understand the systems responsibility towards the user. In this modeling activity task cases are defined and mapped to find relations between them. At Exor, task cases or use cases are not formally defined for small projects. The systems in these projects are simple enough to be developed without use cases. However, as stated in section 4.4 the developers often think in terms of use cases. They may not write them down but when coding, the idea of them is present. This implicit way of utilizing use cases may just as well be enough for the simpler systems. For the more complex systems that require several developers on board, use cases are usually defined. This part of the processes coincide to some degree. The usage-centered design process propose more formal task modeling as it is more systematic than Exor's work process.

The use cases (or task cases) are supposed to be mapped and sorted once they are defined. This is for the most part not done at Exor. It would be a good idea to do that if the amount of task cases turns out to be quite big. Also, finding a focal task case should be done. This is important to establish which task case the primary interaction space should be based on. It could be the case that for the majority of projects at Exor this step of the process is unnecessary. Because essentially, to do the task modeling is a matter of time. If there is enough time to do it, it is done, or at least some use cases are done.

The next model in the usage-centered design process is the content model. The content modeling is important to make a well organized user interface. If the task modeling has been done properly, doing the content model is not complicated. Since task modeling is not common at Exor the content modeling differs from the suggested content modeling activity of the usage-centered design process. At Exor the content modeling is done more implicitly and is not as structured.

The content modeling helps uncovering which interaction spaces are needed and the tools required to support the task cases. Again, since most projects are fairly small, consequently the system required is quite intuitive to begin with. Hence there will not be that many interaction spaces to deal with. Possibly, for most systems, just one interaction space is needed to support the requirements. Because of that the content modeling is not really justifiable. When dealing with systems that are not very complex it is quite feasible to create a well organized user interface without doing a proper content model. This phase of the processes differ a great deal but for most projects at Exor it is understandable that this modeling activity is not done explicitly.

The implementation modeling is supposed to render a prototype of some sort. Constantine and Lockwood opted to do the first prototype using sheets of paper and Post-it notes. When that is done it is to be reviewed by the users and then refined. At times this is done at Exor, although this is not really common. A more common approach at Exor is to start coding the actual system and once there is something to show, put it up online to get feedback from the customer or user. This approach is arguable. It is time efficient since coding is started immediately, however by using this approach the customer may feel hesitant to give feedback because the prototype may appear as something unchangeable. The customer might think that it is already coded, hence it would take time to change it. Essentially it all depends on the dialogue between the developer and the customer. The developer should express that it is a prototype and if something is not what the customer expected, it can be changed.

Another approach used at Exor when deciding on a good design for the user interface is using a similar web site (most projects at Exor use a web interface) as inspiration. This is good since the customer gets a good idea of what to expect.

Once all models are done they should be verified. Since they are all closely related this can be done using the criteria stated in section 2.4. However all models are not done fully at Exor, if done at all. Hence, the model verification activity is not done at Exor.

What is important in all modeling activities of the usage-centered design process is continuously getting feedback on the model and refining it. This is done at Exor, the prototypes are regularly put up online to get feedback from the customer. This is clearly a usage-centered activity.

5.3 Implementation

According to the usage-centered design process, when all models are done, a firm base on which to start implementing is achieved. As said, a proposed technique is using object-oriented programming, at least for the user interface. As we have seen all models of the usage-centered design process are not developed at Exor, hence the foundation on which to base the programming may not be as comprehen-

sive as desired. This could be a problem for the more complex systems but for the more basic systems this is generally not an issue.

At Exor coding starts fairly early on in the project and since they utilize agile development methods it is harder to conform to usage-centered design. Usage-centered design is very structured whereas agile development tends to be more unstructured. The code-and-fix mentality of Exor conforms quite bad to usage-centering. It may be user-centered but not necessarily usage-centered (note the difference). Coding a quick prototype for feedback and then refining is a proposed technique of user-centered design. In usage-centered design it is preferred to not program a prototype early on, instead an early prototype should be made using paper and Post-it notes. That prototype should then be reviewed and refined in collaboration with the user. Exor do not spend time on models, rather they want to have something to show to the customer quickly. The reason for the quick start of coding is mainly economic since it is Exor's understanding that the customer never wishes to pay for modeling activities. This is unfortunate since doing the models before coding could increase usability, at least for the larger, more complex, systems.

A proposed technique when coding the user interface is using object-oriented programming, mainly because it allows component-based building which in turn supports reusing already familiar interface components. Reuse is highly desired when designing for usage. Exor use object-oriented techniques, thus this clearly conforms to usage-centered design. However, merely using object-oriented techniques is not a warrant that the user interface becomes usage-centered. Though reusing familiar standard components is frequently done for the more traditional web-based systems that Exor develop. This activity is obviously usage-centered.

Usage-centered design proposes implementing the system iteratively. At Exor coding is typically iterative, thus this clearly complies with the proposed methods by Constantine and Lockwood.

5.4 Help

To support help to the user is needed to provide usability. Help can come in different forms, stated in section 2.5. The most cost-efficient help method is constructing instructive interfaces. Exor do build instructive interfaces, they provide pop-up hints resolving the 'seeking identification' and 'seeking clarification' help categories. Most of the time the interfaces are intuitive enough to not require extensive help, so some of the help categories does not need to be supported.

Part of helping the user is seeing to that he/she does not make an error. Error prevention is preferred to displaying an error message. Exor tries to prevent errors by designing instructive interfaces, validating input data and using pre-defined lists when the user is supposed to choose from a set of data. This is very much in compliance with usage-centered design.

Since documentation is minimal at Exor, providing a help section to resolve the rest of the help categories is not done. Though, at the end of every project there is a meeting between Exor and the customer to do a thorough walk-through of the system. This activity clarifies a lot of the functionality of the system. Thus, the need for a detailed all-covering help section might not be as big.

5.5 IT-kompis

Exor's concept 'IT-kompis', where the customer is in direct contact with the developer, is a little special. Though it may not influence the development in terms of usability it could possibly improve usability in other ways. The direct and frequent contact between customer and developer benefits communication. By skipping the middle-man, such as a project manager, helps the developer and the customer to have a clear understanding of the problem. By allowing the customer to continuously be involved in the development it will prevent misunderstandings. The customer will also get a clear understanding of the functionality of the system. Involving the customer or user is clearly a part of usage-centered design. Though Constantine and Lockwood say that the user or customer should not be over-involved in the process because it could be a waste of time for both the developer and the customer. Rather they should be involved at specific points in the process, such as the requirements gathering and reviewing of prototypes. Better results could be obtained if the customer is not involved in the actual designing of the user interface. Involving the customer too much in the process could also lead to requirements creep, which one has to be wary of.

Part of 'IT-kompis' is also providing support if there is a problem with a system. A downside of the concept could be that the developer at any time can be interrupted in his work because a customer is calling for support. This delays on-going projects and the development becomes scattered, which is not ideal.

In conclusion, the concept of 'IT-kompis' may very well lead to better systems in terms of usability, not necessarily development wise, but because of the improved communication the delivered system could be better understood by the customer which is needed to provide usability.

5.6 Suggested modifications

Since the systems that Exor deliver are mainly smaller web-based systems, one could argue the need to fully implement the usage-centered design process at Exor. Many times the systems are fairly non-complex with not that many different use cases, not that many interaction spaces and so on. However, I think there are some areas where Exor could do some changes to try to improve usability.

5.6.1 Users

At Exor, involving the real end user of the system in the early phase of the project is

not always done. I think that this is something that should be looked in to. It is important to base the requirements on correct information. The most correct information is usually provided by the real users. Hence, Exor should try to get the customer to involve some real users of the systems at all times possible. Also, mapping the real users is not done. I think this would be a good idea to do. Try to uncover into which different roles the users can be defined. If not using roles as in usage-centered design, the users could be mapped using personas. This would be useful for the larger systems where the range of different users is wider.

5.6.2 Models

In usage-centered design there is comprehensive modeling activities before a line of code is even written. It is evident that Exor does not work this way. I think that, for the more complex systems, it would be a good idea to at least do the user role model and the task model. If they are done properly the content model is not hard to do, but in Exor's case it may not be possible because of the time and cost restraints.

Overall I think that Exor could spend some more time properly defining task cases, or at least use cases. This activity is generally low prioritized. I think that it could be a good idea to sit down and think for a while, to clarify the explicit problem at hand, before leaping in to start coding immediately.

5.6.3 Prototypes

The biggest problem with Exor's work process that I found is the prototyping. Usage-centered design propose extensive prototyping before coding starts. At first, the prototypes should be low fidelity passive prototypes that are reviewed together with the real users. The low fidelity passive prototypes encourage creativity and also send a signal to the user that this is something that can be changed easily thus encouraging the user to give feedback.

At Exor, prototypes are typically early drafts of the real system. This is in total violation of usage-centered design. I think that it would be great idea to put more thought behind the design of the user interface before programming starts. Though I understand that, as we have seen throughout this discussion, it may not be justifiable due to time and cost restraints.

5.7 Conclusions

It is evident by the discussion that there are many factors involved in designing software. I think in Exor's case it would be quite difficult to completely convert to the usage-centered design process. It is simply not feasible economically, organizationally, nor regarding time. Exor is a quite small company with just eight developers employed. That means that most of the systems they develop has to be fairly small to begin with, otherwise when developing a huge system possibly the whole manpower would be involved. Usage-centered design is more applicable when a bigger system is

to be developed. Because of the small size of the projects, often just one developer, it is just not justifiable (in Exor's case) to formally do all the steps of usage-centered design. But, as I stated in the last section there are some modifications that could be done without too much of an effort.

Since many of the systems developed by Exor contains just a few possible user roles, a few task cases, possibly just one interaction space, all that modeling seems unnecessary. Though it may not be unnecessary to the developer it will definitely seem so to the customer, and this is what it all comes down to in the end. To me it has become clear that the customer is often quite impatient and unwilling to pay more than absolutely necessary for the desired functionality of the system.

My impressions from the focus group sessions is that often the developers would like to work more usage-centered, but it is not possible since money controls what activities are to be done. The customer rarely wants to pay for modeling activities or documentation, hence those activities are skipped.

When I started investigating Exor's work process it soon became clear to me that it differs a great deal from the proposed work process by Constantine and Lockwood. That became a problem initially since the processes differed so much it became hard to compare and analyze.

When investigating Exor, I did interviews, focus group sessions and took courses. I think all of them went very well and cannot think of any better way to do it. I would have liked to test some systems, developed by Exor, for usability. Due to lack of time this could not be done.

Furthermore it would have been interesting to meet some customers of Exor to ask them what they think about the usability of the systems they have received. This could also not be done due to lack of time.

To conclude, I would say that Exor have some areas where they could do some changes in order to work more usage-centered. The big obstacle in the way for working more usage-centered is economics. To me Exor do not seem reluctant to work usage-centered, in fact the customers generally are. This is a huge problem if we ever are going to be able to use some form of usage-centered design.

Exor may not work truly usage-centered, however I would definitely say that to some degree they work user-centered (again, note the difference). In the requirements gathering they have the right focus, then they do quick prototyping and then they get feedback and improve. All of them are user-centered activities.

5.8 Summary

The purpose was to investigate, analyze and evaluate Exor's work process with respect to the process described by Constantine and Lockwood in the book "Software for use - A practical guide to

the models and methods of usage-centered design". The investigation was done using interviews, focus group sessions and courses.

The outcome of the evaluation is that, due to factors we have covered, a complete conversion to usage-centered design might not be feasible nor desirable in Exor's case. It is more reasonable to do some minor changes in key areas such as the ones suggested in section 5.6.

6 Acknowledgments

First of all I would like to thank Jörgen Enström for making this thesis possible. He has since the beginning been supportive and helpful. He invited me to courses given by himself at Exor which was interesting and helpful in the work of this thesis.

Furthermore I would like to thank all developers at Exor for sparing time to attend the focus group sessions which was fundamental for the work of this thesis.

I would especially like to thank Ellinor Källholm for scheduling all the interviews and of course for providing me with useful information in the interview I had with her.

Finally I would like to thank Iordanis Kavathatzopoulos for reviewing my thesis.

7 Appendices

7.1 A: Use cases

Ivar Jacobson first introduced the technique for specifying use cases. A use case describes the interaction between an actor and a system. It is represented as a sequence of steps describing the actor's actions and the system's response. Actors may be end users or other systems. Each use case describes how the actor will interact with the system to achieve a specific goal. The purpose of the use case is to describe the sequences of events that lead to the system doing something useful. The language used is typically that of the end user, avoiding technical terms. Use cases are typically written by system analysts together with the end user. [15]

7.2 B: Personas

Personas was first described in the book "The inmates are running the asylum" written by Alan Cooper. A persona is a description of a typical user of a system. It is an example of the kind of person that could use the system. The reason for defining personas is that to design an effective system it needs to be designed for a specific person. A persona is a fictitious person based on the knowledge of the users. The goal in defining a persona is to bring the users to life by capturing their personalities, motivations and giving them real names. Often a photo is included in the persona to make it even more real.

Personas are not just made up, rather they are based on the results from the requirements investigation. Personas should be specific to secure a greater degree of success. The persona should express the goals of the user so that the designer can see what the system needs to do, and not do. Typically a system will have a set of different personas associated with them. A few of them (not more than three) will be the primary personas. A primary persona is someone who has to be satisfied and cannot be satisfied by a user interface that is designed for another persona.[16]

7.3 C: Agile software development

In agile software development programming is done in small increments. Planning is kept to a minimal as opposed to doing a long-term development plan common in other development methods. Development is done in iterations which are short, typically 1-4 weeks. Within that iteration all elements of the development cycle are included: planning, requirements analysis, design, coding, unit testing and acceptance testing. Each iteration typically delivers some functionality, adding further functionality with each iteration. At the end of each iteration customer representatives review the progress

and redefine functionality priorities. Project progress is generally measured by working software.

The idea behind agile development is to reduce the overall risk and be able to adapt more quickly to changes. Agile methodologies are often used when a product needs to be developed rapidly. Documentation in agile methods is kept to a minimal as face-to-face communication is preferred.[17]

7.4 D: Scrum

Scrum is an agile development method which iteratively and incrementally adds functionality to software. An iteration is called a sprint and typically a sprint is 2-4 weeks long. In Scrum you have a prioritized list of functions to be implemented, called the product backlog. From that list high priority functions are chosen to be included in the sprint backlog which is the work to be done in the upcoming sprint. Each function is given an approximate value of the amount of time it takes to develop. After a sprint there should be a functioning deliverable to show to the customer.

There are essentially three roles in Scrum that are committed to the project: the Scrum master, the developer, and the product owner. The Scrum master helps communication between product owner and the development team, furthermore the Scrum master: protects the team from interference, helps the team be productive and makes sure the actual status of the project is visible and understandable to all. The product owner is responsible for defining the requirements for the product and the priorities among them.[18]

The Scrum development process can be seen in figure 6.[19]

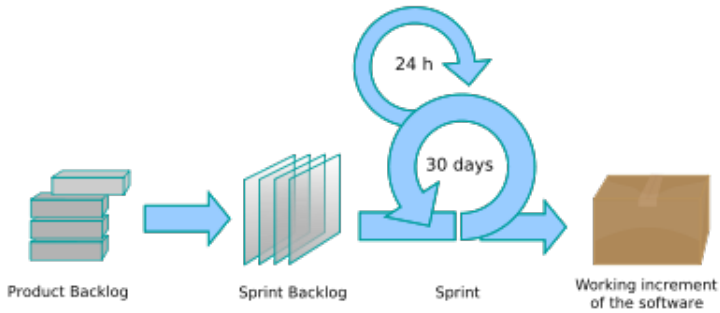

Figure 6: **Scrum development process**

7.5 E: Scenarios

Scenarios are descriptions of the behavior of the system and the environment in certain situations. Scenarios are used to describe and clarify the relevant properties of the application domain. They are also used to uncover system requirements. Furthermore they are used to evaluate and validate design alternatives. [20]

7.6 F: Object-oriented programming

Object-oriented programming is a programming technique that uses objects and the interactions between them to design software systems. The idea is that each object is supposed to be its own ”machine” with its own specific role and responsibilities towards the rest of the system. There are a number of different concepts within object-oriented programming. The most basic ones are class, methods and inheritance. A **class** defines the characteristics of an object. The **methods** are the functions that an object can have. A class may get characteristics from another class using **inheritance**, a sub-class inherits attributes and properties from its super-class.

8 References

Notes

[1]Gerry Gaffney, director of the Melbourne usability consultancy, taken from The Sydney Morning Herald, http://www.smh.com.au/news/platform/usability-at-the-core-of-ipod-emotion/2005/10/31/1130717779105.html?page=2, 2008-10-25

[2]Användarcentrerad systemdesign - Jan Gulliksen, Bengt Göransson, 2002, Studentlitteratur, Lund, ISBN: 91-44-02029-5

[3]Usability Net, http://www.usabilitynet.org/tools/13407stds.htm - 2008-10-27

[4]Usage-centered engineering for web-applications by Larry L. Constantine and Lucy A.D. Lockwood, 2002

[5]Exor.se - http://www.exor.se/default.aspx?pid=2 - 2008-10-27

[6]Beyond User-Centered Design and User Experience: Designing for User Performance by Larry L. Constantine, 2006

[7]Software for use - A practical guide to the models and methods of usage-centered design - Larry L. Constantine and Lucy A.D. Lockwood, 1999, Addison-Wesley, Reading, MA, ISBN: 0-201-92478-1

[8]Usability Engineering Jakob Nielsen, 1993, Morgan Kaufmann Publishers, San Francisco, ISBN: 0-12-518406-9

[9]Exor course for all newly employed, 2008-10-22

[10]Interview with Ellinor Källholm, Exor, 2008-11-07

[11]Interview with Jörgen Enström, Exor, 2008-11-18

[12]Focus group session with Daniel Andersson, Per Hallström and Henrik Hedlund, Exor, 2008-11-11

[13]Focus group session with Fredrik Hultgren, Martin Pettersson and Niclas Våhlin, Exor, 2008-11-04

[14]Focus group session with Andreas Pettersson and Filip Stjernberg, Exor, 2008-11-14

[15]Object Oriented Software Engineering: A Use Case Driven Approach - Ivar Jacobson, 1992, Addison-Wesley, ISBN-10: 0201544350

[16]The Inmates Are Running the Asylum: Why High Tech Products Drive Us Crazy and How to Restore the Sanity - Alan Cooper, 2004, Sams - Pearson Education, ISBN-10: 0672326140

[17]agile-software-development.com - http://www.agile-software-development.com/2007/02/10-things-you-need-to-know-about-agile.html - 2008-12-01

[18]Scrum course at Exor, 2008-10-07

[19]Wikipedia.org - http://en.wikipedia.org/wiki/Scrum$_{(development)}, 2008 - 12 - 01$

[20]Utilizing Scenarios in the Software Development Process, by Kevin M. Benner, Martin S. Feather, W. Lewis Johnson, Lorna A. Zorman, 1993