Jukka Penttinen

USABILITY ASPECTS OF NETWORK ELEMENT MANAGEMENT UNIT

Master's thesis in

Information Systems Science

26.04.2004

University of Jyväskylä

Department of Computer Science and Information System

Jyväskylä

ABSTRACT

Penttinen, Jukka

Usability aspects of Network Element Management Unit

Jyväskylä: University of Jyväskylä, 2004.

69 p.

This research tries to find out usability level of the Network Element Management Unit, NEMU, applications. This was planned to happen via usability evaluations. Used evaluation methods are consistency and heuristic evaluation, Ravden & Johnson method, visual walkthrough and Cognitive walkthrough.

KEYWORDS: Usability, usability design, usability testing, task-centred design

TIIVISTELMÄ

Penttinen, Jukka

Verkkoelementin hallintayksikön käytettävyysnäkökohtia

Jyväskylä: Jyväskylän yliopisto, 2004.

69 s.

Tämän tutkimuksen tarkoituksena on selvittää verkkoelementin hallinta yksikön, Network Element Managenemt Unit, NEMU, sovellusten käytettävyyden tila tällä hetkellä. Tutkimuksen empiirinen osuus tapahtuu käytettävyystestauksien avulla. Käytettävyystesteissä käytettän erilaisia menetelmiä, kuten heuristinen ja yhtenäisyyden arviointi, visuaalinen ja kognitiivinen läpikäynti sekä Ravden & Johnson menetelmä.

AVAINSANAT: Käytettävyys, käytettävyyssuunnittelu, käytettävyystestaus, tehtävälähtöinen suunnittelu

Forewords

First I would like to thank Petteri Mustalahti for the opportunity to made this work. Without Petteri's motivation I would had quitten this a long time ago. I would also like to thank all others who have been involved in this work: Keith Allen, Mauri Haapasaari, Pasi Majakangas, Timo Taskinen, and Mika Toivola from Nokia, Antti Pirhonen and Markku Sakkinen from Jyväskylä University. And, the last but not the least, my wife Satu, she has used several hours correcting my English grammar.

Many things and several people have been changed, including me, around this work. I think that there are more essential things to reach some objectives than spend eternity only for the most ambitious result.

Tikkakoski 26th of April 2004

Terms and Abbreviations

NEMU	Network Element Management Unit
Platypus	Design pattern library
Pronto	Problem report database
Testee	A participant of the test
Tester	A person who controls testees and a test session.
WIMP	Windows, Icons, Menus, and Pointing devices

Content

1 Introduction

Owl lived at The Chestnuts, an old-world residence of great charm, which was grander than anybody else's, or seemed so to Bear, because it had both a knocker AND a bell-pull. Underneath the knocker there was a notice which said:
PLES RING IF AN RNSER IS REQIRD
Underneath the bell-pull there was a notice which said:
PLEZ CNOKE IF AN RNSR IS NOT REQID.
A. A. Milne, Winnie the Pooh

Usability has become an increasingly important feature in any technical device, not only in information technology but also everywhere in technology. Demand for good usability in information technology has become more important because computers have come closer to, and more frequently used by people who are unfamiliar with computer technology. Some wise men said in the 1960's that there would be only a few big computers at the beginning of new millennium. Before those experts even uttered their predictions they were doomed to failure. Nowadays computers surround people everywhere; they have become a part of everyday life for almost everyone. There are even ubiquitous computers like wrist computers or intelligent vacuum cleaners.

Because information technology and computers are generalised there has arisen a need to adapt computers to meet users' needs and not visa versa. In the old times it would have been enough that engineers were introduced how to use a huge computer. Nowadays it is essential to teach engineers how other people act, because those poor engineers have to design information systems which fit into pockets. This same limited knowledge of capability of human actions is restriction in usage of wider information systems. There is often only one person controlling the flow of worldwide information streams on one computer screen. These actions can be so complex that some decades ago maybe hundreds or even thousands of workers would have been required to carry out the same action.

Knowledge and understanding of users and their actions places a tremendous strain on designers. Those who can comprehensively identify the needs of their system users' are probably the winners when customers make their investment decisions. Knowing the users could help and streamline a software development process and it might reduce the need for corrections of applications. A usable system could be more reliable and its users could be more satisfied with their jobs.

Cost efficiency is one of the concerns of project leaders. It is quite normal that usability is not the first priority when project organisations make their project plans. But choosing the right persons to make the right things and embedding a usability process as part of the software development process could lead to better results with less resources.

The main objective of this study is to find out the usability level of the NEMU applications. That work starts with definition of terms and concepts that relate on usability. In the first section defined the term *usability,* subsections have descriptions of usability process and problem. Definition of usability process is dealt with, because the knowledge of the process can make it repeatable. Description of usability problems helps testers and testees work, because the definition of the problem can make it visible. In the next subsection includes descriptions of usability evaluation methods that are used in tests. Another approach of usability is described in next subsection; there are definitions of task based usability process. This section includes methods which can use when starting point for a development work and evaluation is users task. Chapter three is the case NEMU. In this case are described the environment, methods, event, and results of the study. The last chapter makes conclusion of this work.

2 BACKGROUND

This chapter describes the most important definitions and terms in this research. The first subsection describes meaning of concept of usability. Next subsection defines the usability process and its meaning of this work. Subsection three makes overview of problems by defining and describing them. The last subsection includes closer lookup for task-based approach of usability. This last subsection contains methods, which can be used as a designing, implementing, and evaluating usability in a task centred framework.

2.1 The Elements of Usability

According to the Free On-line Dictionary of Computing (FOLDOC, 2003) usability is: "*The effectiveness, efficiency, and satisfaction with which users can achieve tasks in a particular environment of a product. High usability means a system is: easy to learn and remember; efficient, visually pleasing and fun to use; and quick to recover from errors.*" This definition connect usability as a part of programming, but the usability can be a feature of physical as well as software product. This paper deals only with usability of software products.

The usability has many different aspects depending on the definers. Usability can be a synonym to ergonomic. In this context it means a well-organised working environment, or less eye and hand movements for one action. The relationship between ergonomic and usability has been defined more precisely in the International Standard Organisation standard (ISO 13407). Usability can be the quality of a product; for example, the end users' opinion of a better product is that the product that has only a few errors (Bevan 1995, p. 122). Nigel Bevan has defined connections between quality and usability in several studies.

The definition s*oftware reliability* can be the same as usability. John Musa says in his book (1998, p.18) that usability attributes could be also reliability attributes. Those attributes are, for example, ease of learning and ease of relating user problem to system capabilities. The reliability of frequently used functions is the most important thing when choose the functions to be tested.

The first impression of a system, psychologically, is essential and it affects users' opinions of the whole system (Musa 1998 p. 20).

Karat (1997, s. 691) says that a usability attribute is the interaction between a product and its context of use. The context of use cannot be universal. In ISO's standard (9241-11) it is defined that context includes the users, desired goals of users, tasks, equipment to make tasks and usage environment in physical and social level. An impact of attributes of context may vary and hence affect users' experience of systems usability.

Jakob Nielsen (1993 p. 23) has defined a concept of usability. Usability is a part of the system acceptability; see Figure 1 Nielsen's definition. System acceptability is divided into *social* and *practical acceptability.* Social acceptability means all those factors of a system which are related to soft values of the system. This social acceptability could be restriction, for example, of applications in CSCW context. Practical acceptability is divided into several parts, for example, cost, compatibility, reliability and *usefulness.* Usefulness can be separated into two different parts: *utility* and *usability.*

Attributes of usability are *learnability, efficiency, memorability, error prevention* and *subjective satisfaction*. Learnability means that a new user can learn usage of the system easily. Efficiency means that a user can make desired work with the system efficient after s/he has learnt its usage. Memorability means that system functions are easily recalled after some period of not using it. Error prevention means that system has a low number of errors and that the system can easily recover from error situations. Satisfaction means that the system is pleasant to use for a user.

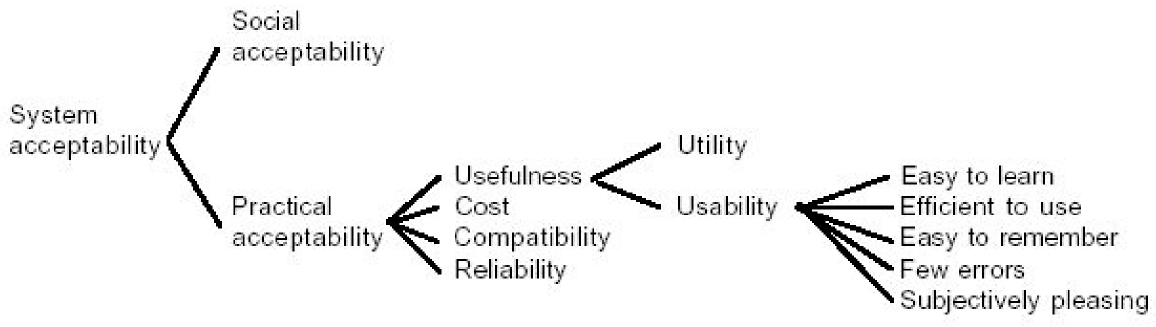

Figure 1 Nielsen's definition

Most of these usability attributes can be measured quite easily except subject pleasing. This is almost impossible to measure directly with some common meters. Can we say that he or she likes this system in some amount of kilograms or meters? Suitable meters are statistical methods; ask users their opinion on systems pleasure.

Ben Shneiderman (1998, s.15) mainly agrees with Nielsen on the definition of usability, but he defined it via meters. Shneiderman's meters are learning time, execution time, vulnerability of errors, keeping the skill, and subjective pleasant. His opinion is that good usability includes clearly observable benefits, which could return the investments. Those benefits could be, for example, increased job satisfaction and reduced stress level.

ISO (9241-11, p. 2) defines usability as "*extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use*". In this context effectiveness means the accuracy and completeness in which users achieve specified goals. The meaning of efficiency is in this case the relationship between expending of resources and effectiveness. Satisfaction is the user's positive attitudes and freedom from discomfort when using the product.

It is quite common that people understand that usability only means, for example, ease of use, attainable or ease of learning. All of these spoken language synonyms of usability are both right and wrong. As already mentioned, none of the usability attributes can define usability alone, because the usability is the collective total of the product's features. User and context of use can affect usability at least as much as well defined standards. The main goal of standard is to create harmony and subjective satisfaction.

2.1.1 Usability process

Nielsen (1993, p. 16) states that usability design is a process, because there are not even two software projects, which are alike. In this context Nielsen means that the thing that might be usable in one project is not in another project. This is the reason why projects have to adapt to the situations. In common meaning a project starts and ends, but a process exists and develops.

In a software development process, a wide variety of usability methods can be used in each phase of the process. There are own methods to each phase or some methods, which cover up the software process from feasibility study to user feedback handling. An approach of a project can be, for example, contextual, process or user-centred. In this paper the main focus is on the task centred approach. Regardless of the approach, a goal of any project has to be a usable product. A task centred process is a process in which the main focus is the users' tasks and main target is to make these users' tasks streamlined and simple. The development of expert users' systems should have the same objective. If the approach of this study were user centred, then main concern would be focused on discovering the user needs that are not yet clearly defined. Because in this case end user needs are quite well defined then task-based approach might be better one. Probably the best results would reach when process uses the best features of both approaches.

A usability process is not a separate path in a software development process though it is part of it in all phases of a process life cycle. The usability process according to Nielsen (1994) is a method of product design which eliminates fences of fluent use of the product. This kind of process utilised guidance and design rationales. Nielsen announced the heuristic usability process as a set of acts:

- Guides and design principles
- Knowledge of cognitive psychology
- User centred approach
- Iterative design and usability evaluation in all phases of process.

Guidance can be, for example, an in-house guide, which describes the size and colour of a dialog and the position of command buttons on it. Guidance can also be based on some standards such as Microsoft's descriptions of Windows. The use of guidelines in a software development process could shorten and reduce, but not eliminate, the number of iterations in the design-evaluate-redesign cycle of human computer interaction development (Strong 1994).

Gulliksen (1996, p.12) argues that standards are necessary and they are good, but are too difficult to grasp and use due to their size and complexity. He also says that guidelines are useful but their origin and quality are often questionable.

Knowledge of cognitive psychology helps software designers to understand the characteristics of humans acting in different situations. Important aspects, which have to be noticed, are for example, memory and perception. Knowledge of cognitive psychology is important when technical and usability requirements are made to sequences of acts. For example, better knowledge of human behaviour has changed the operations of ATM machines. Earlier, the sequence of acts in withdrawing money from an ATM was: feed card, press PIN, input amount of money, take money, take a receipt and the last act remove card. The new improved sequence is; feed card, input PIN, input amount of money, take card, take money and last one is take receipt. The main reasons for these changes were that people often forgot their card in the ATM. This is quite easy to explain by psychological arguments; when a goal of action is achieved, i.e. money has been taken from the ATM; people can leave and forget their cards.

Nielsen (1993, p. 107) emphasises the meaning of iterative design because according to his studies one round of iteration can improve product 38 %. He (Nielsen 1993, p.105) also warns about problems of iterative design if there is only one usability attribute to improve. It is more important to get several notes to assure right changes. Iterative approach is used in several software production methods, for example, in Boehm's spiral model (Boehm, 1988) or in Brooks' prototype model (Brooks, 1975). The iterative method that Nielsen recommends suits very well to the normal software process. A cycle that contains design and evaluation can help designers to make better solutions with minor changes of code, and reduce the time needed in a product's development. Because iterative approach is already known and used in a normal software process, it is one reason why it is easy to adopt Nielsen's thoughts in any software production method. There is a need for usability adaptation of the method. (Nielsen, 1994)

John Gould and Clayton Lewis (Gould & Lewis 1985; Gould et al. 1991) proposed four principles that should guide any development process in which usability is important. The first principle is early focus on users and their needs. The second and third principles together are: on the early stage started and whole process lasting usability testing. And the last point, which they mentioned is iterative design. That is integrated on all the other actions of process, for example, usability testing.

Also Mayhew's (1999, p. 6) approach of usability is iterative. Her usability engineering lifecycle model includes three main steps. These three steps are requirement analysis phase, design, development and test phase and installation phase. User profiling, task analysis, platform capability and constraint defining, and general design principles are the main acts to form usability goals in requirement analysis phase. Designing, testing and development phase are divided into three separate levels. The first level deals with work reengineering, conceptual model design, conceptual model mock-ups, and iterative conceptual model evaluation. The second set focuses on standardisation; screen design standard, screen design prototype, and iterative screen design standard evaluation. The last level concerns detailed user interface design and detailed user interface evaluation. A style guide is made between every level. The last phase of Mayhew's model is installation. The main target of this phase is to collect user feedback after a product has been released. The feedback is very useful when the same product will either be developed or only a few cosmetic corrections are required.

Nieminen et al. (1997, p.11) have defined a usability-engineering process that is an aggregate of design and evaluation, and which has a tight relationship with the lifecycle of a software product. Nieminen et al. process starts with requirement specification and ends with perceiving user feedback. These steps have their own usability acts, which have to improve usability noticeably. He also says that there are some important methods that should be used. He mentions, for example, that in the requirement gathering phase a task analysis should be carried out, and in the implementation phase checklists are essential. In the testing phase, it is important that testing results are compared against usability aims, and in the follow-up phase, it is primarily important that user feedback is used and developers are aware of it.

In ISO standard 13407 (ISO 13407) four design stages are mentioned that lead to better usability. The start of these stages are understanding and specifying the context of use. The main goal in the next stage is to specify the user and organisational requirements. The third stage produces design solutions and the last stage evaluates the previously made designs against requirements.

In the book "A technology for Producing Ideas" Young (1994) presented the following designing process framework: gather facts and information, examine and explore, incubate, give design its birth, examine and refine, iterate (6-8 times), deliver product.

As mentioned earlier, usability depends on its context. There could not be two different companies that have the same kind of project. Even inside one company there might not be the same kind of project. Projects are always different in that at least the product has different end users. Nevertheless usability should be a part of the process in every case. Some methods, which improve usability, are very clear, explicit, and formal, but in many cases it is quite troublesome to find a suitable method or there is not any good solution at all. When some method is planned to be used in a software project its applicability should be evaluated and judged each time. After methods applicability evaluation has become an automated process then it is possible that usability could be a part of an ordinary software development project and live its own life as a process.

2.1.2 Usability Problem

Before it is reasonable to find any problems and contingent faults from a software product we need to define what we are looking for. There could be a situation where the discovered error is some new feature of a system, or some error could not be found because tester or testees perceive that it is not fault at all, or fault is not too severe. The first of the previously mentioned situations is typical when new users are testing a totally new system and the last one when designers are testing their own production.

A usability problem, as any other problem in a software product, can be a fault or an error. The difference between the definition of a fault and an error is that the error is something, which diverges from design principles or requirements. The fault is something, which is too difficult to understand or use.

The fault can also be a design defect. These definitions do not mean that a poorly designed but exactly implemented application is perfect and error free. On the other hand, that perfectly designed and implemented software that is used by wrong people or application, which is used in a wrong place, is faulty.

A usability problem must be well defined and a part of other faults, which have been found (Wilson & Coyne 2001, p. 18). They emphasise the fact that the usability problem is an ordinary problem, which can be fixed. That thought does not expand the gulf between an ordinary software design and a usable software design.

Pressman (1997, p.196) says that a *fault* is a problem that is found after software is released. An e*rror* is a problem, which is discovered before the products are released.

Haikala and Märijärvi (1998, p.261) say that an *error* is divergence between implementation and its requirements. Without an exact and accurate definition we are not able to observe an error. Since all software includes errors, it can be possible that the error can affect the fault when the application is running. The error can also be non harmful or it can cause trouble situations, which users feel as non-welcome features of the system.

Kaner et al (1999, p. 60) expand the theory that Haikala and Märijärvi used in a form that an error is a divergence between an application and its specification if, and only if, the specification exists and it is correct. They agree that poor specification leads to a totally poor program, not perfect. Thus they recommended Myers definition (1976): there is a program error if the program does not do what a user expects it to do.

According to Musa (1998 s. 85), an error is an incorrect or missing action of a person or persons that caused a fault in a program. A software fault is a defective, missing or extra instruction or set of related instructions that is the cause of one or more potential failures. Errors can be classified where they are arising in: communication, knowledge, incomplete analysis, and transcription errors. There can be three different communication errors; inter-role, intra-role and temporary. Inter-role means that there is a lack of communication between a customer and a designer, a designer and a coder, a designer and a tester.

Intra-role errors are among designers, coders, and testers. The last ones, temporary errors are errors between different times for the same person. A transcription error occurs between a mind and paper or a machine. An example of the transcription error would be wrong text in a menu or in a command button.

Usability error is a design flaw that causes a fault, which leads into error behaviour, and furthermore degrading performance of the system. Users have their own idea how the system should work. Designers also have their own thoughts on how the system has to work. In the case of a design defect, designers and users have different models in their minds. The designers and users have their own mental model of the system. If there are differences between designers' and users' mental model it will cause a conflict (Norman 1988, p.140).

According to Sinkkonen et al. (2002, p.133), a designer "sends a message" to a user via the user interface. If the user does not understand the signal (characters) then the message will not get across. Or if the context of the product is not familiar to the designer, then s/he probably makes wrong choices of terms and errors. Those errors can lead to a strange action of the system; the user does not understand designer's idea.

Usability problems can vary from minor cosmetic divergence to major functional disability. Classifying the severity of the usability problem is quite difficult because of users' needs, ability and possibilities. The same problem in different context and viewed by a different user can be totally different on its severity and overall meaning. A novice user can feel something unusable but an expert user has no problem at all with the same thing. Because usability is subjective it is hard to find a universal severity scale. There is some kind of rating commonly used in Nielsen's heuristics. This five-step scale is used to describe the severity of discovered usability problems. Only levels one to five are relevant, because the level zero means that there is no problem at all or testee did not faced with it. Level one means that found problem is good to be fixed in next release and the level five means that there is a catastrophic problem, the whole system is totally unusable.

This four-step severity scale is used in this research for all usability problems. This scale is the same as the Nokia Pronto database, which is used for Nokia in-house testing. These main levels define a problem's severity. Most of the cases are placed in levels two and three as in normal distribution; only a few cases are in level four. Level four is used for problems, which are mainly ideas for the next release. Hardly any of the problems are in level one. Level one is for catastrophic cases. All of the main levels divide into two to four sub levels. These sub levels are used only for fine-tuning the seriousness of the problems.

When the aim is a usable product, what is tested and what we are looking for are not the same. Only the definition of searched and possibly found problems could rationalise the whole project. An exact definition of problems helps a team to avoid the biggest pitfalls before testing because concentration on wrong things wastes resources and can generate disagreement concerning usability.

2.2 Usability Evaluation

This study has two main focuses. The first is usability evaluation and other is discovering a method or methods that can improve the outcome. The objective of the first part of the study is to determine the level of usability of the NEMU applications. This chapter determines and makes an overall view of those methods that are useful in achieving this goal. Some of these methods are quite useful because they are used in many phases of software development. See, for example, scenarios in (Nielsen, 1993, p.100). They are applicable in designing and in the final testing phase. Dumas and Redish (1993, p. 99) argue that usability evaluation can give extra value to a product and a company. They think that those values could be; sales increase, growth in the reputation of a company, increased productivity, reduction in resources needed in training and support, also reduced need for new versions and corrections, and easier creation of helps and training. Rubin (1994, p. 27) states that it is better to test something than to test nothing.

Rubin (1994) states that normally formal tests are used to prove or refute hypothesis and less formal tests are used to discover and correct usability problems. Increase of formal methods needs a larger number of testees and exact design and implementation of tests to ensure validity and reliability of tests and their results.

Dicks (2002, p. 26) says that it is important that usability testers know the limitation of methods, so they can take a critical view of test results. This is important because it has been too difficult to persuade most managers to invest in usability testing at all. Dicks introduces five general categories of misconceptions that lead to non-usable outcomes.

1. Misunderstanding of the concept of usability itself and of the distinction between usability studies and empirical tests.
2. Two types of problem with statistics: assuming that a set of quantitative statistics equals a usability test, and misusing statistical results, especially from tests performed without large enough user samples.
3. Using usability test for verification rather than usability.
4. Lack of knowledge of the limitations and the proper methods of usability testing to ensure valid and reliable results.
5. Testing for ease of use but not usefulness.

Item three in Dicks' list warns about mixing usability test and verification. Dicks (2002, p.28) says that it is not a usability testing to test documentation against a system or a part of it. This kind of test tests the usability of documents and also the usefulness of documentation.

Usability evaluation is a common term of usability inspection and testing methods. Usability testing is based on observing of the testees' activity and inspection is based on evaluation with the usability rules. Usability or domain professionals make evaluation with using different rules, heuristics. Usability inspection can be done in any phase of a system developing process. The only criteria are to use suitable methods for evaluation. Usability testing needs a working model of a system or some suitable model, which represents the evaluated system. These kinds of system representations can be:

- Mock-ups are system models without functionality. In other words, the designer makes a non-functioning prototype of the system. These prototypes do not have any functionality of them components. They are only snapshots, drawn pictures of the desired system or, for example, an overhead projector transparency, which is changed by the tester (Rowley & Rhoades, 1992, p. 391).

- Prototypes could be an exact implementation of some important features of a coming system or they could be made like a mock-up, but then there is a quite reduced accuracy of systems features. The first one is called a vertical prototype and the second one is a horizontal prototype. The reasons for using either prototype depend on the number of feasible features or the whole system's criticality (Nielsen, 1994, p.94; Nielsen, 1987; Life et al. 1990).

- Scenarios that are in written form of the desired system. These stories have some defined use. The user uses a certain computer system with which a certain task will be done. The task will be done in a defined environment and according to defined schedule (Nielsen, 1993, p. 99).

- Wizard of Oz-model is the same kind of system figure as a mock-up, but it has human controlled functionality. Wizard of Oz is composed of two parts: user and system replica. System replica is a system figure that is co-ordinated by human. The test is simple. The user makes the desired action and the system replica responds as a real system. Normally the user does not know that the system is not "real". In this method some basic information can be easily collected from the target environment and its users. This method is appropriate for the study of human behaviour and/or for systems requiring future technology.

As a usability inspection method can be, for example, walk through or heuristic method. Both of those methods, usability testing and inspection, can be made in a laboratory or in an actual environment.

Walk through is an evaluation method that tries to simulate actions of real end users in a real system. In walk through, the testee is given a scenario, which describes the intended action and the testee tries to make it by using the system's mock-up, prototype or already implemented system. One walk through type evaluation is described more specifically in subchapter 2.2.4 Cognitive Walkthrough.

Heuristics are group rules and regulations that are focused on important usability factors. The most commonly known usability heuristics are Nielsen's (Nielsen 1993 p. 20) ten usability heuristics; more is said about these in the next subchapter. Heuristics can be basic guidelines to designers and developers.

2.2.1 Heuristic Evaluation

Heuristic evaluation is a usability inspection method which is based on the usability principles called usability heuristics. Testees evaluate each element of a dialog by comparing it with heuristics. All divergences which are found in evaluation meeting are usability problems. This method was introduced for the first time by Molich and Nielsen (1990). One of the advantages of this method is cost efficiency. This method is also good when it is used with a small number of evaluators.

Also one of the benefits is a quite low requirement of evaluators' knowledge of usability. Nielsen (1993, p. 156) claims that with this method up to 75% of usability problems of the system can be found with a group of only 3-5 evaluators.

The evaluators go through the explored interface. In this method, the testees compare the found objects, for example, menus and dialogues of the system to heuristics (Molich & Nielsen, 1990 p. 339; Nielsen 1993, p. 20; Nielsen 1994a, p. 30; Nielsen 1994b, p. 153). All findings of the evaluations are compared to usability principles, which are:

- Use simple and natural dialogue
- Speak the users' language
- Minimise users' memory load
- Consistency

- Feedback
- Clearly marked exits
- Shortcuts
- Good error messages
- Prevent errors
- Help and documentation

Evaluators make their own estimations of the severity of each found problem. All, which are found, have to be compared with the same scale. Nielsen (1993) has presented two different ways to make severity ratings.

One of the possible models is that testers make their estimation of usability problems and their severity after a test meeting. Factors that affect the severity of the problem are the frequency with which the problem occurs, and the impact of the problem if it occurs, and the persistence of the problem (Nielsen 1993, p. 103).

The rating scale of usability problems can be five steps from zero to four, for example;

0 = Not a relevant usability problem.

1 = Cosmetic problem only. Needs to be fixed if there is extra time available in a project.

2 = Minor usability problem. Fixing should have low priority.

3 = Major usability problem. Important to fix with high priority.

4 = Usability catastrophe. Imperative to fix before release.

Another possible way to estimate problems is to make co-ordinates, where one axis is the number of users who deal with the problem. The other axis defines the severity of problem. The scale of the axis can be two or multi divisional (Nielsen, 1993, p. 104).

Heuristic evaluation does not advise how to fix the found problems; that is one of the weaknesses of this method (Nielsen, 1993, p. 102). These weaknesses can be overcome in debriefing sessions. In this session, all the participants can make their own suggestions on how to fix these problems.

2.2.2 Discount Usability Engineering

Discount usability engineering, DUE, is a light version of Nielsen's heuristics. Nielsen introduced it for the first time in his book (1993, p. 17). This method is based on the hypothesis that anything is better than nothing. If it is possible to collect any data of a process, then it is possible to make improvements in the software development process. The aim of this method is to be an easy and cost efficient way to improve usability in software projects. This method is not supposed to be the most efficient method to treat usability; rather on the contrary, it guarantees some kind of the base of to improve usability process. Nielsen also suggests using parallel designs to make designing faster, for example, there should be more than one team working on solutions at the same time. One of Nielsen's aims might be to create a method that can spread as wide as possible, because of its easiness and low costs. The method is quite easy to use even for those who have no previous experience of systematic usability.

Discount usability engineering is based on four main principles:

- **Observing users and their tasks** as invisibly as possible.
- **Scenarios**, written stories of system. A more precise description of scenarios is included in section 2.2 page 7. For example, scenarios can be used with a paper version of a user interface.
- **Simplified thinking aloud**, which means that the testees at the same time when they make their tests they tell what they are thinking about and what they are going to do next. In this method it is quite easy to determine what users want to do and why they act in a certain way with the application. The method is so simple that a tester does not need to be an expert in psychology. Normally testers write notes and possible recordings are precise enough to document tests.

- **Heuristic evaluation**, these are defined more precisely in chapter 2.2.1 Heuristic Evaluation. Non-experts can also apply heuristics to many different problems; however a tester should have a moderate expertise to get the best results.

Instructions for handling DUE administratively:

- Recognise the need of usability in the organisation.
- Make sure that usability is understood at management level.
- Organise resources.
- Integrate usability as part of the development process.
- Check that every user interface is tested.

Or this method can be even simpler: Choose one of the user interfaces. Do a simple usability test in which some of potential customers make a test without extra guidance. If any problems are found in the tests, use the iterative approach in the next version. “If no usability problems are found, then be happy that you have been lucky (on you project)” (Nielsen, 1993, s. 21).

Nielsen reports evidence that groups of assessors are more effective in identifying problems than individuals and that experience in usability engineering as well as specific domain expertise are useful. He also argues that the optimal sizes of such assessment groups are smaller when such "double specialists" are used, i.e. 2-3 persons are sufficient, rather than 3-5 when only usability specialists are used. He also says that with novices much larger groups (i.e. 14 members) are needed to identify most of the problems. However, some evidence was also reported that usability problems concerning an existing system are difficult to detect by this method, and also that it can be difficult to find missing dialogue elements using paper mock ups rather than full prototypes (Nielsen, 1993).

It is proposed that designers can use the technology themselves with experts or users, but that the best results are obtained when experts rather than end users are used to perform the assessment. The technology is intended to be used in the early stages of designing, with the rapid prototyping of design solutions.

The effectiveness of the technology is still being debated at the time of writing. It is a recent development that moves away from the conventions of performing detailed trials with larger numbers of subjects. One of the weaknesses of the approach is the assumption that it is easy to predict how many experts are needed in order to identify problems with a given system, and Nielsen uses a rather simplistic model in making his argument. However, the approach is likely to be in use as it is cost effective and fairly simple to apply. With all such techniques, however, it should be recognised that the quality of the assessment produced will be largely dependent on the range of tasks, which users try and perform with a system, and if tasks are not suitably representative, they are likely to miss some of the problems that users will have in practice.

2.2.3 Participatory heuristic evaluation

Muller and McClard (1995, s.115) have modified Nielsen's heuristic usability evaluation method to be more process oriented. The first version of this method includes all of Nielsen's heuristics, see section 2.2.1, and three new heuristics. Improvements, which have been added, are: process-oriented heuristics and to treat users as work domain experts in an evaluator group. Muller (Muller et al., 1998, p.16) has released an improved version of this method. The latest addition entails the concept of privacy. All of Muller's heuristics dealing with tasks and work supports are:

- Support human skills,
- provide a pleasurable and respectful working experience,
- help the user to provide quality results to his/her clients,
- aid the user in protecting the privacy of all involved his/her work.

Participatory heuristic evaluation follows the procedure of normal heuristic evaluation. The main difference between participatory heuristic evaluation and normal heuristic evaluation is that participatory heuristic evaluation concentrates more on users. Muller et al. (1998, p. 15) have noticed that, in certain circumstances, it is good that evaluators are members of a developing team. They say that they should get and motivate real users in usability evaluations, because these people are experts in their work domain. This method cannot substitute for user's involvement in designing or user testing, but this method could be a good aid when designing prototypes Muller, 1998, p. 18).

2.2.4 Cognitive Walkthrough

Cognitive walkthrough uses a precisely described process to simulate the user's problem solving process. This method requires that at least usability or domain experts use it. These testers walk through the task scenarios, stories that describe the execution of the task and then they compare it with the functions that are implemented in the system. This method can be used in the early stage of the development process; even a sketch of the system can be tested.

Cognitive walkthrough was introduced for the first time in Lewis et al. (1990) publication. The authors have improved original study later (Wharton et al., 1994; Lewis et al., 1997). This usability inspection method focuses on ease of learning. This method is based on the theory of explorative learning and also Norman's (1986) theory of action. Norman's theory of action divides user action into a cycle of *execution* and *evaluation*. He divides the phases of the cycle into seven stages, between user's goals and the physical system. See Figure 2. The parts of the cycle are; at the first, establish the goal. The second group of actions: forming the intention, and specifying the sequence of action. The last group of stages includes executing the sequence of actions and perceiving the system state. In the last group there are interpreting the system state with respect of user's own expectations and evaluation of the system state with respect to user's goals and intention of interpretation. Then the cycle starts again if needed.

All the possible functions of all the system's objects, for example, the menu and command buttons of a window, makes the *affordance of the system.* A simple example of affordance could be a printer icon in MS Word menu bar that makes of vision of printing existent view. Affordance is the most important factor of the system in stages 2 to 4 of execution and the evaluation cycle.

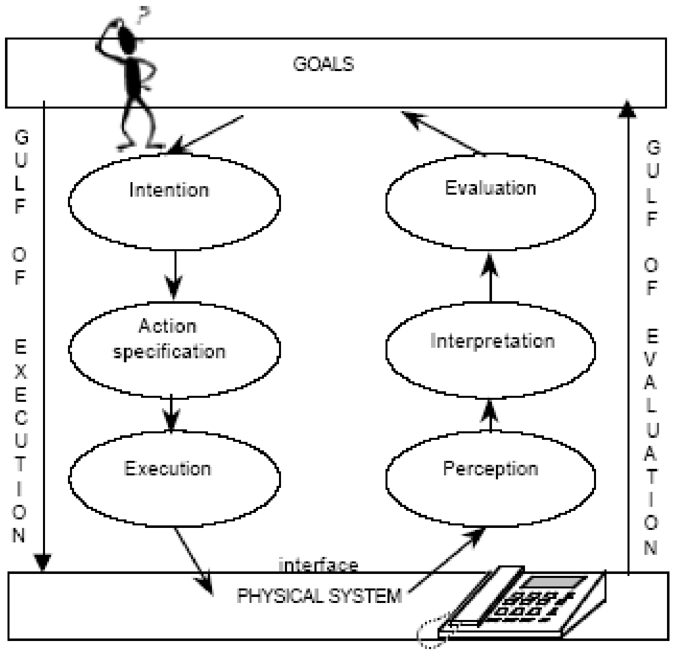

Figure 1 Cycle of execution and evaluation

The process of cognitive evaluation can be divided into five steps (Wharton et al. 1994):

1. Define inputs for the walkthrough.
2. Find a group of analysts.
3. Walk through the tasks.
4. Record critical information.
5. Think of ways to fix the problems.

Cognitive walkthrough gives detailed guidance for steps one to four. Guidance for the last step might not be relevant, because of the need for wide variety of usability. Usability depends on, for example, its context and users.

The preparation phase, which includes steps one and two, gathers basic information about the researched system. This information includes data of the end user's skills and knowledge of the intended system. Also knowledge about the tasks is important information; its sequence of action, because all system functions are not rational in a survey. The most important parts of the interfaces of the system are reasonable to evaluate alike in paper sketch and in final implementation form.

A walkthrough session can be individual or a group process. Some researchers' opinion is that a group session is more efficient in finding problems, but this method's originators suggest this kind of method have to be called a *pluralistic usability walkthrough* (Bias, 1993, p. 63).

During evaluation, the testees study the interaction between users and the system. Testees concentrate only on the selected tasks, because all of the explored tasks have to be studied carefully. At first it has to be analysed how well the system supports the users when they are forming a goal and sequence of actions to achieve the goal. Next it has to be analysed how well the system gives hints and tips on executing the actions. In the third part it is analysing the feedback of the system. Every part of the evaluation has its own form, which supports the testees' analysis of the system (Wharton et al., 1994). Samples of these forms are included in appendix 7.1.

The originator of this method argues that this method and explorative learning have close connections (Lewis et al., 1990, p. 241). From this point of view there can be one important reason to use this method. That reason is that normally users do not read manuals, they want to start using a program by doing and exploring it. The system must show all the hints and tips, which can help users in their aims to set up goals. The system must be easily learnt so that a normal user can start using it without reading the manuals or excessive training.

Preece (1994, p. 679) mentions problems with this method. She argues that this method is only capable to find problems, which are restraining the learning of the system. She also claims that this method does not take notice of most of the other cognitive factors like memory.

2.2.5 Ravden & Johnson method

This method was introduced for the first time in the book by Susannah Ravden and Graham Johnson (Ravden & Johnson 1989). This method gives results quite easily but it is quite demanding concerning testees' knowledge of usability and context area and it is relatively time consuming. A professional can use this method and it can be only an extra value method. This method is not recommended as a single usability inspection in a software project.

In their book, Ravden and Johnson describe a practical usability evaluation method, which uses an *evaluation checklist*. Analysis of user interaction is carried out with a checklist: users fill in detailed checklists about the acceptability of various aspects of the interface. Thus testees are highlighting particular types of problems. The method is designed for use by the developers together with the users. With this method it is also possible to use evaluators who may not be the actual users of the designed system. (Ravden & Johnson 1989)

End users and developers together start the evaluation. In the evaluation meeting, testers go through all the tasks of the system and compare them with the checklist. The checklist contains questions on several aspects of usability. The questions are based on general user interface guidelines, see

Table 1. They are divided into 9 sections, each containing from 13 to 17 questions, and the user is expected to answer questions like "Does the system validate user inputs before processing, wherever possible?" or "Where interface metaphors are used, are they relevant to the tasks carried out with the system?" on a 4-point scale ranging from *always* to *never*. In addition, the checklist has also two sections of questions on general usability problems encountered during use.

This sort of checklist is not cost efficient to use and it is quite difficult for the users to fill in. There is a problem with these questionnaires: the user may find parts of the software usable; and other parts difficult to deal with. As the checklist is aimed at giving an overall evaluation of the software, it might not provide advice for the developers. What corrective actions would have to be taken if the user's answer to a question on software compatibility doesn't agree with user expectations sometimes?

Table 1 The Usability Guidelines (Ravden & Johnson 1989)

Feature	**Description**
Visual clarity	Information on the screen should be clear, well organised, unambiguous and easy to read.
Consistency	The way the system looks and works should be consistent at all times.
Compatibility	The way the system looks and works should be compatible with user conventions and expectations.
Informative feedback	Users should be given clear, informative feedback everywhere they are in the system, what actions they have taken, whether these actions have been successful and what actions should be taken next.
Explicitness	The way the system works and is structured should be clear to the user.
Appropriate functionality	The system should meet the needs and requirements of users when carrying out tasks.
Flexibility and control	The interface should be sufficiently flexible in structure, in the way information is presented and in terms of what the user can do, to suit the needs and requirements of all users, and to allow them to feel in control of the system.
Error prevention and correction	The system should be designed to minimise the possibility of user error, with inbuilt facilities for detecting and handling those, which do occur; users should be able to check their inputs and to correct errors, or potential error situations before the input is processed.
User guidance and support	Informative, easy-to-use and relevant guidance and support should be provided, both on the computer (via an on-line help facility) and in hard-copy document form, to help the user understand and use the system.

Using this kind of a checklist as a developer tool for heuristic evaluation of user interfaces would probably make it easier for the developer to identify usability problems. But it would not help the developer in focusing attention towards the most important aspects of usability.

2.2.6 Consistency Inspection

In this method, the main target is to find differences between the same kinds of applications. These differences can be functional, layout, colour scheme, or textual divergence. Functional consistency means, for example, that the same dialog must do the same thing in different applications. Layout and colour scheme consistency means the same kind of window component layout and colour through the system, for example, command buttons are in the same place and order in dialogs and any mandatory text field has a yellow background. Textual consistency keeps an eye on all the texts in an application. An example of textual consistency means that menus and menu items must be the same. Information, warning and error messages all have the same style. The same abbreviations are used for the same meaning everywhere in the applications. (Wixon 1994)

In order to carry out consistency inspection you will have to first build up an inspection team. There has to be at least one developer of all applications and one usability expert. This jury goes through step by step all components of all the applications. The best way to do this is to project a computer screen onto a silver screen so, that all the participants can comment the same application or component at the time. If there are in-house development instructions or orders then all the divergences found have to be compared against it first and after that against other applications. This technology is best used in the early stages of development, when initial development work has not progressed to the point where products that require extensive changes to ensure consistency will not require total overhauls. The ideal time for consistency inspection is when design documents for each of the individual products are almost done, and before any actual work on building the products has commenced. This method normally finds dozens of inconsistencies and, as a result, there is normally quite a big list of changes to do.

2.2.7 Visual Walkthrough

This technology is included in some of the above methods, but it can be used as an individual method. In a visual walkthrough, the tester uses a checklist to determine those parts of the system that will be ambiguous or unclear. Nieminen & Koivunen (1995) have developed this method to get information about users' perception and interpretation of the user interface and its components. The basis of this method is Dehlholms (1992) picture analysis, and, more precisely, the first two steps of it: In the first step, describe and identify the visible elements in the picture and describe their relationships with each other. And in the second step analyse the meaning of the elements in the picture, in this context the picture would be an interface.

Visual walkthrough is a method that cannot be an only usability testing method; on the contrary, it can be used to complement a common usability test and thinking aloud method. It is applied at the beginning of a test session before any test tasks. In the test, the participants are asked to tell what components and groups of components they see in the user interface and what they think their functionality is. The aim of the visual walkthrough is to gather information of what the users notice in the interface, in what order they notice the components and in what sort of groups, how they understand the visual messages, and what terms they use as they describe the system and its functionality.

As a conclusion of the previously described methods, we can say that every method has its own strengths and weaknesses. The tester must be familiar with the methods and their properties this is the main point to achieve the best results in testing. Suitability to the purpose of using the previously presented methods is listed in Table 1 (see below). The table shows the usability methods and the product lifecycle stage in which a method can be used. There are different methods for different phases of the product development lifecycle, many of them being complementary. Because none of the methods is better than any other, instead of using only one method a combination of methods should be selected that is most beneficial for the case at hand.

Table 1 Suitability of Usability Methods.

Name	Task analysis	Designing	Implementing	Testing	Follow-up
Heuristic evaluation		x	x	x	x
Participatory evaluation		x	x	x	x
DUE, Discount usability evaluation		x	x	x	x
Cognitive walkthrough		x	x	x	x
Consistency evaluation		x	x		
Ravden & Johnson method		x	x	x	x
Visual walkthrough		x	x	x	

2.3 Task Centred approach

A software design process could have many different kinds of approaches. The choice of approaches is quite broad, commonly used are, for example, user centred, task centred, contextual, or hierarchical methods. Normally, if a process tries to get usable output, a user centred approach is used as a design process. In this approach the user is in the main focus of the system work.

In the NEMU environment, user centred design methods cannot be used as normal developing method because the context of the desired system is different than a “normal” software development target. There are some requirements and restrictions that the NEMU system environment has: The users are professionals in their work domain, they have their own special way to work, and the target system is complex, dynamic and rapidly changing.

There is a family of applications, which are not universal. Some of the users are used to work with old computer systems that have command-based control systems that is the one reason why users could not have or they could have very constricted WIMP skills of their work environment. Some of the applications are used on a daily basis and some are used very rarely, maybe those applications are needed only once a year or in an extreme case of emergency. Because the target system users have different educational and cultural backgrounds, in all cases the system must support their way to work too. In this study the main focus have been set a task centred approach, because the methods of this approach can offer support for most of the previous mentioned requirements and restrictions.

2.3.1 Contextual Design

Contextual design (Holtzblatt & Beyer, 1993; Beyer & Holtzblatt, 1998; Holtzblatt & Beyer, 1999) is a method that can provide a tool for gathering data from the users' work context, for example, data of real workflow and environment. This method is based on field research. The basic goal of this method is to provide designers grounded and detailed information of users and their tasks. This method defines coherent series of actions, which lead to a better and well-designed system. As mentioned earlier in chapter 2.1.2, problems are not the same as every system and organisation has their own.

This is quite demanding for the method, which has to accommodate specific needs. Contextual design provides a useful framework for tailoring a design process (Beyer & Holtzblatt, 1999, p. 33). Contextual design can be divided into six phases; see

Figure 1 (Holzblatt & Beyer 1999, p. 33). The first phase is contextual inquiry where users are observed and interviewed in their own work environment. The next phase is work modeling. In this step, the main target is to make concrete and detailed models of the users' work. The models are made to help designers to find important details and clarify the complexity.

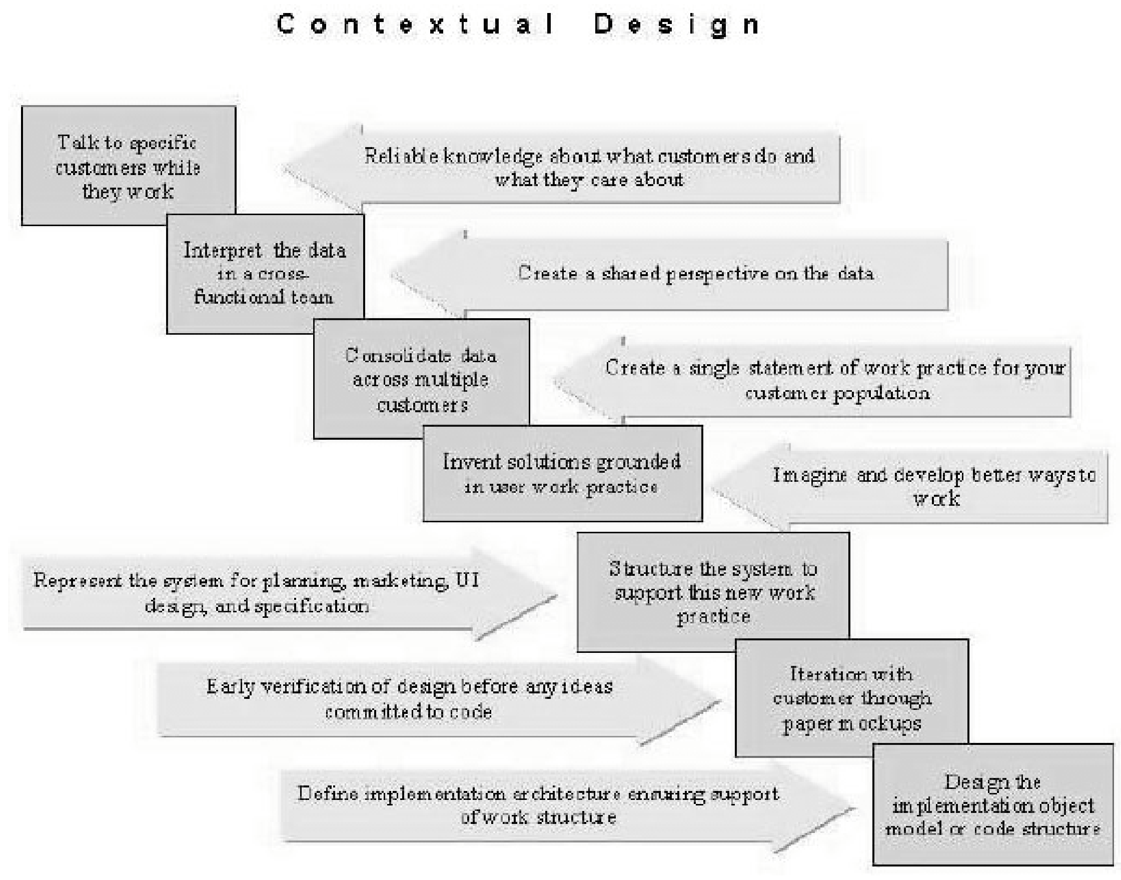
Contextual Design
Talk to specific customers while they work
Reliable knowledge about what customers do and what they care about
Interpret the data in a cross-functional team
Create a shared perspective on the data
Consolidate data across multiple customers
Create a single statement of work practice for your customer population
Invent solutions grounded in user work practice
Imagine and develop better ways to work
Represent the system for planning, marketing, UI design, and specification
Structure the system to support this new work practice
Early verification of design before any ideas committed to code
Iteration with customer through paper mockups
Define implementation architecture ensuring support of work structure
Design the implementation object model or code structure

Figure 1 (Holzblatt & Beyer 1999, p. 33)

The third phase is to consolidate and combine the findings and the models from different users into one model that includes every observed user. In the fourth step design team makes work redesign based on the previous findings. These new models of work have to include all the essential aspects of users' tasks, but it should bring something new to it. In the next phase the user environment design, which describes the workflow, is made. This new workflow model is based on the desired system. Mock-ups and tests with customers are in the sixth phase. And the final stage is to put the plans into practice, which takes note of environment, requirements, and key features of the system (Beyer & Holtzblatt, 1998, p. 22).

Holzblatt & Beyer (1999, p. 33) have added one phase to the end of this process, which is to design the implementation object model or code structure. Contextual design has some disadvantages. Firstly process is quite informal because there are no question forms that participants complete. Instead of forms, the interviewer must observe carefully the user's tasks and their answers. For this reason this method is demanding for the tester. And other disadvantage of this method is that it is quite expensive, because of the field study. Often the use of this method is still reasonable even as resources are spent at the design phase, because correction of the software after release is much more expensive.

Norros (2003 p. 2) sees some advantages of the contextual design. At the first, the strength of the contextual design approach is that it changes the focus from the design of technical products or services to the construction of new ways of work. Secondly this method has well-defined design process to the analysis and modelling of the users' real practices. Also essential factor is in the concept, when it shows the designers how they have to learn to what this information means for design. Norros also criticizes this concept, because the approach is both prescriptive and descriptive. For example, the analyses of use are restriction to human-technology interactions. She also argues that the unpredictability of open systems and the context of use would demand modelling of human behaviour in a way which has no restrictions.

2.3.2 GOMS

There are some methods that can evaluate learning usage of programs and tracing execution of actions. In this chapter two methods are introduced. These methods are good when estimations of execution or learning times are needed. The theoretical base of these methods is cognitive psychology.

GOMS is an acronym that stands for Goals, Operators, Methods, and Selection rules. This model describes the content and structure of data and skills that are needed when routine tasks are being executed. This method, also like cognitive walkthrough, is based on Norman's (1986) theory of action.

This theory is described in chapter 2.2.4 Cognitive Walkthrough on page 7. The GOMS model is composed of *methods* that are used to achieve specific *goals*. The methods are composed of *operators.* The operators are specific steps that a user performs and are assigned a specific execution time. If a goal can be achieved by more than one *method*, then *selection rules* are used to determine the proper method.

Kieras (2002, p. 1153) suggest three arguments why it is essential to use GOMS models. The first argument is the rationality principle; normally a human tends to act in the most reasonable way. Kieras makes reference on the Card et al. (1983) publication in this context. The second point is procedural primacy, which means that normally people do their tasks in certain order regardless of system status. The last argument is explicit representation, when it is tending to estimate advantages of a system that must be based on the clear, explicit, and formal definition.

In the highest-level of GOMS model, the tasks are divided into subtasks. The certain subtask consists of a sequence of actions by which the tasks are executed. In the lowest level of the model there are user actions that are reviewed when they are executing subtasks. The result of this model is a hierarchical description, which describes the task, which is divided into subtasks. And there are also the actions, which are needed to execute those subparts.

There are four basic GOMS models. These four models have different variations on dealing with complexity and modelling different activities. These available models are:

- KLM, Keystroke-Level Model, rapid execution time evaluation method, oldest of all GOMS models (Card et al., 1980). This method is not originally a GOMS model, but John & Kieras (1994) have joined it as result of their studies,

- CMN-GOMS, Card-Moran-Newell-GOMS, Original GOMS model (Card et al., 1983),

- NGOMSL, Natural GOMS Language, model can be used when learning time estimations are needed. This model expands the CMN-GOMS model by adding natural language features. The model is developed by (Kieras, 1988; Kieras, 1997),

- CPM-GOMS, Cognitive Perceptual Motor GOMS, also known as Critical Path Method GOMS, is based on the parallel multi–process stage model of human information processing (John & Kieras 1994).

The most important benefits of using the GOMS model in the design process are coverage and consistency. Coverage and consistency mean that the model can reach most of the system's function points and all of those function points are treated in the same way. Also in this model, it is easy to get operation sequences and estimates of execution time because of coverage. Comparing of the results is more reliable because operations are walked through in the same way.

GOMS can be used both quantitatively and qualitatively. Quantitatively, it gives predictions of execution time and learning of system. With these quantitative predictions, you can examine such trade-offs in the light of what is important to your company, and what is relevant to your user-group or task situation.

Qualitatively, GOMS can be used to design training programs and help systems. The GOMS model is an exact description of the knowledge needed to perform a given task and thus it describes the content of task-oriented documentation. You only need to tell the new user what the goals are, what different methods could be used to achieve them, and when to use each method (selection rules). This approach has been shown to be an efficient way to organise help systems, tutorials, and training programs as well as user documentation.

GOMS can be used in various applications and fields. For examples, CPM-GOMS, which can represent several active goals at the same time, has been developed for using a telephone operator workstation. More about these application topics can be found in John and Kieras (1994).

KLM-GOMS provides a simple framework for describing expert behavior as linear sequence of steps, and its simplicity has benefited evaluations of straightforward interface and tasks, for example, the telephone operator task (Gray et al. 1993). Simple KLM-GOMS model can provide a total task time for given interfaces, but does not make any prediction, for example, about learning, memory, and failures.

Card et al. (1980) provided the most detailed list of the weaknesses of GOMS. The weaknesses are as follows:

- The model is applied to skilled users, not to beginners or intermediates.
- The model does not account for either learning of the system or its recall after a period of disuse.
- Even skilled users occasionally make errors; however, the model does not account for errors.
- Within skilled behaviour, the model is explicit about elementary perceptual and motor components. The cognitive processes in skilled behaviour are treated in a less distinguished fashion.
- Mental workload is not addressed in the model.
- The model does not address functionality. That is the model does not address which tasks should be performed by the system. The model addresses only the usability of a task on a system.
- Users experience fatigue while using a system. The model does not address the amount and kind of fatigue.
- Individual differences among users are not accounted for in the model.
- Guidance in predicting whether users will judge the system to be either useful or satisfying or whether the system will be globally acceptable is not included in the model.
- How computer-supported work fits or misfits' office or organisational life is not addressed in the model.

Kieras (2002, p. 1154) have also presented some limitation of GOMS models. The most important restrictions are that GOMS cannot be used in modelling of procedural features of user interface, and model does not support task analysis. This method is quite demanding, because of the complexity of the method. First you have to make the model of the users' tasks and understand it, after that you can make a GOMS model.

2.3.3 Other methods

In this section it is presented some methods, which are not designed to use at a usability design or evaluation method, but these methods can be used as part of usability testing. These methods can be used to reduce the usability methods coverage of the production lifecycle.

Nielsen (1993, p. 207) refers to Diaper's (1989) studies when he mentions observation as a usability method; observation is useful both in the task analysis phase and in follow-up studies. Observing real users in a real environment initially reveals the tasks they must perform and suggests the needed features and functions for the product. In a follow-up phase, this can be used to validate that the product design was successfully implemented.

One of the simple methods of usability evaluation is a questionnaire; this method can find subjective user preferences and are easy to repeat. The tester has to guarantee that the user cannot misunderstand the questions and thus give false data. The questionnaire needs to be carefully formed and pilot tested. Nielsen says that interviews are a good way of finding out the opinion of the user about something in more depth than with questionnaires. Interviews are time consuming and hard to analyse. In addition, the data from one interview is not easily comparable with data from another interview. (Nielsen 1993, p. 209)

Below in Table 1 is listed the suitability of methods during different phases of the development lifecycle. This table is in addition to Table 1 on page 7. In this those methods, which support the task-based approach, are presented. The contextual design in its original form (Holzblatt & Beyer, 1993; Beyer & Holzblatt, 1998) does not have any kind of support to the implementing phase.

The latest addition of methods (Holzblatt & Beyer, 1999) mentions support, but does not have any instruction on it. According to one of the method originators (Card *et al.,* 1983) GOMS model is for predicting expert users' performance. These methods do not directly lead to identification of usability problems. Therefore, they are not considered as usability inspection methods in some context. (Virzi, 1997)

Table 1 Suitability of Usability Methods.

Name	**Task analysis**	**Designing**	**Implementing**	**Testing**	**Follow-up**
Contextual design	**X**	**X**	**(X)**[1]		
GOMS (all models)	**X**	**X**		**(X)**[2]	
Observations	**X**				**X**
Questionnaires	**X**				**X**
Interviews	**X**	**X**	**X**	**X**	**X**

As a conclusion of this subsection is that task centred usability methods as the methods that are described in subsection 2.2 have some common. None of they are good at if they are used alone. Every method has their own strengths and weaknesses, but combination of they can clear, simple enough, and effective. Some of the methods can be used in task-based approach and in user centred too, but some of them are clearly designed for handling users tasks.

[1] According originators of this method, this method does not support implementing.

[2] According one of this method originator, this is a model for predicting expert users' performance.

The conclusions of the chapter 2 is that more essential than consider the approach of usability is adapting and combining the right methods to helping developers to making more usable applications to users. As said in the first subsection, none of this things that are related on usability cannot make usability alone, neither none of methods or right process cannot make it. Usability of product is combination of knowledge, skill, and attitude.

3 CASE NEMU

This chapter describes main terms, methodological basics, and the framework of the research. At the beginning of this chapter there is descriptions of environment of this study after that there are descriptions of preparing, executing, and results of this research.

NEMU, Network Element Management Unit, is the common name for an additional network management system. NEMU offers extra value to telephone network operators by adding graphical tools to maintain and follow up the DX200/IPA2800 switching exchanges operations. More detailed description of NEMU is in section 3.2.

This chapter tries to find out the answers to the main questions of this study

- What is the level of usability in NEMU?
    - What are the best usability features of NEMU?
    - What are the most important usability features which should be improved in NEMU?
- What are the causes for prospective usability problems?
    - How could these problems be minimised in the future?

The customer of this study stated the first main question; whole meaning of this research is to find out the usability problems of the NEMU or to find out that there are no usability problems at all. If any problems are found then it is clear that the discovered problem should be resolved in some way. All of the identified usability problems should have at least a minimal improvement plan. This plan could be a clear order to correct the identified usability problem against the corresponding validation rule. Validation rules could be, for example, universal usability heuristics or in-house guidance or standards.

The second main question is only for case that there is one or more usability problems and orderer wants to prevent those problems reoccurrence by improving software production process. It is obvious that as all of the found usability problems are not related to software process then it's important to classify problems in certain categories by possible cause of emergence. Classification system tries to find out usability problems because of emergencies at least into the following classes: Technical domain system and used development environment, instruction and process usability competence and maturity. A decision of process improvement is made after this research but this paper tries to make it clearer in a field of usability.

Sub questions try to focus on the main questions to most important aspect of NEMU usability by making concrete founding and improvement suggestions.

3.1 Telephone Network

A switching exchange is the most important part of a telephone network which main function is to make a link between two telephone users. This link is a connection between a subscriber and a receiver. A switching exchange also has other quite important functions that relate to a call such as signal transmission (free/busy line), reception of the dialling, number analysis, connection of the call, monitoring the call, clearing the call, collecting statistics and collecting charging data. If a call is local there is only one switching exchange but if the desired call is a long distance call there could be several switching exchanges involved in making the call available. Every telephone network needs at least one switching exchange but the number of switching exchanges depends on the number of users and the size of the exchanges that is measured, for example, by the accepted number of connections. A big city would have hundreds of switching exchanges.

Traditional switching exchanges offer switching services in fixed networks but nowadays mobile and different kind of data communication services are the most important usage of switching exchanges.

One type of switching exchange is the Nokia DX200. This name is used as a general label for Nokia network elements that are used in a data communication network. Development of the DX 200 began in the 1970's and the first products were connected in fixed telephone networks in 1980. Nowadays DX200s are used in several types of networks such as mobile switching centres (MSC), home location registers (HLR), base station controllers (BSC) and still in fixed telephone network switching centres (FSC). MSCs are used as a centre in all kinds of mobile telecommunication. HLR is the mobile networks' user subscriber location database and BSC is the mobile network base station-controlling unit. Each of those products is based on the same DX200 hardware. All the necessary functionality of DX200 is made in software (Silander 1999).

DX200 is in principle a general-purpose computer; only the software makes any difference when compared to other computer systems. This system also has switching exchange typical parts, which these are needed only in fixed telephone networks. The whole system is built with standard components, which means for instance Intel's 32-bit standard processors. This makes possible an increased efficiency of DX exchanges as soon as new processors are coming to the market.

From an architectural viewpoint DX200 exchanges are a scalable, efficient, loosely connected multiprocessor system with high level of fault tolerance. Scalability and efficiency mean that the capacity of exchanges can be easily changed depending on local needs. A loosely connected multiprocessor system means that one DX200 exchange could have several physical computers which are connected together and each of these computers is specific in its own task. These separate computer units communicate together via signal channels. Fault tolerance means that if some part of system stops nevertheless other parts are still going.

Layer architecture guarantees an easily modified system. The DX exchange system consists of three different layers. The two lower layers of DX, computing platform and switching platform, build a fault tolerant system platform. The upper, the third, layer is the application layer that gives all of services such as call control and subscriber services. Not all DX exchange systems need all of these three layers, for example, HLR (home location register) needs only the computing and application layers.

3.2 NEMU

NEMU, Network Element Management Unit, is the common name for an additional network management system. This system is separate but cannot work without the DX200/IPA2800 switching exchange. The NEMU system is composed of client computers, a server computer platform and its operating system and a specific collection of applications and server services. NEMU offers extra value to telephone network operators by adding graphical tools to maintain and follow up the switching exchanges operations. More detailed information about NEMU is in Appendix 2.

3.3 Starting of the Case NEMU

The first look up for the NEMU developing process was quite poor in a usability fields. This section of development does not have any plan for usability or human-computer interaction that supports to development. There were not clear and simple commonly used rules to build usable applications, or these rules were quite difficult to adapt for NEMU development environment and its needs. Developers do not have any training in usability. This lack of knowledge relate also testing crew not only designers and programmers. Some developers' attitude was quite negative for usability. This might be caused by the lack of correct information. On the other hand, because the NEMU project was quite new, the people have to concentrate on developing the product and they do not have extra time to think nothing else. During this study the most of this section crew got basic information of usability on a training day, some of them got usability teaching twice, because they were participants of a usability evaluations.

3.3.1 Designing of Testing

In this case, choosing the applications to evaluate is quite easy; all of the available applications have to be tested. The low number of possible applications also caused that easiness; there were only eight separate applications to be tested. Some of those applications were so simple that they have only one window and a few user controlled functions.

Two of the tested applications were not ready when tests started, this did not cause any problems in testing. Just before the tests started, two more applications that had to be also tested, were found. Another one was small utility tool, which is designed to use very rarely and the other was a new concept of application platforms.

Some of the applications set some restrictions to evaluation methods, and some of applications are so important that they might be tested using as many methods as it is possible. In this particular case there were planned to use visual walk through, heuristic evaluation, and cognitive walk through as testing methods. Ravden & Johnson method is designed to act as a comparison material to heuristic evaluation. Consistency evaluation has been already used as usability evaluation method. The most essential applications to be tested all of those methods and others at least visual walk through, heuristic, consistency and evaluation.

Test cases must walk through end user's normal workflow and its exceptions. It also has to go through all of the requirements of the system. In this case all of the test cases were assumed to be as in the everyday life of an operator. There were included only few exceptions in cases that it might be possible to find some more problems. Designing those cases was quite hard; some designers had to be interviewed, as well as the test crew, and some other persons who are familiar of telephone network operators' normal workflow. Even that was not enough in practice. The more reliable test there would have been interviews and observation of real end users instead of developers, but within resources of this study this was not possible.

Gathering of the testees was the most difficult phase of the preparing work. If you send email it's a miracle if some of invented respond to your mail. If you ask face-to-face they commit, but maybe next day they send an email or an SMS in which they say that they cannot attend because they are so busy. In this case it was so positively surprising that most of the developing team members wanted to participate these test, but this voluntariness does not be so good in other sections.

To gather all needed testees you have to use all relationships and competence that you ever had. One quite easy way to recruiting of testees is use the “power of manager”. That kind of recruiting decreases motivation and weakens results, so for using this method has to carefully. But in this case that method has tremendous success. In NEMU case there were needed 24 testees on six separate evaluations, two groups for Ravden &Johnson method and four groups for heuristic evaluation. Cognitive walkthrough needs only a few volunteer testees.

The aim was gather those 24 test person plus at least six reserve person in evaluation meetings. This aim came almost true; all the test persons and one reserve person were found. Cognitive walkthrough needs also some testees, but requirements of these testees were quite hard, thus there were only few possible candidates. There were only two of three volunteers who could take part in this test. That was not enough but it was better than nothing. And the last group that needed in pilot test was quite easily formed; they are all members of the same student project group. Testing persons that was needed in consistency evaluation was colleted by another person and I was only on testee of this group.

The most essential part of the testing preparation phase is the reservation and preparation of testing rooms. These rooms must contain appropriate equipments and enough seats to test persons. Also one thing has to be ensured that the room is separated from its environment. Separated testing room ensures that conversations between test administrative and testees can be without distractions and confidential and separation reduces the noises and interruptions from the environment. Right equipment means that there are working computers; data projectors, silver screen, and other needed devices such as tape recorders and/or video cameras and a tester has trained to use them.

When you have found all testees and you are reserved all needed rooms you can make a final schedule of tests. Who is where and when? In this phase you have to combine all wishes and restrictions of test person timetables. All of the testees have proposed two dates and time when they can participate. After few email conversations every test had all its needed testees. This testing session was planned to be five days long, one day there have to be two separate groups.

Pilot Tests, Case *Application K*

Before the actual usability evaluation of NEMU applications there was made a pilot test with Jyväskylä Polytechnic student project members in their application. The project group had produced a NEMU *application K*. For more details of *application K*, see Appendix 3 NEMU Applications.

Not all of the application features were ready or they worked improperly, in other words, the application was not finalized. This pilot testing tried to find out the usability problems of *application K* and at the same time evaluate and check usability testing materials such as test cases and test forms. These tests succeeded, usability problems of the application K were found and testees found a few errors in test materials. This usability testing made essential improvements to developing the *application K*, because all of its features were not implemented at that stage. This test gave some advice to developers and at the same time gave some support to the developers that their application was not so bad at all. Redesign work was not going to be too extensive, because some parts of the evaluations were made only as a paper test. This means that there were no implementations, thus developers could make a new, improved version of their application without wasting previously done work. More precise results of the *application K* usability tests are in Appendix 5 Pilot Usability Tests Results.

At first there were planned to be visual walkthrough and heuristic evaluation in all tests. Both of those usability-testing methods give quite same results. After this pilot test it was clear that only one of the methods was needed in the further tests, and the heuristic evaluation was selected. Heuristic evaluation has chosen because it is easier to learn and use than visual walkthrough and heuristic evaluation have clearer instructions to execution.

3.3.2 Preparation of Testing

Before the test session you have to ensure that you have right and working equipment and you can use it. One important thing that you must check is that testees have pencils and papers to make their own notes. There should be a needed amount of all forms and instructions.

You need to have the latest or the certain versions of the applications on the test computer. If you are going to test paper mock-ups, you have to assure that all pictures are in the right order. One essential thing that must be done is removing all distractions away from the testing room, for example, too loud clock.

3.3.3 Testing Situation

First of all the testing situation should be made relaxed and confidential for the testees. Tester or testers must introduce themselves and the purpose of meeting. Next, the basics and background of methods and tested applications must be told. There have to remember that a tester does not introduce too detailed application that they do not affect the testees' opinion. One essential thing that has to emphasize to testees that all evaluations are extremely confidential that means all said or wrote is reach only very limited group of test administrative. Nobody even testee's manager cannot get actual comments of evaluation meeting. Normally all material should be handled anonymously and all comments that could reveal the testee would be changed into unrecognisable form before evaluations results are published. One important thing that has to be told is that there would not be evaluated testees ability or knowledge in any level and every testee is free to leave during the evaluation. Meetings can be recorded or videoed only by explicit permission of testees. When all starting procedures are handled then actual evaluation meeting begin.

This described procedure was used in visual walkthrough, heuristic evaluations and Ravden & Johnson methods. Testees of cognitive walkthroughs made their evaluations on they own workplaces and on their own schedule. Consistency inspection meeting was more like pluralistic walkthrough. Secretary, who made notes about testees' observations, also directed meeting.

3.4 Results of Usability Testing

This chapter takes a cursory view of usability tests results of this research. Here is only a short summary of the results of the usability testing, because of confidentiality reasons. Closer results are in appendixes. These results are divided into own sub chapters by evaluation methods. All of the used methods are presented in their own sub section. More detailed collection of the results of the usability tests can be found in *Appendix 5 Pilot Usability Tests Results.*

3.4.1 Heuristic Evaluation

Heuristic evaluation was the main usability evaluation method in this research. There are three main reasons to use this method. First Heuristic evaluation is easy to learn and use. Second this method is quite cost efficient. That means that using this method needs only three to five testees and this method returns approximately 75% coverage of all systems usability problems. The last reason is the easy learning of this method this is good point for all testees especially for developers in professional meaning. After learning developers can use this method in a more informal way, they recognised the places of possible errors during their developing work.

Evaluation meetings were arranged by collecting a team, this team includes one developer, one person from testing, and two persons, who have different background than other two testees. These non-developing people must have good skills in computers and another must have some kind of knowledge of switching exchange environments and/or operator work. This last requirement means that all of those four testees are people from documentation. Testee who has not any domain knowledge and who is not member of development team is usually a project management person from another section or assistance crew of development section.

Evaluation environment was a normal meeting room that had one laptop computer, data projector, and silver screen. Every testee had their own set of test cases, see *Appendix 4 Usability Test Cases,* and test forms, see *Appendix 1 Used Usability Test Forms,* and short instruction of evaluation method in paper form.

Procedure of evaluation meeting was quite simple. The meeting started with introduction part which content was short briefing of the method and application or applications that are evaluated, and the instructions were told to execute the evaluation.

Next there is described short brief of heuristic evaluation results. More detailed results of heuristic evaluations are in *Appendix 7 Heuristic evaluation results by applications.*

The first team tested five small *applications B, C, F, H*, and *I*, see *Appendix 3 NEMU Applications*. Totally this method found 24 different discoveries on these applications. Only 15 of found errors are in level three on Nielsen's scale, and three out of 15 errors were so severe that more than two testees noticed them at the same time. Findings are mainly related on functionality of the system more than, for example, layout of dialogues. Because of size of test group there might be about 6 more problems that this test did not found.

The second team test *applications A,* and *E*, see *Appendix 3 NEMU Applications. Application E* cannot be evaluated as other applications, because of its structure and features. That application was tested with using applied heuristic evaluation method. Testees found on *application E* a few problems and suggested some improvements. *Application A* is normal application and its evaluation has not any divergences. Testees found totally 20 problems in which more than two out of three was level 2 and the rest were in level 1. Problems related mainly in window or dialogue layout and the lack of shortcuts. Testees overall estimation of application A was quite positive.

Team three made usability test in *application J*, see *Appendix 3 NEMU Applications.* The tests consists evaluations of the main window and three dialogues of it. This application gets totally 24 estimations the most of these estimations are in level one. Only four of all estimations are in level 3 and two of those evaluations are unanimous enough. The found problems mostly related on missing texts and ambiguities of functions. Estimated dialogues got the poorest estimation on the lack of the help and too complex design. This application was not finalized during the tests; this was the most essential reason of problems of this application.

Team four made tests for the *applications G*, see *Appendix 3 NEMU Applications.* Total number of problems was 30, but third of problems were in level 1. This application was the first of NEMU applications in this section and during implementation phase there was not properly working version of Platypus available. All of the found problems relate on the layout or the help system of the application. Testees do not found anything comment on functionality of the application.

Application D was tested in meetings of group two, three, and four. The basic aim of this evaluation was check features of the new concept and possible bottlenecks of it. This application was in pre-design phase, and it has only a few working functions. Applied testing method is combination of heuristic evaluation, visual walkthrough, and pluralistic walkthrough. In the testing meeting, tester guided testees through application. Testees answer to questions and they tried to find their own suggestions to developing. More than a half of the testees estimated the application D was so good that it could be develop as a new product. Some of the testees doubted about essence of the whole application. Developers of *application D* got own report of evaluations. In *Appendix 7 Heuristic evaluation results by applications* is more about the results.

3.4.2 Cognitive Walkthrough

Cognitive walkthrough was planned to execute by only a few testees. Finally two suitable testees made it. One was from NEMU testing crew and the other was usability skilled testee, but not from NEMU development personnel. Those two testees made this test on their own. Both testees got the same instructions, forms and scenarios but different applications. Tested applications were A, B, C, F, H, and J. See details in *Appendix 3 NEMU Applications*. Cognitive walkthrough test forms are in *Appendix 1 Used Usability Test Forms* and test cases are in *Appendix 4 Usability Test Cases.*

These tests gave a quite low number of notices, because of low number of testees. That does not mean that tests were unsuccessful. Testees found some interesting points that can affect on usability of the whole system, for example, application B does not have refresh feature. In other tests its absence was not noticed at all, but the meaning of this feature is quite essential for the task of the end user.

Testees rated their findings by using a scale: how many percent of users notice that problem. All meaningful findings have to have at least 25 percent estimated probability. Reliability of these results are quite low, thus these findings have to be treated like testee's point of view more than great scientific truth. I do not underestimate testees skills in any level and their contribution for this study. More about the results of cognitive walkthrough are in *Appendix 8 The Results of Cognitive walkthrough.*

3.4.3 Consistency Inspection

This usability evaluation was not designed and organised due to this study, it is a part of normal development routines. In this meeting all existing NEMU applications styles, layout and consistency were evaluated. The results of this meeting were quite shattering. There were 13 applications in compared, and none of these applications were in the same visual or naming style. For example there were 12 different about boxes, one application even had them all. Applications had differences in their menus. There were two or more words in the same meaning in several cases.

One reason to this kind of results might be the problems with Platypus. When the NEMU development work started there was not Platypus like utilities at all, and Platypus developing work took its own time. More precise results of consistency evaluation found in *Appendix 6 The Results of The Consistency Inspection Meeting.*

3.4.4 Ravden & Johnson Method

This method was used only as a reference and it was used only with two separate application tests. Applications that are evaluated in this method are *application C* and *J,* see *Appendix 3 NEMU Applications*. The results of this method were quite same than heuristic evaluation as assumed. For example, application C has 33 different notices. In heuristic evaluation has been found 24 different notices and R&J method set of 25 hits. Almost half of the found problems were found by both methods at the same time. The most of the R&J methods discoveries are related to styles and layout, meanwhile problems that were found by heuristic evaluations are mostly functionality aspects.

The cross-reference of the findings of both methods is in *Appendix 9 Cross-reference heuristic evaluation vs. Ravden & Johnson method.*

Using this method takes clearly more resources than heuristic evaluation. That could be a consequence of difference of testing material, in heuristic evaluation testee makes choices by ticking to the box, but in this method testees have to write down some opinions and discoveries. Making own notices can produce more fertile thoughts to improving the system than bare check marks.

3.4.5 Visual Walkthrough

This usability evaluation method was designed to perform a part of all evaluation meetings. This method is used only in pilot tests because this method is quite time consuming and results are mainly the same than for example results in heuristic evaluation. This first and only evaluation was done with Jyväskylä polytechnic project course group. In this meeting was evaluated *application K*, see *Appendix 3 NEMU Applications*. One important notice was that the application wasn't ready when it was tested and some parts of evaluations were made in paper mock-ups of designed system. Mock-up test was quite easy to arrange. In this case it succeeded and offered several changes and corrections to further development work.

In general, tested application was good and it does not need remarkable corrections, but some features that relate on layout needs to be fixed. There are some problems with readability and finding the information, those may cause by too dense layout. The results of this test need some reservation, because the same persons who have made and evaluated it and test materials were not final form. Also the number of testees was quite low and test results might not be objective. More precise results of the visual walkthrough are in *Appendix 5 Pilot Usability Tests Results.*

3.4.6 Conclusion of the Results

Testing situations succeeded, in other words there ware not bigger problems. Some minor problems happened; a tape recorders cassette ran out once and a tape recorder battery dried out once. Once there was a wrong version of tested application in the test server, changing the server cleared out that situation. Once other testing crew freezes the whole server at the middle of usability test. The most of those problems could be forecast but, for example, totally system crash down is not predictable. In this case there were several servers, which could replace the server that have problems.

Every used method ware found at least some usability problems. All the methods gave results, which are typical for the method. Typically results were, for example, observations of strange or absent features in cognitive walkthrough or incoherent about boxes that was found in consistency inspection. All of the used methods gave the same results, such as missing shortcuts or wrong titles of dialogues. Differences between cost efficiency of methods were explicit, for example, visual walkthrough takes two hours for the *application K* and the same application tested in an hour with heuristic evaluation, but the results were quite the same. If the testees' suggestions of improvement are essential, then should use methods, which support that action. In this case there were used methods that were applied to collecting proposals and suggestions to developing. In all meeting alike in heuristic evaluation meetings this was put into practice as in open conversation at the end of the meeting.

Some changes have to been made make before the next test. All meetings need at least one person more that person should be a representative of end users. Test cases have to be collected with real end users of they real working environments or they have to even comment it. All materials and equipments need to be double checked before actual test event.

Testing session was quite exhausting for tester, thus it is not realistic to arrange more than two tests that takes more that two hours each in a day. Depending on evaluation it might not be reasonable use same testees more than a one test in a day.

Overall feeling about NEMU usability testing is quite satisfying. Some things succeeded and some did not, but every minute of the tests was eminently edifying. One important notice was that all of the testees said after the test meeting that they have learnt something about the usability of software. They might be having been polite to me.

3.5 User Feedback

Because some of the applications have been used about year and half in real environment with real end users, there has been received user feedback that relates on NEMU applications. Number of feedback is quite small. This is probably a sign of the lack of documentation of feedback instead of lack of user feedback. Handling the user feedback as normal errors, such as Prontos, is essential to development process.

These user feedbacks which have been reported can be divided into two categories, possible design flaws and problems with help and documentation. The total number of reported user feedback is 11, but there are two separate problems that both have been reported twice by different organisation. Five out of 11 reported NEMU prontos have third level severity of the problem classification. Four of them are some kind of design or implementing problems and one of them is clearly lack of documentation. Other six prontos are marked on level two on severity category; all of them are related on some kind of lack of help or documentation. More precise description of the user feedback is in

Appendix 10 User feedback.

As a conclusion of the user feedback can be said that there is competent developing and testing crew, or customers are too lazy to make any complains on their NEMU applications. The latter reason does not seem so likely. If there is no help and documentation Prontos at all then total number of Prontos might be less than a half, in the future there have to make some improvements in the help and documentation.

3.6 Other discoveries

In informal discussions with development team members, before and after the usability tests, I made some discoveries, which could be a possible reason, which can lead to poor usability. Frequently mentioned cause was the lack of clear and unambiguous guidance. For example, BTS SW GUI Checklist is 60 pages or Platypus 3.0 Designers Guide is 219 pages and these are not the only ones. According to developer's unclear guidance it means, for example, priority of documents. Different documents can describe the same thing but in a different way, in this particular case it is important that a developer knows the order of the documents. Developers say that Platypus is so ambiguous that even its authors cannot understand it. This might be true; because some part of it was so detailed that a normal application developer does not need all of its information. Another problem of Platypus is that it is constantly changing. If some part of an application was made under old Platypus version and newer version of Platypus was released then previously made application was not compatible with Platypus or visa versa. A developer feels that the lack of good guidance frustrates and it also causes delays to the work. To prevent delays developers try to make some own solutions and inventions to solve problems. That kind of activity consumes resources of real development and can be the cause of unnecessary errors, and incompatibility and incoherence between applications. Those unnecessary errors can affect the moods of development team and it can reduce hostility against usability, if those errors are found during the usability tests.

Several developers brought in some suggestions, which can be helpful. First one is some kind of design patterns. Developers say, for example, that they need simple models that can make some trivial basic components such as a table. One of the suggested helping hands is a simple model example application or some applications with their commented codes. Those applications can be included in the most essential features of desired system, such as information, input and output dialogs. There can be implemented also some seldom used features, like directory tree listing and special instance of windows. The second suggestion to help every day work is “Hall of Shame”. If somebody has made a terrible blunder, there is no need to make it again.

Developers mentioned also reasons that cause consistency problems. There are no exact definitions, what document is primary, and what the latest document is. They also emphasise the most important reason that leads poor outcome, which is that there is any kind of common guidance or design rules available when development starts with respect to usability.

4 Actions Toward Better Applications

If the software have some feature that user can't use it's the same the software haven't that feature at all. Allan Cooper

In this chapter I have collected a few actions towards better applications. These can be considered as results of this research. These actions are the most cost efficient and simple ways to introduce usability into a software development process.

Detecting and identifying users and their work is the first step to improve a software product. After identification can be done Jacobson's *use cases* of desired system. The use case can be used as a basis of task analysis. The well-defined use cases help a designer to concentrate his attention on the most essential or desired targets on early stage. More essential is to set focus on primary functions of the application than nice little tricky features that are also important for the whole system. Only the knowledge of essential user's tasks can help this phase. The user interface must satisfy three different paradigms at the same time: the technology, metaphor, and idiomatic paradigm. The first one is based on understanding how things work. The second paradigm is based on intuiting how things should work, and the last one, idiomatic paradigm, is based on learning how accomplish things (Cooper, 1995, p.54).

An important thing is to collect usability data during throughout the whole software development lifecycle. In every phase of the development process come out ideas and happen improvements. That means that when developers make new inventions for the advanced applications those ideas should be collected into one common database. That database can include all good and bad inventions with explanations, instructions, and codes. Testing collects its own notes and hopefully designers take advantage of them. Into this category belong collecting user feedback and its analysis. It might be the most important source of usability data. "*If achieving the users' goals is the basis of our user interface design, then the user will be satisfied and happy. If the user is happy, he will gladly pay us money, and then we will be successful.*" (Cooper, 1995, p. 11) As Cooper says, if your product is good, you can be glorious.

It’s important that user documentation is built at the same time as the related software; then documentation testing and verification can be done at the same time with software testing. Interaction between developing and documenting has to be smooth and continual. This communication helps a documentation crew to understand all nuances of features of the application.

Learn about your own and other developer’s errors. Some developers say in interviews that they need some kind of *hall of fame* and *shame* to help the design process. Designers do not need to reinvent the wheel.

Another simple action towards better applications is to add a usability person in to a project. This means that somebody from a development crew or someone outsider has to be named as a usability “coordinator” of the section. The person keeps eye on changes of in-house usability instructions and delivers information to all other developers. A direct contact to the testing crew is one essential task that this usability person has to do. As Karat (1997, p. 35) argues: simply renaming a job title can help usability to establish its position in the organisation, brings down futile fences and opens new possibilities to improve the development process. He mentions, for example, his own experience of renaming his title from ”human factor specialist” to ”usability engineer”, which opened the access to different development teams.

5 DISCUSSION

This research finds some answers to questions on the case NEMU. The first main question was "*What is the level of usability in NEMU?*" Overall usability of NEMU applications was quite high; there were some deviations in both directions. But there were a great number more good than poor things. The result of usability evaluation is better than assumed at the beginning of this work. Some problems were so in the foreground that they gave a poor first impression of the whole system. A closer look up showed that those problems were minor in their impacts and they were easily fixed.

The first sub questions of the primary question were "*What are the best usability features of NEMU*?" and "*What are the most important usability features which should be improved in NEMU?*" The best aspect of the whole NEMU application family is that all applications are uniform in their layout and consistent in their usage, even if the consistency inspection results are not so good. All remarkable found problems have been fixed. Documentation is not yet as it has to be, that is clearly the poorest feature of NEMU user feedback can easily confirm that argument.

The second main question, to which this research tries to find an answer, was "*What are the causes for usability problems?*" and its sub question was "*How these problems could be minimised in the future?*" Answers to both questions are not ambiguous. Some reasons for the first question might be the age of the section and scattered organisation. Young age means that there is not a stable way to work yet, but this can also seen as a strength of development work. Scattered organisation causes needless underestimation of other parts of organisation, and it might cause lack of information, because of exiguous interaction. Answers of the second questions are not clear, but in this case increasing the level of the knowledge of usability might bring back of its investments. Supportive work of development and usability actions and methods may help developers' everyday work, for example, code or sample library for rarely used functions and dialogue layouts.

The results of this research have some restrictions. All the testees were some kind of professionals, but not in the domain of the target system. This means that the results might be different if real end users had made their own estimation. Other essential note is that NEMU applications are not designed for normal consumers, such as word processors are. Users of NEMU applications are experts of their own domain and applications have to be adapted for this purpose.

Almost all of the user feedback items that were received in the Pronto system have been handled as the result of usability evaluations. They have already been used or going to be used in development work. Some Prontos, which are in the lowest level, might be used in future.

After this research there are still things that have to be studied. The first one, which needs some verification is the effect of a task-centred approach to NEMU applications like cases. Are there any good reasons why it would be better or wore then a use user-centred approach? Other level of the first question could be whether there are any other approaches which support development work better than the currently used one. This was one of the original planned research questions, but it did not suit the final schedule.

One quite interesting field of study might be the problem classification model and developing a “solved problems” database to support NEMU development. When there are quite a large number of performed usability tests and with they a larger number of problems have been found with them, control and exploitation of these become quite problematic. With a simple and efficient classification database one can easily made queries and find solutions when new problems appear.

6 Bibliography

Bevan, N., 1995. Measuring Usability as Quality of Use. Journal of Software Quality, 4, pp. 115-130

Beyer, H., & Holtzblatt, K., 1998. Contextual Design: Defining Customer-Centered Systems, San Francisco: Morgan Kaufmann Publishers

Bias, R., 1994. The Pluralistic Usability Walkthrough: Coordinated Empathies. In J. Nielsen & R. Mack (Eds.) Usability Inspection Methods, John Wiley & Sons Inc., pp. 63-76

Brooks, F., 1975, The Mythical Man-Month, Reading: Addison-Wesley

Boehm, B., 1988. A Spiral Model of Software Development and Enhancement. IEEE, Computer. 21(5), pp. 61-72

Card, S., Moran, T. & Newell, A., 1980. The Keystroke-Level Model for User Performance Time With Interactive System, Communications of the ACM, 23(7), pp. 396-410

Card, S., Moran, T. & Newell, A., 1983. The Psychology of Human-Computer Interaction, Hillsdale: Lawrence Erlbaum Associates

Cooper, A.,1995. About Face: The Essentials of User Interface Design, Foster City: IDG books Worldwide Inc.

Dehlholm, F., 1992. Picture Analysis of Screen Images. In Leponiemi, J. (Ed.), NordDATA'92 Precedings, pp. 353-359

Diaper, D., 1989. Task Observation for Human-Computer Interaction. In D. Diaper (Ed.), Task Analysis for Human-Computer Interaction, Chichester: Ellis Horwood, pp. 210-237

Dicks, S., 2002. Mis-usability: On the Uses and Misuses of Usability Testing, In Proceedings of the 20th Annual International Conference on Computer Documentation, ACM Press, October 20-23, pp. 26-30

Dumas, J., & Redish, J., 1993. A Practical Guide to Usability Testing, Norwood: Ablex

FOLDOC, The Free On-line Dictionary of Computing, <http://foldoc.doc.ic.ac.uk/foldoc/index.html>. Checked 26.08.2003

Gould, J. & Lewis, C., 1985. Design for Usability – Key Principles and What Designers Think. In Proceedings of the ACM CHI'83 Conference on Human Factors in Computing Systems, pp. 99-106

Gould, J., Boies, S. & Lewis, C., 1991. Making Usable, Useful, Productivity-Enhancing Computer Applications. Communications of the ACM, 34(1), pp. 75-85

Gray, W., John, B. & Atwood, M., 1993. Project Enrnestie: A Validation of GOMS for Prediction and Explanation of Real-World Task Performance, Human-Computer Interaction, pp. 237-309

Gulliksen J., 1996. Designing for Usability – Domain Specific Human-Computer Interface in Working Life, Uppsala: Acta Universitatis Upsaliensis

Haikala, I. & Märijärvi, J., 1998. Ohjelmistotuotanto, Jyväskylä: Satku

Holtzblatt, K., & Beyer, H., 1993. Making Customer-Centered Design Work for Teams. Communications of the ACM, 36, 10, pp. 92-103

Holtzblatt, K. & Beyer, H., 1999. Contextual Design, Interactions, 6, ACM, pp. 32-42

ISO/DIS 9241-11:1999, Ergonomic Requirements for Office Work With Visual Display Terminals (VDTs) – Part 11: Guidance to Usability

ISO/FDIS 13407:1999, Human-Centred Design Process for Interactive Systems.

John, B. & Kieras, D., 1994. The GOMS Family of Analysis Techniques: Tools for Design and Evaluation, Technical Report, Carnegie Mellon University

Kaner, C., Falk, J. & Nquyen, H. Q., 1999. Testing Computer Software, second edition, John Wiley & Sons

Karat, T., 1997. User-Centered Software Evaluation Methods, In M. Helander, T.K. Laudauer & P. Prabhu (Eds.), Handbook of Human-Computer Interaction, Elsevier Science, pp. 689-704

Kieras, D., 1988. Towards a Practical GOMS Model Methodology for User Interface Design. In M. Helander ed., Handbook of Human-Computer Interaction, Elsevier, pp. 135-157

Kieras, D., 1997. A Guide to GOMS Model Usability Evaluation Using NGOMSL. In M. Helander, T.K. Laudauer & P. Prabhu (Eds.), Handbook of Human-Computer Interaction, Elsevier Science, pp. 733-766

Kieras, D., 2002. Model-Based Evaluation. In J. Jacko & A. Sears, (Eds.), The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications, Lawrence Erlbaum Associates, pp. 1139-1151

Lewis, C., Polson, P., Wharton, C. & Reiman, J., 1990. Testing a Walkthrough Methodology for Theory-Based Design of Walk-up-and-Use Interface, In Proceedings of the ACM CHI'90 Conference on Human Factors in Computing Systems, pp. 235-242

Lewis, C. & Wharton, C., 1997. Cognitive Walkthroughs. In M. Helander, T.K. Laudauer & P. Prabhu (Eds.), Handbook of Human-Computer Interaction, Elsevier Science, pp.717-732

Life, A., 1990. Simulation and the User Interface, London: Taylor & Francis

Mayhew, D., 1999. The Usability Engineering Lifecycle: A Practitioner's Handbook for User Interface Design, San Francisco: Morgan Kaufmann Publishers

Molich, R. & Nielsen, J., 1990, Improving a Human-Computer Dialogue. Communications of The ACM, Vol 33, no. 3, pp 338-348

Musa J., 1998. Software Reliability Engineering, McGraw-Hill

Muller, M. & McClard, A., 1995. Validating an Extension to Participatory Heuristic Evaluation: Quality of Work and Quality of Work Life. CHI 95 Conference Companion, pp. 115-116

Muller, M., Matheson, L., Page, C. & Gallup, R., 1998. Participatory Heuristic Evaluation. In Interactions, Vol 5, no. 5, pp 13-18

Myers, G., 1976. The Art of Software Testing, New York: John Wiley & Sons Inc.

Nielsen J., 1993. Usability Engineering, Academic Press

Nielsen, J. & Mack, R., 1994a. Usability Inspection Methods, New York: John Wiley and Sons

Nielsen, J, 1994b. Enhancing the Explanatory Power of Usability Heuristics. In Proceedings of the ACM CHI'94 Conference on Human Factors in Computing Systems, pp.152-158

Nieminen, M. & Koivunen, M., 1995. Visual Walkthrough. In G. Allen, J. Wilkinson & P. Wrigth, (Eds.), HCI'95, People and Computers, Adjunct Proceedings, The University of Huddersfield, The School of Computing & Mathematics, pp. 86-89

Nieminen M., Riihiaho S., Sinkkonen I. & Parkkinen J., 1997. Käytettävyysopas. Ohjelmiston käytettävyyden arviointi –seminaari, Tieturi

Norman, D.A., 1986. Cognitive Engineering, In D. Norman & S. Draper (Eds.) User Centered System Design: New Perspective on Human-Computer Interaction, Lawrence Erlbaum Associates

Norros L, 2003. Understanding Use in the Design of Smart Objects – Reflections on the Conception of Collaborative Design, Proceedings of Smart Objects Conference

Polson, P., Lewis, C., Rieman, J. & Wharton, C., 1992. Cognitive Walkthroughs: A Method for Theory-Based Evaluation of User Interfaces. International Journal of Man-Machine Studies, 36, pp. 741-773

Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S. & Carey, T., 1994. Human-Computer Interaction, Addison-Wesley Publishing Company

Pressman R., S., 1997. Software Engineering: A Practitioner's Approach, Fourth Edition, McGraw-Hill

Ravden, S. & Johnson, G., 1989. Evaluating Usability of Human-Computer Interfaces: A Practical Method, Chichester: Ellis Horwood Limited

Rowley, D. & Rhoades, D., 1992. The Cognitive Jogthrough: A Fast-Paced User Interface Evaluation Procedure. Proceedings of the ACM Conference on Human Factors in Computing Systems, pp. 389-395

Rubin, J., 1994. Handbook of Usability Testing, New York: Wiley

Silander, S., 1999. DX Primer: Introduction to DX200 Software Engineering, Helsinki: Edita

Sinkkonen, I., Kuoppala, H., Parkkinen, J. & Vastamäki, R., 2002. Käytettävyyden psykologia, Helsinki: IT Press

Shneiderman, B., 1998. Designing the User Interface: Strategies for Effective Human-Computer Interaction. 3rd ed., Reading: Addison-Wesley

Strong G., 1994. New Directions in Human-Computer Interaction Education, Research, and Practice, <http://www.sei.cmu.edu/community/hci/directions>, Checked 26.06.2003

Tukiainen, M. & Lempinen, R., Raportti B-1994-1, 1994. Joensuun yliopisto raporttisarja B, Also available in ftp-site: <ftp://ftp.cs.joensuu.fi/pub/Reports/B-1994-1.ps>, Checked 29.08.2003

Virzi, R., 1997. Usability Inspection Methods. In M. Helander, T.K. Landauer & P. Prabhu (Eds.) Handbook of Human-Computer Interaction, 2nd ed., Elsevier Science, pp.705-715

Wharton, C., Rieman, J., Lewis, C. & Polson, P., 1994. The Cognitive Walkthrough Method: A Practitioner's Guide. In J. Nielsen, & R. Mack (Eds.), Usability Inspection Methods, New York: John Wiley and Sons

Wilson, C. & Coyne, 2001. K.P. (Eds.), Tracking Usability Issues: to Bug or Not to Bug? Interactions, Vol. 8, 3, pp. 15-19

Wixon, D., Jones, S. Tse, L. & Casady, G., 1994. Inspections and Design Reviews: Framework, History, and Reflection, In J. Nielsen, & R. Mack (Eds.), Usability Inspection Methods, New York: John Wiley and Sons

Young, J., 1994. W., A Technology for Producing Ideas, NTC Business Books

7 APPENDIX

7.1 Appendix 1 Used Usability Test Forms

A Heuristic evaluation test form

Name of the application	Severity				
Heuristic	0	1	2	3	4
Feedback					
Speak the users' language					
Clearly marked exits and user control					
Consistency and standards					
Prevent errors					
Minimise user memory load					
Shortcuts, flexibility					
Use simple and natural dialogue					

Heuristic checklist

Yleistä

Seuraavassa on tutkimuksessa käytettävä heuristisen arvioinnin muistilista. Alla oleva kymmenkohtainen muistilista on sovellettu Jakob Nielsenin heuristisen arvioinnin muistilistasta (Nielsen 1993).

Listaa on täydennetty esimerkki kysymyksillä, joiden tarkoituksena on valaista kutakin muistilistan kohtaa, joskaan ei antaa tyhjentävää kuvausta sen tarkoituksesta. Usein muistilistan jokainen otsikko on riittävä kimmoke mahdollisen ongelman tunnistamiseen ja luokitteluun.

1. Ohjelman tilan näkyvyys

Käyttäjän pitäisi aina pystyä nopeasti huomaamaan mikä on ohjelman tila ja käyttäjän sijainti ohjelmassa.

- Onko ohjelma pystyssä (ts. toimiiko se vielä)?
- Onko ohjelma vastaanottanut syötteeni?
- Mitä seuraavaksi on tapahtumassa?
- Missä päin ohjelmaa olen?
- Minne voin mennä seuraavaksi?
- Olenko menossa oikeaan (haluamaani) suuntaan?

2. Ohjelman ja tosielämän vastaavuus

Ohjelmassa pitäisi käyttää normaalissa prosessiteollisuudessa tuttuja termejä, sanontoja ja käsitteitä.

- Ovatko sanat ja lauserakenteet helposti ymmärrettävissä?
- Käytetäänkö käsitteitä samassa merkityksessä kuin tosielämässä?
- Toimivatko metaforat loogisesti?
- Onko ohjelman käyttö ristiriidassa muun maailman toimintaan?

3. Käyttäjän kontrolli ja vapaus

Käyttäjän pitäisi päästä nopeasti ja vaivatta takaisin kunkin vaiheen alkutilaan, tehtyään epätoivotun tai virheellisen valinnan. Ohjelma ei myöskään saisi tehdä häiritseviä asioita käyttäjän tahtoa vasten tai tältä kysymättä.

- Joutuuko ohjelmassa navigoimaan turhien hyppyjen kautta (ts. saman voisi tehdä paljon nopeammin pienellä muutoksella)?
- Joutuuko tietylle sivulle päästäksensä noudattamaan pakollisia ja hankalia reittejä?
- Pääseekö pääsivuille ja tärkeimmille alasivuille takaisin helposti?
- Voiko virheellisen syötteen (esim. lomakkeessa) muuttaa vielä lähettämisen jälkeen riittävän helposti?
- Onko näyttöjä/ikkunoita liian paljon, voisiko asiat ilmaista tiiviimmin?

4. Yhteneväisyys ja standardit

Viestien ja toimintojen pitäisi tarkoittaa yhteneväisesti aina samoja asioita (sanoja tai merkityksiä ei saisi vaihtaa lennossa). Olemassa olevia verkko- ja muita standardeja pitäisi hyödyntää yhteneväisyyteen pyrittäessä.

- Onko nimiä, värejä ja muita tunnisteita käytetty yhtenäisesti kaikilla sivuilla?
- Onko linkkejä, painikkeita, tunnisteita ja syötekenttiä käytetty yhtenäisesti läpi ohjelman?
- Ovatko navigointipalkki ja painikkeet tutuissa paikoissa?

- Näyttävätkö linkit, painikkeet ja syötekentät tutuilta (esim. väri ja muoto)?
- Onko navigointityyli eheä läpi ohjelman?

5. Virheiden estäminen

Ohjelman pitäisi tunnistaa mahdolliset virhetilanteet ja estää niiden toistuminen kertomalla käyttäjälle ennen virheen tapahtumista. Opastus pitäisi olla aina helposti saatavilla ja ymmärrettävissä.

- Onko lomakkeissa hyödynnetty riittävästi virheellisten syötteiden tarkistusta?
- Annetaanko ongelmallisista syötteistä selkeä ja opastava ilmoitus ajoissa?
- Onko syöte- ja toimintotilanteissa saatavilla opastus?

6. Tunnistaminen mieluummin kuin muistaminen

Asioiden, toimintojen ja vaihtoehtojen pitäisi olla näkyvissä käyttöliittymässä. Käyttöliittymän painikkeiden ja syötteiden pitäisi liittyä ohjelman toimintoihin loogisesti, niin että näiden vastaavuus on pääteltävissä helposti. Käyttäjää ei saisi pakottaa muistamaan asioita ruudulta toiselle siirryttäessä.

- Ovatko tärkeimmät toiminnot näkyvissä aina niin, että niiden sijaintia toisella sivulla ei tarvitse muistaa?
- Ovatko kaikki toimintoja aikaansaavat käyttöliittymäelementit merkitty niin, että ne ymmärretään painikkeiksi (tai syötekentiksi, tms.)?
- Ovatko käyttöliittymän elementit sijoitettu niin, että niiden riippuvuus ja suhde muihin ruudun elementteihin on selvä?
- Voiko käyttäjä edetä sivulta toiselle, ilman että hänen täytyy muistaa ulkoa aikaisemmilla sivuilla näkemäänsä tietoa?

7. Käytön joustavuus ja tehokkuus

Käytön pitäisi olla joustavaa ja tehokasta sekä aloitteleville että edistyneille käyttäjille. Käytön pitäisi olla myös joustavaa ja tehokasta käyttäjän laitteistosta ja yhteydestä riippumatta.

- Ovatko yleisimmät toiminnot aina käytettävissä ja näkyvillä?
- Voiko monimutkaista tai laajasisältöistä liittymää muokata yksinkertaisemmaksi tarpeidensa mukaan?
- Mukautuuko ohjelma eri näytöille, selainversioille, kirjasintyypeille, väreille, konetyypeille, käyttöjärjestelmille ja yhteysnopeuksille?
- Voiko usein käytettyihin sivuihin linkittää helposti?

- Voiko dynaamisesti tuotetut sivut saada helposti ladattua uudestaan (esim. kyselyt)?

8. Esteettinen ja minimalistinen design

Ruudulla pitäisi olla ne elementit, jotka ilmaisevat halutun tiedon, toiminnot, tunnelman ja tyylin, ei enempää.

- Onko ruudulla käytetty rajoitetusti värisävyjä (n. 1-3)?
- Onko kirjasintyyppejä ja kokoja käytetty rajoitetusti (n. 1-3)?
- Onko tyhjää tilaa hyödynnetty selkeyttämään näyttöjä?
- Kiinnittyykö huomio tärkeimpiin elementteihin ensin?
- Hallitseeko jokin tai jotkin elementti koko näyttöä ja sen navigointia muiden kustannuksella?
- Onko teksti sopivan mittaista, tyylistä ja kokoista ruudulta luettavaksi?

9. Virhetilanteiden tunnistaminen, ilmoittaminen ja korjaaminen

Virheilmoitusten pitäisi kertoa selkokielellä **mitä** tapahtui, **miksi** näin kävi, **miten** asia voidaan korjata ja kuinka se voidaan välttää ensi kerralla.

- Onko virheilmoitus ymmärrettävissä?
- Selviääkö virheilmoituksesta mitä tapahtui, miksi ja miten korjata/välttää tilanne?
- Ovatko virheilmoitukset kohteliaita ja välttävät syyttelyä?
- Ovatko korjaavat toimintaohjeet helposti seurattavissa?

10. Opastus ja ohjeistus

Vaikka käytön pitäisi tapahtua ilman opastusta ja ohjeita, ovat ne usein välttämättömiä käyttäjille. Näiden pitäisi olla helposti saatavilla, nopeasti etsittävissä, toimintaan ohjaavia, käyttötilannetta tukevia ja riittävän lyhyitä.

- Annetaanko opastusta automaattisesti vaikeissa paikoissa?
- Ovatko ohjeet aina saatavilla?
- Ovatko ohjeet ja opastus tilanne- tai sivukohtaisia?
- Ovatko ohjeet helposti ymmärrettävissä ja vaiheet toteutettavissa?
- Ovatko ohjeet lyhyitä (lyhyisiin, mutta järkeviin kokonaisuuksiin pilkottuja)?

Vakavuusluokitus

Jokainen arvioinnissa löydetty ongelma luokitellaan vakavuusasteikolla, joka kertoo asiantuntijan mielipiteen käytettävyysongelman vakavuudesta. Ongelman vakavuuden luokitus pitäisi nojata ainakin seuraavaan neljään seikkaan:

- **Esiintymistiheys**: kuinka usein potentiaaliseen ongelmatilanteeseen törmää? (usein/harvoin)
- **Vaikutukset käyttäjälle**: onko ongelmatilanteesta helppo vai vaikea selvitä? (vaikea/helppo)
- **Toistuvuus**: Onko ongelma helposti ohitettavissa, kun sen on kerran tunnistanut, vai vaivaako se jatkuvasti? (toistuva/ohitettava)

Vakavuusluokka ilmaistaan numeroilla nollasta neljään seuraavasti:

0 = En pidä ongelmaa käytettävyysongelmana
1 = Kosmeettinen ongelma: korjataan kun ehditään
2 = Pieni käytettävyysongelma: vaikeuttaa käyttö, korjataan
3 = Suuri käytettävyysongelma: vaikeuttaa merkittävästi, korjataan heti
4 = Katastrofaalinen ongelma: ohjelma lähes käyttökelvoton

B Cognitive walkthrough test form (Polson et al. 1992)

Cognitive Walkthrough for a Step

Task ____ Action# _______

1 Goal structure for this step

1.1 Correct goals

What are the appropriate goals for this point in the interaction? Describe as for initial goals.

1.2 Mismatch with likely goals

What percentage of users will not have these goals, based on the analysis at the end of the previous step? Check each goal in this structure against your analysis at the end of the previous step. Based on that analysis, will all users have the goal at this point, or may some users have dropped it or failed to form it? Also, check the analysis at the end of the previous step to see if there are unwanted goals, not appropriate for this step that will be performed or retained by some users. (% 0 25 50 75 100)

2 Choosing and executing the action

Correct action at this step: ____________________________________

2.1 Availability. Is it obvious that the **correct** action is a possible choice here? If not, what percentage of users might miss it? (% 0 25 50 75 100)

2.2 Label. What label or description is associated with the **correct** action?

2.3 Link of label to action. If there is a label or description associated with the **correct** action, is it obvious and is it clearly linked with this action? If not, what percentage of users might have trouble? (% 0 25 50 75 100)

2.4 Link of label to goal. If there is a label or description associated with the **correct** action, is it obviously connected with one of the current **goals** for this step? How? If not, what percentage of users might have trouble? Assume all users have the appropriate goals listed in Section 1. (% 0 25 50 75 100)

2.5 No label. If there is no label associated with the **correct** action, how will users relate this action to a current goal? What percentage might have trouble doing so? (% 0 25 50 75 100)

2.6 Wrong choices. Are there other actions that might seem appropriate to some current goal? If so, what are they and what percentage of users might choose one of these? (% 0 25 50 75 100)

2.7 Time-out. If there is a time-out in the interface at this step, does it allow time for the user

to select the appropriate action? How many users might have trouble? (% 0 25 50 75 100)

2.8 Hard to do. Is there anything physically tricky about executing the action? If so, what percentage of users will have trouble? (% 0 25 50 75 100)

3 Modification of goal structure

Assume the correct action has been taken. What is the system's response?

3.1 Quit or backup. Will users see that they have made progress towards some current goal? What will indicate this to them? What percentage of users will not see progress and try to quit or backup? (% 0 25 50 75 100)

3.2 Accomplished goals. List all current goals that have been accomplished. Is it obvious from the system response that each has been accomplished? If not, indicate for each, how many users will not realise it is complete.

3.3. Incomplete goals that look accomplished. Are there any current goals that have not been accomplished, but might appear to have been based on the system response? What might indicate this? List any such goals and the percentage of users who will think that they have actually been accomplished.

3.4 "And-then" structures. Is there an "and-then" structure, and does one of its subgoals appear to be complete? If the subgoal is similar to the supergoal, estimate, how many users may prematurely end the "and-then" structure.

3.5 New goals in response to prompts. Does the system response contain a prompt or cue that suggests any new goal or goals? If so, describe the goals. If the prompt is unclear, indicate the percentage of users who will not form these goals. (% 0 25 50 75 100)

3.6 Other new goals. Are there any other new goals that users will form given their current goals, the state of the interface, and their background knowledge? Why? If so, describe the goals and indicate, how many users will form them. NOTE that these goals may or may not be appropriate, so forming them may be bad or good.

Guide for cognitive walkthrough

Yleistä
Käytettävyystestauksessa käytetään yhtenä arviointi tapana kognitiivista läpikäyntiä. Tällä menetelmällä on tarkoitus löytää ohjelmien käyttöä ja niiden oppimista hankaloittavia rakenteita. Tässä testissä *ei arvioida* millään tavalla testaajan tietoja tai taitoja.

Menetelmän kuvaus
Tässä menetelmässä testaaja arvioi ensin lopputuloksen joka on mahdollista suorittaa testattavalla ohjelmaosalla. Tämän jälkeen testaaja arvioi samalla tietyllä kriteereillä selviytymismahdollisuuksiaan kohti asettamaansa tavoitetta. Ja lopuksi testaaja arvioi onnistumisensa tehtävässä. Seuraavassa kerrotaan testausmenetelmästä tarkemmin.

Varsinaisen läpikäynnin vaiheet ovat:
Valitaan tarkasteltava tehtäväkokonaisuus ja rajataan se.
Kuvataan käyttäjän alustavat tavoitteet (miksi ollaan ylipäätään tilanteessa, jossa tätä laitetta halutaan käyttää?).
Käydään läpi valittu tehtäväkokonaisuus ohjelman tai sen määrittelyjen avulla ja analysoidaan käyttäjän päätöksentekoprosessi kaikissa tilanteissa, joissa joudutaan suorittamaan valintoja tai tulkitsemaan tuloksia.

Kognitiivisessa läpikäynnissä käydään vuorovaikutustilanne läpi seuraavia asioita tarkastelevien kysymyskokonaisuuksien avulla:
Mikä on käyttäjän tavoite toimenpiteiden suorittamista aloitettaessa? Onko tavoite oikea?
Havaitseeko käyttäjä, onko oikea toimenpide käytettävissä?
Osaako käyttäjä yhdistää tavoitteensa oikeisiin vuorovaikutuselementteihin?
Ymmärtääkö käyttäjä toiminnon suoritettuaan, että hän on edistynyt tavoitettaan kohti?

Kognitiivisen läpikäynnin arviointilomakkeessa on tarkennettu ohjeisto kognitiivisen läpikäynnin suorittamiseksi, seuraavassa on lyhennetty kuvaus:
Ensimmäinen osa on kognitiivista läpikäyntiä tukeva lomake.
Toinen osa on jaettu kolmeen alalomakkeeseen:
Ensimmäisellä lomakkeella käydään läpi tehtävän osan suorittamiseen liittyvä käyttäjän tavoite.
Toisella lomakkeella arvioidaan käyttäjän mahdollisuuksia valita ja toteuttaa haluttu, oikea tai väärä toiminto.
Viimeinen lomake käsittelee käyttäjän tavoitteissa tapahtuneita muutoksia.
Lisäksi kognitiivisen läpikäynnin arviointilomakkeessa on tarkat kuvaukset arvioitavista tehtävistä

Lopputuloksen analyysin kannalta keskeisimmän osan muodostaa **käyttäjän päätöksentekoprosessin tarkastelu** ei ainoastaan tilanteen selvittäminen.

C Ravden & Johnson Method test form (Tukiainen & Lempinen, 1994)

JÄRJESTELMÄN KÄYTETTÄVYYSTESTI (ver. 1.1)

Testattava järjestelmä__________________________ Versio______

Testaajan nimi____________________________ Pvm___________

OSA K1: VISUAALINEN SELKEYS						
Visuaalisella selkeydellä tarkoitetaan sitä, että järjestelmän näytöt tulisivat olla selkeitä, hyvin organisoituja, yksikäsitteisiä ja helppoja lukea.	**Aina**	**Useimmiten**	**Joskus**	**Harvoin**	**Ei koskaan**	**Kommentit**
1. Onko tärkeät tiedot riittävästi korostettu? (esim. kohdistin, aktiivinen ikkuna, ohjeet, virheilmoitukset)						
2. Kun syötät tietoja, tiedätkö						
(a) mihin kohtaan tieto syötetään?						
(b) missä muodossa tieto tulisi syöttää (esim. pvm)?						
3. Onko tiedot ryhmitelty loogisesti?						
4. Onko erityyppiset tiedot erotettu selvästi toisistaan? (esim. ohjeet, syöttöalueet, tulosalueet)						
5. Jos näytöllä (ikkunassa) esitetään paljon tietoa, onko näyttö jaettu selkeästi hahmotettaviin osiin?						
6. Auttaako värien käyttö tietojen hahmottamisessa?						
7. Onko esitettävä tieto helppo nähdä ja lukea?						
8. Onko näytön (ikkunan) yleisrakenne selkeä?						
9. Ovatko kuviot ja kuvat (esim. diagrammit) selkeitä (esim. koko, väri, selitteet)?						
10. Löydätkö helposti tarvitsemasi tiedot?						
11. Millaisen yleisarvosanan antaisit näyttöjen (ikkunoiden) selkeydestä	☒ Täysin riittämätön ☒ Melko riittämätön ☒ Kohtalainen ☒ Melko riittävä ☒ Täysin riittävä					

Muut kommentit						
OSA K2a: TIETOJEN YHDENMUKAISUUS						
Tietojen yhdenmukaisuudella tarkoitetaan sitä, että järjestelmän syöttö- ja tulostietojen muoto tulisi olla käyttäjälle tuttu ja sama tietojen kaikissa esiintymissä.	**Aina**	**Useimmiten**	**Joskus**	**Harvoin**	**Ei koskaan**	**Kommentit**
1. Ovatko käytetyt lyhenteet, koodit ja muut esitystavat						
a) helppo tunnistaa ja ymmärtää?						
b) yleisesti käytettyjen esitysmuotojen mukaisia (esim. päivämäärä muotoa pp.kk.vvvv)?						
2. Ovatko käytetyt ikonit, symbolit ja muu kuvallinen tieto						
a) helppo tunnistaa ja ymmärtää?						
b) yleisesti käytettyjen esitysmuotojen mukaisia?						
3. Onko käytetty ammattisanasto ja terminologia sinulle tuttua?						
4. Esiintyykö samanlainen tieto säännöllisesti samassa muodossa						
a) syötettäessä tietoa järjestelmään?						
b) järjestelmän esittämänä?						
c) järjestelmään syötettäessä ja järjestelmän esittämänä ?						
5. Esitetäänkö ja käsitelläänkö tieto yksiköissä, joiden kanssa normaalisti työskentelet? (esim. markat, kilometrit)						
6. Vastaako tulosteissa oleva tietojen esitysjärjestys ja -tapa näytön esitysjärjestystä ja -tapaa?						

7. Millaisen yleisarvosanan antaisit tietojen yhdenmukaisuudesta	⊠ Täysin riittämätön ⊠ Melko riittämätön ⊠ Kohtalainen ⊠ Melko riittävä ⊠ Täysin riittävä
Muut kommentit	

OSA K2b: KÄYTÖN YHDENMUKAISUUS

Käytön yhdenmukaisuudella tarkoitetaan sitä, että järjestelmän käyttäminen tulisi olla käyttäjän ennustettavissa ja noudattaa tuttuja käytäntöjä.	**Aina**	**Useimmiten**	**Joskus**	**Harvoin**	**Ei koskaan**	**Kommentit**
1.Onko kunkin tiedon syöttötapa yhdenmukainen järjestelmässä (esim. tietojen syöttäminen näppäimistöltä, toimintojen valitseminen valikosta)?						
2. Reagoiko järjestelmä samalla tavalla toisiaan vastaaviin toimintoihin (esim. ENTER-näppäimen painallus eri tilanteissa)?						
3. Onko järjestelmässä käytetty ammattisanasto ja terminologia sinulle tuttua?						
4. Ovatko ohjaustoiminnot samankaltaisia kuin muissa mahdollisesti käyttämissäsi järjestelmissä (esim. tulostoiminnon käynnistäminen)?						
5. Onko järjestelmän toiminnot järjestetty tehtävääsi vastaavalla tavalla?						
6. Voitko suorittaa tarvitsemasi toiminnot helposti haluamassasi järjestyksessä?						
7. Toimiiko järjestelmä olettamallasi tavalla?						

8. Millaisen yleisarvosanan antaisit käytön yhdenmukaisuudesta	⊠ Täysin riittämätön ⊠ Melko riittämätön ⊠ Kohtalainen ⊠ Melko riittävä ⊠ Täysin riittävä
Muut kommentit	

OSA K3: PALAUTTEET						
Palautteilla tarkoitetaan sitä, että järjestelmän antamat palautteet tulisivat olla selkeitä, informatiivisia ja käyttäjän sen hetkistä toimintaa eteenpäin vieviä.	**Aina**	**Useimmiten**	**Joskus**	**Harvoin**	**Ei koskaan**	**Kommentit**
1. Ovatko järjestelmän ohjeet ja ilmoitukset ytimekkäitä ja selkeitä?						
2. Liittyvätkö järjestelmän ilmoitukset käsiteltävään asiaan?						
3. Ilmaisevatko ohjeet ja kehotteet selvästi, mitä tulee tehdä?						
4. Onko sinulle selvää, mitä toimintoja voit tehdä kussakin vaiheessa?						
5. Onko sinulle selvää, mitä sinun on tehtävä suorittaaksesi tietyn toiminnon (esim. mitä valintoja/optioita valita, mitä näppäimiä painaa)?						
6. Onko sinulle selvää, mitä tietoja kulloinkin pitää syöttää?						
7. Onko sinulle selvää, mitä syötteiden nopeutuksia järjestelmä sisältää (esim. lyhenteet, peräkkäisten valintojen yhdistäminen (onko nopeutukset näkyvillä)) ?						
8. Ovatko tilailmoitukset (esim. mitä järjestelmä tekee tai on juuri tehnyt): a) informatiivisia?						
b) tilanteeseen sopivia?						
9. Ilmoittaako järjestelmä selvästi suoritettuaan pyytämäsi toiminnon (onnistuneesti tai epäonnistuneesti)?						
10. Ilmoittaako järjestelmä käsittelevänsä syötettäsi, mikäli toiminto kestää pitkään?						
11. Ilmeneekö virheilmoituksista selvästi: a) missä virhe ovat?						

b) mikä virhe on?						
c) miksi virhe tapahtui?						
12. Onko sinulle selvää, mitä pitäisi tehdä virheen korjaamiseksi?						
13. Millaisen yleisarvosanan antaisit järjestelmän palautteista	☒ Täysin riittämätön ☒ Melko riittämätön ☒ Kohtalainen ☒ Melko riittävä ☒ Täysin riittävä					
Muut kommentit						
OSA K4: YKSIKÄSITTEISYYS						
Yksikäsitteisyydellä tarkoitetaan sitä, että järjestelmän toiminta ja rakenne tulisi olla käyttäjälle selvä ja ymmärrettävä.	**Aina**	**Useimmiten**	**Joskus**	**Harvoin**	**Ei koskaan**	**Kommentit**
1. Kun järjestelmä tarjoaa joukon toimintovaihtoehtoja (esim. valikossa), onko sinulle selvää, mitä kukin toiminto tarkoittaa?						
2. Onko sinulle selvää, missä osassa järjestelmän osassa kulloinkin olet?						
3. Onko sinulle selvää, mitä järjestelmän eri osat tekevät?						
4. Onko sinulle selvää kuinka, missä ja miksi yhdessä osassa järjestelmää tekemäsi muutokset vaikuttavat muihin järjestelmän osiin?						
5. Onko sinulle selvää, miksi järjestelmä on organisoitu ja rakennettu kuten se on?						
6. Onko sinulle selvää, miksi näyttöjen (ikkunoiden) sarjat ovat sellaisessa järjestyksessä kuin ne ovat?						
7. Onko järjestelmän rakenne mielestäsi järkevä?						
8. Onko järjestelmä tehtäväsi kannalta hyvin organisoitu?						

9. Millaisen yleisarvosanan antaisit järjestelmän yksikäsitteisyydestä	☒ Täysin riittämätön ☒ Melko riittämätön ☒ Kohtalainen ☒ Melko riittävä ☒ Täysin riittävä					
Muut kommentit						
OSA K5: OIKEA TOIMINNALLISUUS						
Oikealla toiminnallisuudella tarkoitetaan sitä, että käyttäjä voi suorittaa haluamansa tehtävän haluamallaan tavalla.	**Aina**	**Useimmiten**	**Joskus**	**Harvoin**	**Ei koskaan**	**Kommentit**
1. Onko käytettävissä oleva syöttölaite (esim. hiiri, näppäimistö) sopiva tehtävän suorittamiseksi?						
2. Sisältääkö jokainen näyttö (ikkuna) kaiken sen tiedon, joka on tehtäväsi kannalta oleellista?						
3. Onko sinulla koko ajan käytössäsi kaikki tarvitsemasi toiminnot?						
4. Onko sinulla saatavilla kaikki se tieto, jota tarvitset nykyisen tehtävän suorittamiseksi?						
5. Salliiko järjestelmä, että teet kaiken sen minkä katsot tarpeelliseksi tehtävää suorittaessasi?						
6. Sisältävätkö avustus- ja opastustoiminnot todellisia tehtäväalueesi esimerkkejä ja ongelmia?						
7. Millaisen yleisarvosanan antaisit järjestelmän toiminnallisuudesta	☒ Täysin riittämätön ☒ Melko riittämätön ☒ Kohtalainen ☒ Melko riittävä ☒ Täysin riittävä					
Muut kommentit						
OSA K6: JOUSTAVUUS JA HALLINTA						

Joustavuudella ja hallinnalla tarkoitetaan sitä, että järjestelmä joustaa käyttäjän toiminnoissa ja käyttäjä voi itse muokata järjestelmää jonkin verran.	**Aina**	**Useimmiten**	**Joskus**	**Harvoin**	**Ei koskaan**	**Kommentit**
1. Onko sinun helppo perua suorittamasi toiminto ja palata edelliselle tasolle tai näytölle? (esim. jos teet väärän valinnan)						
2. Jos voit perua toiminnon, niin voitko myös perua perumisen (ts. kumota peruuttamisen)?						
3. Voitko riittävästi kontrolloida järjestystä, jossa pyydät tietoja tai suoritat joukon toimintoja?						
4. Voitko riittävästi selailla näyttöjä (ikkunoita) edestakaisin?						
5. Jos järjestelmä on valikko-ohjattu, niin voitko helposti palata päävalikkoon mistä tahansa osasta järjestelmää?						
6. Voitko tarvittaessa helposti siirtyä järjestelmän eri osiin?						
7. Voitko tarvittaessa kumota koneen tuottaman (esim. oletusarvo) tiedon?						
8. Voitko helposti muuttaa tiettyjä osia käyttöliittymästä omien mieltymystesi ja tarpeittesi mukaan (esim. värit, oletusarvot ja tiedon esitysmuodot)?						
9. Millaisen yleisarvosanan antaisit järjestelmän joustavuudesta ja hallinnasta	⊠ Täysin riittämätön ⊠ Melko riittämätön ⊠ Kohtalainen ⊠ Melko riittävä ⊠ Täysin riittävä					
Muut kommentit						

OSA K7: VIRHEIDEN KORJAAMINEN

Virheiden korjaamisella tarkoitetaan sitä, että käyttäjä voi helposti korjata tekemänsä virheet ja ei joudu tekemään ylimääräistä työtä virheitä korjatakseen.	**Aina**	**Useimmiten**	**Joskus**	**Harvoin**	**Ei koskaan**	**Kommentit**

1. Voitko helposti korjata tekemäsi virheet (esim. kirjoitusvirheet) ennekuin järjestelmä käsittelee syötteen?						
2. Onko sinun helppo korjata tekemäsi virheet jälkikäteen?						
3. Saatko virheen tapahtuessa kaiken tarvitsemasi diagnostisen tiedon ratkaistaksesi ongelman? (esim. missä ja mikä vika on, mitä tarvitaan sen korjaamiseksi)						
4. Millaisen yleisarvosanan antaisit virheiden korjaamisesta	☒ Täysin riittämätön ☒ Melko riittämätön ☒ Kohtalainen ☒ Melko riittävä ☒ Täysin riittävä					
Muut kommentit						

OSA K8: KÄYTTÄJÄN OPASTAMINEN

Käyttäjän opastamisella tarkoitetaan sitä, että käyttäjä saa opastusta haluamallaan hetkellä ja että opastus on selkeää ja toiminnan kannalta hyödyllistä.	**Aina**	**Useimmiten**	**Joskus**	**Harvoin**	**Ei koskaan**	**Kommentit**
1. Voitko käynnistää avustustoiminnon helposti mistä tahansa kohdasta järjestelmää?						
2. Onko selvää, kuinka käynnistät ja lopetat avustuksen?						
3. Esitetäänkö ohjeet selvästi häiritsemättä kesken olevaa tehtävääsi?						
4. Pyytäessäsi ohjeita, selittävätkö ne nykyisessä tilanteessa valittavissa olevat toiminnot?						
5. Löydätkö ohjeista tärkeän tiedon suoraan tarvitsematta selailla läpi tarpeetonta tietoa?						
6. Salliiko avustustoiminto myös järjestelmän muiden osien tietojen selailun?						
7. Tarjoavatko järjestelmän kirjalliset käyttöohjeet perusteellisen, järjestelmän kaikki ominaisuudet kattavan kuvauksen?						
8. Löydätkö käyttöohjeesta helposti tarvitsemasi						

osan?						
9. Onko järjestelmän kaikkien avustusmuotojen rakenne sellainen, että löydät niistä tehtävän suorittamiseksi tarvitsemasi tiedon helposti?						
10. Selittävätkö avustustoiminnot tarpeeksi hyvin virheet (sekä sinun tekemäsi että järjestelmästä johtuvat) ja niiden korjaamiseen tarvittavat toimenpiteet?						
11. Millaisen yleisarvosanan antaisit käyttäjän opastamisesta	☒ Täysin riittämätön ☒ Melko riittämätön ☒ Kohtalainen ☒ Melko riittävä ☒ Täysin riittävä					
Muut kommentit						

OSA K9: JÄRJESTELMÄN KÄYTETTÄVYYDEN ONGELMAT

Onko jokin seuraavista aiheuttanut sinulle ongelmia	**Ei ongelmia**	**Vähäisiä ongelmia**	**Vähän ongelmia**	**Joitkin ongelmia**	**Paljon ongelmia**	**Kommentit**
1. Selvittää, kuinka järjestelmää käytetään						
2. Puutteellinen tai puuttuva järjestelmän käytön opastus						
3. Huonot tai puuttuvat järjestelmän käsikirjat						
4. Ymmärtää, kuinka suoritat haluamasi tehtävät						
5. Tietää, mitä sinun pitää tehdä seuraavaksi						
6. Käsittää, kuinka näytön tieto liittyy tehtävääsi tai toimintaasi						
7. Tietää, kuinka löydät haluamasi tiedon						
8. Vaikeasti luettava tieto						
9. Liian paljon värejä näytöllä						
10. Häiritseviä värejä, joita on vaikea katsella						
11. Järjestelmän joustamattomuus						

12. Avustustoiminnon joustamattomuus						
13. Et tiedä, missä osassa järjestelmää olet, mitä olet tekemässä tai tehnyt						
14. Liian paljon muistettavaa tehtävää suoritettaessa						
15. Järjestelmän liian nopea toiminta, jolloin on vaikea ymmärtää mitä on tekeillä						
16. Tieto, joka häviää näytöltä ennen kuin ehdit lukea sen.						
17. Järjestelmän hitaus						
18. Järjestelmän odottamattomat toiminnot						
19. Tehtävään sopimaton syöttölaite						
20. Tietää, minne tai miten syöttää tietoa						
21. Tiedon syöttämiseen kuluva liian pitkä aika						
22. Virheiden välttämiseen vaadittava varovaisuus						
23. Ratkaista, kuinka korjata virheet						
24. Virheiden korjaamiseen kuluva liian pitkä aika						
25. Samantyyppisten tehtävien suorittaminen eri tavoilla						
26. Millaisen yleisarvosanan antaisit järjestelmän käytettävyydestä	☒ Täysin riittämätön ☒ Melko riittämätön ☒ Kohtalainen ☒ Melko riittävä ☒ Täysin riittävä					
Muut kommentit						

OSA K10: YLEISIÄ KYSYMYKSIÄ JÄRJESTELMÄN KÄYTETTÄVYYDESTÄ

Kerro mielipiteesi järjestelmän käytettävyydestä vastaamalla alla oleviin kysymyksiin.	**Kommentit**
1. Mitkä ovat järjestelmän parhaat puolet?	

2. Mitkä ovat järjestelmän huonoimmat puolet?	
3. Onko järjestelmässä osia, jotka ovat hämmentäviä tai vaikeita ymmärtää?	
4. Oliko järjestelmässä jotain erityisen ärsyttävää, vaikka se ei aiheuttanutkaan kohtuuttomia	
5. Mitkä olivat yleisimmät virheet, joita teit käyttäessäsi järjestelmää?	
6. Mitä muutoksia tekisit järjestelmään tehdäksesi sen käyttäjän kannalta paremmaksi?	
7. Onko jotain muuta, mitä haluaisit lisätä koskien järjestelmää?	

7.2 Appendix 2 NEMU at nutshell

NEMU, Network Element Management Unit, is the common name for an additional network management system. This system is separate but cannot work without the DX200/IPA2800 switching exchange. The NEMU system is composed of client computers, a server computer platform and its operating system and a specific collection of applications and server services. NEMU offers extra value to telephone network operators by adding graphical tools to maintain and follow up the switching exchanges operations. More about NEMU is in Appendix 2.

The server computer platform can be any type of processor based; the only limitation for the platform is to be able to run Microsoft Windows, Unix or Linux or HPUX operating systems. Nowadays the server platform is a Compaq multiprocessor server system.

The NEMU applications are a collection of programs for offering extra values for network management to the operator.

NEMU architecture is a client-server type Java application, where the possible client types are both thin and fat client and additionally some applications can run in offline mode without a server connection. Every single connection to the NEMU server needs its own Application Launcher or the other way the Application Launcher can only make connection to one NEMU server at a time. This feature means that the operator can open several Application Launchers and those applications can be operated in the same screen.

When the user needs a NEMU application, he starts the Application Launcher. During the Application Launcher start up it loads system nested Java jars of NEMU in the client computer, secondly, the Application Launcher starts the login dialog and demands the user authentication. The third phase sets up the connection and opening of a new session to the NEMU server. Now all the users accepted applications are in the Application Launcher's window and the user can choose the desired application to use. After shutting down the Application Launcher, it makes ending actions to shut down the open applications, clear the session and user authentication.

All the thin client applications run in a separate server beside the DX200/IPA2800 switching platform but all the individual NEMU functionalities are linked and connect in the DX/IPA server. NEMU applications cannot run without DX/IPA connection because some services use DX switching exchange's own services. NEMU applications GUIs are made in Java, and high-level services such as CORBA and COM components are made mainly in C++. All NEMU applications are designed to run in Windows NT/2000, Unix/Linux and HPUX platforms. In application design must also be noticed limitations and possibilities of target platform.

NEMU applications can be e.g. user management, software version checking, fault management, radio network controlling tools, network performance exploring tool etc. All applications, except the Application Launcher, have a common help library and presentation application, called NED, Nokia Electronic Documentation, that did not work properly during usability testing. This application collection has at least applications launching utility called Application Launcher. The Application Launcher is a Microsoft Windows like application, which can start different applications by offering some basic services. These services are authentication, session management, and own Java help system as mentioned before. All applications, which started in Application Launcher, are authenticated. This means that users do not retype username or password even application requires authentication at the start, if the session is still alive.

The NEMU system administrator can set the limitations for accepted users. The limitation can be, for example, that a user can only view but not set new values via applications to NEMU system.

7.3 Appendix 3 NEMU Applications

Note that the capital letter preceding the program name refers to the command, which is used to call that program, for example, J is used to call the Version viewer

A Action Log Viewer

B Active Sessions This application shows all active connections of the NEMU.

C Application Launcher This application is a base of NEMU software. On this platform can be launched any application on NEMU. Application includes all needed services such as authentication and session management.

D Application Launcher OO-version This application was a prototype of new Application Launcher. This was not finalized and some of it functions were not implemented. OO version means Object Oriented version.

E MMI Window This is command based, Telnet lookout, application, which can connect to any NEMU. Application uses MML, man-machine language, to controlling DX200/IPA2800 switching exchange.

F NE Login This application executes network element login actions. Application has only the login feature and help.

G NEMU Log Analyser is a tool to help analyse event or/and trace logs. It shows events graphically on timeline and also in textual format.

H Personal Settings This application can handle, show and modify, user’s personal settings of NEMU.

I Remote Launcher With this application can launch any NEMU application in other NEMU. Application includes authentication services.

J Version Viewer This application can list versions of the NEMU software's. Application has several different functions to data queries.

K Alarm Browser Jyväskylä Polytechnic students project group members had produced a NEMU application for handling of all kinds of alarms in a NEMU. In this application the possible handling methods are browsing, cancelling and quitting incoming alarms. The application has the main window, which shows all alarms and two dialogs for handling alarms.

7.4 Appendix 4 Usability Test Cases

A Heuristic evaluation, the first meeting

Ne login
Start the application
Try to clarify what you can do
Leave text fields empty and press *OK*
Follow the instructions
Write the right name of the NEMU server, but wrong username or password
Follow the instructions
Log in the NEMU server with right username and password

Active Sessions
Start the application
Try to clarify what you can do
Update you data
Try to stop any of the listed sessions

Personal settings
Start the application
Try to clarify what you can do
Input wrong a password
Input wrong password in *retype password* field
Change password with valid inputs
BEFORE you log out, RESET the previous password!

Remote launcher
Start the application
Try to clarify what you can do
Try to open connect in any other NEMU server with an invalid data
Follow the instructions
Open connection to any of the listed NEMU server
Open any application
Try to clarify which applications are able to start

Application launcher
Try to clarify what you can do
Check the consistency between help and application

B Heuristic evaluation, the second meeting

MMI Window
Start the application
Try to clarify what you can do
Open a new instance of MMI connection

Action log viewer
Start the application
Try to clarify what you can do
Check the menu
Save as –dialog, Settings –dialog, Filter Events –dialog and Detail -dialog
Try to clarify what you can do

C Heuristic evaluation, the third meeting

Version viewer
Start the application
Try to clarify what you can do
Check the menus, icons and help
Properties -dialog
Delete any of the listed files
Check the properties of the any of the listed files
Check the properties of the file that have long name and file path
Settings -dialog
Try any variations of settings
Find -dialog
Try to find with different conditions
Get Version Of Single File -dialog
Try to find with different conditions

User Authority Manager
Start the application
Try to clarify what you can do
Check the menus and icons
Create new a group
Create new a group with the existing group name

Create new a user
Create new a user with the existing user name
Create new a user with the invalid data
Set privileges of the user

Set applications of the user

D Ravden & Johnson method, the fourth meetings

Application Launcher
Start the application (log in dialogue)
Try to clarify what you can do
Check the menus, icons and help

E Ravden & Johnson method, the fifth meetings

Version viewer
Start the application (log in dialogue)
Try to clarify what you can do
Check the menus, icons and help
Properties -dialog
Delete any of the listed files
Check the properties of the any of the listed files
Check the properties of the file that have long name and file path
Settings -dialog
Try any variations of settings
Find -dialog
Try to find with different conditions
Get Version Of Single File -dialog
Try to find with different conditions

F Heuristic evaluation, the sixth meeting

Nemu Log Analyser
Try to clarify what you can do and how you can do it.
Is the view clear and easily learnable?
Check the menus, icons
Check the consistency of help and application

G Additional heuristic evaluation for Application Launcher OO version

Application Launcher, Object Oriented
Try to clarify what you can do and how you can do it.
Is the view clear and easily learnable?
Which one of the basic control feature (tabs or tool buttons) is better to use?
Is there enough information in objects/applications tab, are icons good enough?
Which one of the properties listing is better?
Which one of the structures listing is better (list and menu or tree)?
Free form comments

Skenaario
You are an operator in a big teleoperator's main operation center. Your duty is control your network. Your main tasks are update privileges of the network elements user groups and users.

Next tasks you have to do with the both of AL's interfaces (1 and 2).
Create new group and name it as a *new_group*
Create new user and name it as a *new_user*
Set user password as a *nemuuser*
Leave the MMI username field empty
Link user to previous made group
Set privileges of the group only for the viewing applications

Is there any differences between interfaces, which one might be better or/and efficient to use?

7.5 Appendix 5 Pilot Usability Tests Results

Test backgrounds: These tests are planned usability tests at the same time these tests are pilot tests for wider NEMU usability test. Main purpose of these tests is to evaluate the usability of Alarm Browser and to make check of usability evaluation materials such as test cases and test forms.

Date and time: 20.03.2001 13.00 - 15.10

Place: Nokia, Mattilanniemi 2, Jyväskylä

Test persons: Four persons, who have excellent knowledge of Information technology. Each of them has approximately 7 years experience of computers usage. Their own estimation of their knowledge of usability in general is good.

Tests: Visual walk through and heuristic evaluation.

Background of Tested Application: Tested application was Jyväskylä Polytechnics project course outcome. This application can chart and trace malfunctions and errors of NEMU. This application was not complete, thus part of the tests were made in paper prototypes. So the results of the tests could be used for further developing of application. All of testees ware developers of the application; this fact sets some conditions of results. Evaluation of own work is not easy, and might not be objective.

Visual Walkthrough

Purpose of this test is find things that affect usability either positively or negatively. This test is based on heuristics. Execution of the test: Testee walk through features of application and evaluate it in scale –10, -5, 0, 5, 10, where –10 is the worst and 10 is the best value of estimation, zero is meaningless. All estimations are collected in one table and with statistical methods each feature gets its own value, see Table 1.

Interpretation of Table

Meaning of the first row titles:

- **-10, -5, 0, 5, 10** (*need to fix now, need to fix, good, excellent):* Testees estimations of features.
- **AV !0:** average value of columns of the row, but not zero

- **¼:** quartile of the average value.
- **% in ¼:** percentage of estimation that belong to average value quartile.
- **Todo:** This column has artificial classification, which propose the action of correction or miss fit of evaluation. Values of correction are *excellent, good, need to fix*, and *need to fix now*. Statistical miss fit values are *wide dispersion* and *low number of estimations.* In this test wide dispersion >= 33%, so value is 0 or near it, and low number of estimations >=2. Todo gets the same value than estimation, if there are enough estimations and dispersion is low enough.
- **Feature:** evaluated feature of system.

Table 1

-10	**-5**	**0**	**5**	**10**	**AV !0**	**¼**	**% in ¼**	**Todo**	**Feature**
3	1				-8,75	-10	75	Need to fix now	Readability
			3	1	6,25	10	25	Good	Functionality and navigation
		1	2	1	6,67	10	33	Excellent	Consistency
			1	3	8,75	10	75	Excellent	User control
		2	1		5,00	5	100	Low number of estimations	Help and documentation
	1	2			-5,00	-5	100	Low number of estimations	Error messages
	3		1		-2,50	-5	75	Need to fix	Finding the information
	1	1	1	1	3,33	5	33	Good	Layout of elements
	2		2		0,00	0	0	Wide dispersion	Features texts (colours and fonts)
2				2	0,00	0	0	Wide dispersion	Figures and graphics
			2	1	6,67	10	33	Excellent	Command buttons (clarity and unambiguity)
	1	1	1	1	3,33	5	33	Good	Command buttons (styles and colour)
			2	2	7,50	10	50	Excellent	Overall structure

Summary of the Test

Clearly *Good* or *Excellent* features

- Functionality and navigation
- Consistency
- User control
- Layout of elements
- Command buttons (clarity and unambiguity)
- Command buttons (styles and colour)
- Overall structure

Clearly *Need to fix* or *Need to fix now* features

- Readability
- Finding the information

Other estimations need more evaluation, to reduce dispersion of estimations and to increase meaningful estimations.

Comments

Here are only the most essential free form comments; others are in heuristic evaluation report.

What is the first thing that you noticed?

Picture in upper left, bell icon of alarms, density of text and other information.

How many elements are in the window?

Three or four

Is the layout of window harmonious? If not, why?

Bell icons are not on the same line. Layout is left balanced.

Can you assume the function of application that happened command buttons and other texts?

- History graph
- Timeline
- History time frame
- Acknowledge all
- Unit
- Object
- Alarm text
- Tools

Yes to all texts.

Are all of the terms descriptive?

Not all

Conclusion

Visual walk through tells that application has some excellent features and some of the features need to work so that usability would be better. There was, for example, feature *Command buttons (clarity and unambiguity)* it is good in estimation but in other hand user cannot assume all of its functionality.

Even if the estimation gives a good degree of overall structure, some of the testees said that layout is left balanced. One of the testees said that all feature gets familiar quite soon.

Heuristic Evaluation

Heuristic evaluation is a set of certain rules (heuristics) based on evaluation method. This method tries to find those problems that can be usability problems.

Execution of Test

Testee makes evaluation of whole application or a part of it. Testee walks through tested window on the computer screen and compares it to heuristic checklist and makes his/her estimation in certain scale with respect to the usability of the application.

Test Results

Main window

All testees made this test. Only some of the features were implemented.

Table 2

Main window	**Severity**				
Heuristic	**0**	**1**	**2**	**3**	**4**
Feedback			**X**		
Speak the users' language		**X**			
Clearly marked exits and user control	**X**				
Consistency and standards		**Q**			
Prevent errors		**Q**			
Minimise user memory load			**Q**		
Shortcuts, flexibility		**Q**			
Use simple and natural dialogue		**X**			
Good error messages			**Q**		
Help and documentation		**L**			

Interpretation of Table 2

Q: Too wide dispersion of estimations
L: Low number of estimations
Only the heuristic *Clearly marked exits and user control* was good enough and testees did not require changes of it.

These features need to be fixed

- Feedbacks
- Minimise user memory load
- Good error messages

- Lack of *Help and documentation*, this is not noticed by testees.

Other heuristics rules were not evaluated, because testees did not consider they harmful enough or those heurisrics does not get needed amount of estimations so those heuristics might need some addional studies.

Free form comments

- Background of tabs is too dark, this might not good at LCD's
- Lines are too close each other.
- Icons are too small and blurred.
- Font size and style must be user defined.
- Time column is like zigzag.
- *There is no Find!*
- Metaphors are quite easy.
- Copy icon sucks.
- Reordering of fields is really good.
- Status bar is not clear; it has drowned into the window.
- There need to be a connected –icon in status bar. Refreshing is unknown.
- Icons are ambiguous, you have to know they meaning.
- Bell icon does not fit to colour-blind ->it could be the number of icons that shows the seriousness of alarm.
- Jumps to eyes.
- Title of window is missing.

Settings dialog

This test was made in paper test, because this dialog was not implemented yet.

Table 3

Settings dialog	**Severity**				
Heuristic	**0**	**1**	**2**	**3**	**4**
Feedback			**X**		
Speak the users' language			**Q**		
Clearly marked exits and user control		**X**			
Consistency and standards				**X**	
Prevent errors			**Q**		
Minimise user memory load		**Q**			
Shortcuts, flexibility		**Q**			
Use simple and natural dialogue					**Q**
Good error messages			**Q**		
Help and documentation		**Q**			

Interpretation of Table 3

Q: Too wide dispersion of estimations. Testees does not agree their evaluations

This evaluation does not get unanimous estimations enough.

Clearly poor features

- Use of simple and natural dialogue
- Help and documentation

Some corrections, which need to be made

- Feedback
- Speak the users' language
- Consistency and standards
- Prevent errors, especially on showing the timestamps
- Good error messages

Testees did not notice as a usability problem

- Clearly marked exits and user control
- Minimise user memory load
- Shortcuts, flexibility

Free form comments

- Too many things in the dialogues.
- Some names are weird.
- The text "*during*" is not good.
- Does not follow any standard.
- Looks so awful
- There is not any kind of title in this dialog.

Conclusion

In heuristic evaluation there were found some very clean usability problems. The most essential features that need to be fixed immediately are status of application and making help and other instructions. The best feature of Alarm browser is user control.

In settings dialog there is some important features that have to be fixed or redesign. These are make the visual outlook simple and improve consistency and status of application. There is also some work to be done with error preventing. Testees cannot bring up any good feature.

There is quite easy to improve the usablity of the main window, but setting dialogues need much more redesign work to do.

7.6 Appendix 6 The Results of The Consistency Inspection Meeting

Place: Tampere

Date: 6.2.2001 10.00-15.30

Testees: 7 persons from different Units (Nokia Networks personnel from Helsinki, Jyväskylä, and Tampere).

Tested applications: 13 NEMU applications. In this paper is presented only problems of applications that are included in other usability evaluations of this study.

Common comments

- Tool bars must be harmonised -> tool bar have to act as NET Look&Feel demo (NET Look&Feel: 4.1.4. Nokia bar, 4.1.4.1. Menu bar/Tool bar) (`http://osstre.ntc.nokia.com/NMS/quality/usability/uiguide/netlookfeel/demo/demo.html`)
- Colouring of dialogues has to be as said in guidance. Colouring must be: User Foreground: is light grey (230-230-230) and User Background standard grey (192-192-192). (NET Look&Feel: 5.3.1. User Foreground and Background Colours)
- Status bars have to as said in guidance (Nokia Look&Feel: 4.1.5. Status bar)
- Font colors and sizes have to as said in guidance. Check ”bold” font face usage. (Nokia Look&Feel: 5.3.5. Label Text Colour, 6 Typography, 5.2. Primary and Secondary Colours)
- Items in “Help” -menu are: “Help on This Window” and “Documentation” (Use underlined characters as a mnemonic.)
- Icon, size 20x20, has to insert in menus, if the menu item have no icon, transparent empty 20x20 icon must be inserted.
- If the whole menu has no icons, then no empty icons inserted.
- All menu items text starts with capital.
- Status bar status have to be: application + current action (lower case no comma separate), for example, "Object browser is ready"
- All (command) buttons have to be the same size.

- Tables do not have “double” frames. (See Tincidunt-> Veniam `http://osstre.ntc.nokia.com/NMS/quality/usability/uiguide/netlookfeel/demo/demo.html`)
- Abbreviation of the WCDMA Cell is WCEL
- No version information in window title.

Requirements to Platypus

- Platypus has to offer the same “About box” regardless of application.
- There is wrong font colour (Look & Feel) in ”wrong value” field, it have to be white instead of black.

Some Questions to Representatives

- Which one is right to use Nokia or Nokia Networks?
- Which is the right place command buttons? When they are in the middle of the dialogue and when they are justified on left? What is the size of “tiny” dialoque, in which buttons have to centred? According the NET Look&Feel- guidance buttons must be at the same line as the text, when this is appropriate?
- There is no guidance to edited table.
- How about the table colouring? How expanding table is described?

MMI Window

- Remove "One touch expander" arrows.
- Ctrl-C does not work.
- Menu background is wrong. Check NET Look & Feel. Documentation.
- Nokia bar is missing -> has to add
- Separation line must be inserted over the “About” and “Exit” menu items.
- Mnemonic of the Save is A.
- Help on This Window icon have to be like this:

Action Log Viewer

- Tool bar and menu have wrong colours. No bold font face in menu.
- Tables need “air”
- The “Log” text have to be changed to “File”
- Remove the borders of the titles
- Filter dialogue: Must be left justified.
- Check the background colour of combo box
- No radio buttons in menu

Active Session

- Colours and borders as said in guidance.
- Add Nokia bar
- More "air" in layout
- Tool bars and menus as said in guidance.
- Close and Help items in menu.
- Application layout has to changes as an application style instead of dialogue style.
- If the application is dialogues style application, then structure of application have to changed more simple.

Conclusion of consistency inspection

As the previous topic “*Some Questions to Representatives*” show that there was not enough valid and unambiguous information to support developing. The most of the findings are quite easily fixed and they reoccurrence might be prevented with simple and clear guidance. This inspection shows that this kind of evaluation and comparing between applications increase integrity of whole product family. In this case it is remarkable essential because NEMU applications are made in different sections and place and time between the first and the last evaluated application developing was more than a year.

7.7 Appendix 7 Heuristic evaluation results by applications

A Application Launcher, NE login, Personal settings, Active Sessions, and Remote Launcher.

Application Launcher	**Severity**				
Heuristic	**0**	**1**	**2**	**3**	**4**
Feedback		X			
Speak the users' language					
Clearly marked exits and user control		Q			
Consistency and standards			X		
Prevent errors			Q		
Minimise user memory load					
Shortcuts, flexibility		X			
Use simple and natural dialogue					
Good error messages			Q		
Help and documentation			Q		

Q: The average value of estimations is in this level, but the number of estimations too low (<3).

Comments

- Status of the connection is missing.
- "Save As" dialogue should be directed to cache path instead of application path
- Window opens in a wrong place on the screen.
- Testees suggest personalisation feature
- Some fixes have to make in help. Need some pictures, which could clarify text, and some part of help are too childish.

NE Login	Severity				
Heuristic	**0**	**1**	**2**	**3**	**4**
Feedback			X		
Speak the users' language	X				
Clearly marked exits and user control		X			
Consistency and standards	X				
Prevent errors		X			
Minimise user memory load			Q		
Shortcuts, flexibility	X				
Use simple and natural dialogue	X				
Good error messages		Q			
Help and documentation				X	

Q: The average value of estimations is in this level, but the number of estimations too low (<3).

Comments

- There is no information about which application login this is.
- ”Error” dialogues does not have OK button as a default.
- It might be good if there are previously visited servers and previous login names in drop down lists.
- Application close, if user name or password is typed wrong.

Active sessions	**Severity**				
Heuristic	**0**	**1**	**2**	**3**	**4**
Feedback		X			
Speak the users' language		X			
Clearly marked exits and user control		Q			
Consistency and standards			Q		
Prevent errors			Q		
Minimise user memory load		X			
Shortcuts, flexibility		X			
Use simple and natural dialogue		Q			
Good error messages		Q			
Help and documentation		Q			

Q: The average value of estimations is in this level, but the number of estimations too low (<3).

Comments

- Help is poor.
- Reordering of fields functionality is missing.
- Bad procedure on terminating sessions feature.
- Dialogues are not consistent.
- Application status information is missing.

Personal settings	**Severity**				
Heuristic	**0**	**1**	**2**	**3**	**4**
Feedback			Q		
Speak the users' language		Q			
Clearly marked exits and user control	X				
Consistency and standards			Q		
Prevent errors			Q		
Minimise user memory load		Q			
Shortcuts, flexibility		Q			
Use simple and natural dialogue		Q			
Good error messages				X	
Help and documentation		Q			

Q: The average value of estimations is in this level, but the number of estimations too low (<3).

Comments

- Error messages need to check
- Layout of dialogue is not as described in Nokia Look & Feel guide.
- Help is missing.
- "User name" should as a label field of dialogue.

Remote Launcher	**Severity**				
Heuristic	**0**	**1**	**2**	**3**	**4**
Feedback			Q		
Speak the users' language	X				
Clearly marked exits and user control		X			
Consistency and standards		X			
Prevent errors			X		
Minimise user memory load			Q		
Shortcuts, flexibility		Q			
Use simple and natural dialogue			Q		
Good error messages		Q			
Help and documentation		Q			

Q: The average value of estimations is in this level, but the number of estimations too low (<3).

Comments

- Inappropriate "key" icon.
- No information of the name of the servers.
- Help is missing.
- No information of application of the remote server.

MMI Window	**Severity**				
Heuristic	**0**	**1**	**2**	**3**	**4**
Feedback		X			
Speak the users' language					
Clearly marked exits and user control		Q			
Consistency and standards			X		
Prevent errors			Q		
Minimise user memory load					
Shortcuts, flexibility		X			
Use simple and natural dialogue					
Good error messages			Q		
Help and documentation			Q		

Q: The average value of estimations is in this level, but the number of estimations too low (<3).

Comments

- Status of connection is missing.
- "Save As" dialogue should be directed to cache path instead of application path.
- Window opens in a wrong place on the screen.
- Tool tips are different from menu texts.
- Wrong titles in error message dialogues.
- More detailed help.
- "Ok" button is not default in dialogues.

Action Log Viewer	**Severity**				
Heuristic	**0**	**1**	**2**	**3**	**4**
Feedback			X		
Speak the users' language			Q		
Clearly marked exits and user control		Q			
Consistency and standards		Q			
Prevent errors					
Minimise user memory load					
Shortcuts, flexibility			Q		
Use simple and natural dialogue		Q			
Good error messages					
Help and documentation		Q			

Q: The average value of estimations is in this level, but the number of estimations too low (<3).

Comments

- Status of connection is missing.
- Titles are missing.
- Menu items are not the same than title of dialogue.
- Refreshing does not work properly.
- Bad icons or icons are missing.
- Menus are not in right order or too deep.
- Wrong components in menu, radio button-> check box.
- All mnemonics are missing.
- Default directory should be cache directory.

Setting dialogue	Severity				
Heuristic	**0**	**1**	**2**	**3**	**4**
Feedback					
Speak the users' language					
Clearly marked exits and user control					
Consistency and standards			Q		
Prevent errors					
Minimise user memory load			Q		
Shortcuts, flexibility			Q		
Use simple and natural dialogue					
Good error messages					
Help and documentation					

Q: The average value of estimations is in this level, but the number of estimations too low (<3).

Comments

- Text fields need tool tips.
- Title of dialogue is misleading or it is quite hard understand.
- Help is missing.
- Dialogue needs redesign.

Filter Events dialogue	**Severity**				
Heuristic	**0**	**1**	**2**	**3**	**4**
Feedback					
Speak the users' language			Q		
Clearly marked exits and user control			Q		
Consistency and standards			X		
Prevent errors					
Minimise user memory load		Q			
Shortcuts, flexibility			Q		
Use simple and natural dialogue			Q		
Good error messages					
Help and documentation					

Q: The average value of estimations is in this level, but the number of estimations too low (<3).

Comments

- "Find" functionality is missing.
- Text "Clear" should be "Default".
- Title of dialogue is wrong
- Mnemonics does not work properly.
- Dialogues layout is incoherent and there is used too many different font.

Detail dialogue	**Severity**				
Heuristic	**0**	**1**	**2**	**3**	**4**
Feedback					
Speak the users' language			Q		
Clearly marked exits and user control					
Consistency and standards					
Prevent errors					
Minimise user memory load					
Shortcuts, flexibility		Q			
Use simple and natural dialogue					
Good error messages					
Help and documentation					

Q: The average value of estimations is in this level, but the number of estimations too low (<3).

Comments

- If data not found dialogue closes, annoying feature.
- Mnemonics does not work.
- Buttons are logically in a wrong order.

Version Viewer	**Severity**				
Heuristic	**0**	**1**	**2**	**3**	**4**
Feedback				Q	
Speak the users' language				Q	
Clearly marked exits and user control				Q	
Consistency and standards			Q		
Prevent errors	X				
Minimise user memory load			X		
Shortcuts, flexibility		Q			
Use simple and natural dialogue	X				
Good error messages				X	
Help and documentation		Q			

Q: The average value of estimations is in this level, but the number of estimations too low (<3).

Comments

- Icons are not good.
- Text "Find file" should be "Find".
- "About" dialogue is not as Nokia Look & Feel says.
- Status of application is missing.
- Command button is not disabled even that feature is not possible.

•

Get Version of Single File Dialogue	Severity				
Heuristic	**0**	**1**	**2**	**3**	**4**
Feedback					X
Speak the users' language					X
Clearly marked exits and user control					X
Consistency and standards					X
Prevent errors					X
Minimise user memory load					X
Shortcuts, flexibility					X
Use simple and natural dialogue					X
Good error messages					X
Help and documentation					X

Q: The average value of estimations is in this level, but the number of estimations too low (<3).

Comments

- This is futile dialogue, if this dialogue has not any other meaning.
- This dialogue has been removed after usability test.

Find Dialogue	**Severity**				
Heuristic	**0**	**1**	**2**	**3**	**4**
Feedback			Q		
Speak the users' language					Q
Clearly marked exits and user control			X		
Consistency and standards			Q		
Prevent errors	X				
Minimise user memory load	X				
Shortcuts, flexibility		X			
Use simple and natural dialogue		X			
Good error messages			Q		
Help and documentation	Q				

Q: The average value of estimations is in this level, but the number of estimations too low (<3).

Comments

- Error message goes into the other window
- Cursor does not change.
- Wrong text, “Find file” should be “Find” or “Find String”.

Properties Dialogue	Severity				
Heuristic	**0**	**1**	**2**	**3**	**4**
Feedback	Q				
Speak the users’ language	X				
Clearly marked exits and user control	X				
Consistency and standards	X				
Prevent errors	X				
Minimise user memory load	X				
Shortcuts, flexibility		Q			
Use simple and natural dialogue	X				
Good error messages	X				
Help and documentation	Q				

Q: The average value of estimations is in this level, but the number of estimations too low (<3).

Comments

- Help is incomplete.
- There should be command buttons instead of menu items.
- Fields does not adjust the size of dialogue.

Settings dialogue	**Severity**				
Heuristic	**0**	**1**	**2**	**3**	**4**
Feedback	**Q**				
Speak the users' language	**X**				
Clearly marked exits and user control			**X**		
Consistency and standards	**X**				
Prevent errors	**Q**				
Minimise user memory load	**X**				
Shortcuts, flexibility			**Q**		
Use simple and natural dialogue	**X**				
Good error messages	**X**				
Help and documentation	**Q**				

Q: The average value of estimations is in this level, but the number of estimations too low (<3).

Comments

"Default", "Select All", and "Clear All" buttons are missing.

Nemu Log Analyzer	Severity				
Heuristic	**0**	**1**	**2**	**3**	**4**
Feedback	X				
Speak the users' language	X				
Clearly marked exits and user control	X				
Consistency and standards		Q			
Prevent errors	X				
Minimise user memory load	X				
Shortcuts, flexibility	X				
Use simple and natural dialogue	X				
Good error messages	X				
Help and documentation		Q			

Q: The average value of estimations is in this level, but the number of estimations too low (<3).

7.8 Appendix 8 The Results of Cognitive walkthrough

This method tried to find out things and features that do not help using or learning of system.

Application Launcher

- Progression of event does not see clearly, maybe 25% of users do not notice progression.
- Some users, less than 25%, might use the right button of mouse to choose next action.
- There is no any kind of indication of progression on mouse cursor.

NE login dialogue

- Annoying behaviour. The windows should no be closed if user have typed name or password wrongly.

Personal Settings dialogue

- Title might be as a “Changing password”, because this dialogue does not include anything else. How about the name ”Changing personal settings”?
- If the users objective is change password, this title does not give any clue for decision-making.

Active Sessions dialogue

- There should be a ”Refresh” feature, because users aim is to check real active sessions. This dialogue might have been open quite long time.

Action Log Viewer

Main window

- Some of the names (command buttons and menus) are misleading; user does not combine text and action behind the text.
- Shortcuts are quite exotic, users might not notice they.
- Name “Detail” could be as an “Event detail”.

- If the previous view have been “View -> Log by Type” and user want to view all events then the condition have to change via path “View -> Log by Type -> All events”. This does not be as a clear structure.
- The contrast of the colours of ”timer bar” is too narrow. This might be a problem with LCD’s.

”Setting” dialogue

- Choosing the right action is not obvious on the first time, but it does not bother next time. Help might be helpful, but it is not applicable.
- If user makes any changes then popup window appears. In this window says that: “Changes you made are effective when applications that belongs to authority domains that you are restarted in the server”. This might be very confusing especially the part “restarted in the server”. There is also a severe problem; this popup dialogue does not have any kind of cancel feature.

”Filter Events” dialogue

- Text ”Source” and ”User” might be unclear, because the content of those features is not unambiguous. Below the “Source” button is one empty field. Below the “User” button is text ”N/A”.
- User can change status of check boxes elsewhere or in this dialogue. This is quite confusing. It might be logically if those changes can make only one place.
- The structure of view menu is too deep.
- Sorting orders of logs should be as an alphabetical, Date & Time (as a default), Log and User orders.

Version viewer

Main window

- Reordering of fields can use in one direction.
- System does not give any kind of information of its status.

“Find” dialogue

- If user does not give any condition on “Find” field then find feature make symbolic search and does not give anything as a result.
- ”Get version of single file” feature does not give appropriate error messages.
- “Find next” feature is missing.

7.9 Appendix 9 Cross-reference heuristic evaluation vs. Ravden & Johnson method

Application Launcher

Evaluated property	Description of the problem	Heuristic evaluation	Ravden & Johnson method
Status of application	Status of exception is not visible	X	
	Wrong information in status bar		X
	No information of server and status of connection	X	X
	No hourglass cursor	X	X
	Waiting times are too long, no indication of progress	X	
	Status bar could be bigger font.		X
Menus	Mnemonics are missing or they are wrong	X	X
Icons	Icons are too small		X
Error messages	Have to be clear and logical	X	
	Wrong type ”error” and “information” dialogues	X	X
	Some typos		X
Features of main window	Personification feature is needed	X	
	Exception procedures does not work properly	X	
	User feedback have to be ambiguous	X	
	Wrong user name or password closes whole application	X	X
	No default values	X	
	Scroll bar is missing		X
	Where is the “Search”?		X
Features of dialogues	Default buttons does not react keyboard	X	X
			(continues)

Application Launcher (continued)			
	Futile resizing option	**X**	
	Mnemonics does not work properly	**X**	**X**
	Dialog is too big or it is in wrong place on main window	**X**	**X**
	Some dialogues are too fast	**X**	**X**
	Different “Close” -Button		**X**
	Wrong colour in buttons		**X**
Guidance	Lack of approriate information	**X**	**X**
	Some mistakes on help		**X**
Common problems	Style have to be the same around the application	**X**	
	Better grouping of information	**X**	**X**
	Some font are wrong	**X**	**X**
	The form of the user input have to defined somewhere (IP or net name)	**X**	**X**
	Shade of colours are too close each others	**X**	**X**
	Unification of terms and abbreviations	**X**	**X**

Version Viewer

Evaluated property	Description of the problem	Heuristic evaluation	Ravden & Johnson method
Icons	"Refresh" look like "Undo"	X	X
Menus	Menu items are not the same than titles of dialogue.	X	
Layout of dialogues	”About” dialogue is faulty	X	
	"Settings" dialogue: Check boxes must be at white background. Now they look like disabled.	X	X
	Check boxes in "Find" and "Settings" –dialogs are in different order.	X	X
Content of dialogues	Too many check box.	X	
	Wrong title on dialogue	X	
Features of dialogues	"Settings" dialogue: "Select all" button is missing.	X	
	Find does not give any clue to typing values, especially "true/false" –values.	X	X
	Wrong title in field	X	
	”Find” starts finding without conditions.	X	X
Error messages	(Error) Messages are showed only in the status bar	X	X
	”File not found” –message unclear.	X	
	Error message is missing.	X	
	If ”Find” does not return anything then previous message does not disappear.		X

(continues)

Version Viewer (continued)			
	Status is missing in status bar. Is application alive?	**X**	**X**
Components of the main window	Cursor does not change during the refresh.	**X**	**X**
	Status bar have wrong text sometimes	**X**	
Features of main window	Horizontal scroll bar is missing.	**X**	**X**
	Colours are too same in tabs.	**X**	
	User cannot save a certain view to be as a default view. View is always the same at the start	**X**	**X**
	Futile information	**X**	
	Text fields do not change they size when window is resized.	**X**	
Guidance Common problems	“Copy” does not work.		**X**
	Lack of guidance	*	*
	Enter key does not work as assumed.	**X**	
	Dialogue covers the menu	**X**	
	Hierarch –view tree structure unclear.		**X**
	Automated sorting work too automated	**X**	**X**
	“Settings” need speed buttons “select all”/”deselect all”/”default values”	**X**	**X**

* No estimations. Help does not exist.

Proposals to developing

- “Find” could be “Find string/file”
- Redesign functionality of ”Find”.

Action Log Viewer

Evaluated property	Description of the problem	Heuristic evaluation	Consistency Inspection
Status of the application	Opens inactive	X	
	No information about the log	X	
	Status bar almost useless	X	
	Data refreshing does not work properly	X	
Menus	Menu item <> title of dialogue	X	
	Wrong order	X	
	No Mnemonics or they are wrong	X	
	No bold font face in menu		X
	”Log” text must replace with ”File” text		X
	No radio buttons in menu	X	X
Icons	Could they be as button	X	
	”Refresh” not clear	X	
	Icons are not mostly appropriate	X	
	Some menu bar icon is missing and some menu item is missing	X	X
Layout of the dialogues	Some mnemonics is wrong or missing	X	
	Command button have wrong text “clear”-> “default”	X	
	Long text field does not work, try “Tool tip” feature	X	
	Some mistakes in text	X	
	”Filter” dialogue: Left justified		X

(continues)

Action Log Viewer (continued)			
Features of the dialogues	Wrong background colour in combo box		**X**
	Must be (de)select all button	**X**	
	Buttons logically upside down	**X**	
Common problems	Default folder?	**X**	
	The structure of application does not give any hints to usage -> Use ”tool tips”	**X**	
	Titles of windows are wrong or missing	**X**	
	There is no ”find” functionality	**X**	
	Table needs "air”		**X**
	Remove titled borders		**X**
	Application icon have to add on title bar		**X**

MMI Window

Evaluated property	Description of the problem	Heuristic Evaluation	Consistency Inspection
Status of the application	Status of the connection is not available	X	
Features of the main window	Main window opens in random place of the screen	X	
	Main window opens in random size	X	
	Wrong components in main window		X
	Ctrl-C does not work	X	X
Menus	Menu items are not the same than the title of dialogue	X	
	Menus have a wrong background colour.		X
	Separators are missing in File and Help menus		
	Mnemonic of the "S<u>a</u>ve" is a		X
	"Help on This Window" has a wrong icon		X
Layout dialogues	Mnemonics are wrong or they are missing	X	
	"About dialog" is wrong type	X	
Guidance	Help need some information	X	
Common problems	Letters capital or normal	X	
	Command letters have echo into the screen	X	
	Nokia bar is missing		X
	No defaults folder How about "cache" folder?	X	

7.10 Appendix 10 User feedback

Pronto number	**Severity**	**Topic of reported fault**	**Correction or answer**
P1040105	B Major	ApplicationLauncher: Error-box is unreachable	There was a problem with a thread conflicting with the UI in the session service.
P2205716	C Minor	MML FREEZES	This error occured with all MML- commands that did not fit in a single line. When MMI- system sent a command for MmiWindow to move cursor one step downwards, MmiWindow did not check if such line is already created in it´s text area. Because of this MmiWindow tried to write characters outside of the allocated data area, and data buffer was wrapped over. , Now MmiWindow always checks if necessary data area is already allocated before moving cursor.
P2533016 P2671116	C Minor	Cannot open Application Launcher (problem has occurred since TS4.1)	Unable to reproduce error situations found in the log files., Log files were from a long period of time. Some errors occurred only once. Clearly sometimes everything worked fine. Therefore we were unable pinpoint any error that would cause the need for resetting NEMU server.
P2627616 P2711616	B Major	NEMU in Restart Loop	Nemu was in restart loop, because NemuAlarm failed at startup. NemuAlarm failed because NemuEMTResourceManager crashed. Crash was caused by corrupted temp file of NemuEMTResourceManager. Startup of NemuAlarm fixed so that if some subcomponent of NemuAlarm fails that does not cause restart loop. NemuEMTResourceManager checks now if data in temp file is valid., Nemu was in restart loop, because NemuAlarm failed at startup. NemuAlarm failed because NemuEMTResourceManager crashed. Crash was caused by corrupted temp file of NemuEMTResourceManager. Startup of NemuAlarm fixed so that if some subcomponent of NemuAlarm fails that does not cause restart loop. NemuEMTResourceManager checks now if data in temp file is valid
P2753016	C Minor	Service-Terminal is not supported by MMI-Window	This is not a fault, but more a specification issue. Since the beginning it has been specified that MmiWindow does not support service terminal session. If this feature seems to be necessary anyway, Nokia should be informed about the issue, and maybe service terminal support can be implemented in the forthcoming releases.

(continues)

User feedback (continued)			
P2885816	C Minor	Can't connect to object browser We can't connect to object browser, mml window and user authority manager of nemu via application launcher. all other points are possible to connect. all service which are available are running	P2885816: correction not needed. Reason: Login password was found to be expired by customer.
P2950816	C Minor	Query regarding testing Help functionality When I do Help/Index in AL's Remote Upgrade I get an error box saying: "Open Help failed com.nokia.platypus.help.HelpException: Browser launch failed" If I then click on the link to 'Product Documentation', NED fails to appear and I get the 'Could not connect to Jrun Server' error message again. Interestingly, the User Authority Manager also won't work - this time the error message reads "An unexpected error has occured while opening UAM". Event Viewer shows a large number of yellow error logs concerning DHCPServer which all say "The DHCP server issued a NACK to the client (0002A5B591A3) for the address (10.249.74.20) request".	P2950816: correction not needed. Reason: This fault report contains quite a few errors which do not relate to each other. Therefore no-one seems to be willing to take responsibility of the fault. Please create a fault report for each fault found. If we create fault reports for each cases, the information requests will also be directed to us. Error for using RemoteUpgrade help should be directed to group responsible for RemoteUpgrade. Error with JRun and NED should be directed to group responsible for NED. Error with UserAuthorityManager can be directed to us, E-TEJN group. Error with DHCPServer should be directed to group responsible for DHCPServer. Furthermore, Application Launcher should not be installed to server itself and that configuration is not supported. Java Runtime Environment for Application Launcher may clash with JRun environment.
(contunues)			

User feedback (continued)			
P3124816	B Major	NEMU: Can't connect to Jrun server when you want to co RNC: NEMU	JRun configuration will be corrected to NM1 4.14-1 platform build, JRun configuration of NEMU will be updated as instructed by Macromedia Inc. to prevent JRun services shutting down when user logs off from Windows NT / Windows 2000. JRun servers are terminated during log off because of fault in Java Virtual Machine version used in NEMU. This fault has been confirmed by Macromedia Inc. Until permanent fix has been made, you can enable JRun again by logging on to system and starting "JRun Admin Server" and "JRun Default Server" services. Also restarting NEMU will restart JRun services.
P602141	B Major	Remote Upgrade application do not open	Remote Upgrade is not part of this release.