Sari Leino-Huagie

USER REQUIREMENTS IN HUMAN-CENTRED DESIGN
CASE: OMT++

Master Thesis
Computer Science and
Information systems
26.4.2001

University of Jyväskylä
Department of Computer Science and Information Systems
Jyväskylä

ABSTRACT

Leino-Huagie, Sari Päivikki
User Requirements in Human-centred design – Case: OMT++ / Sari Leino-Huagie.
Jyväskylä: University of Jyväskylä, 2001.
77 p.
Master Thesis

Systems and processes are most useful and usable when their context of use is fully considered. Human-centred design is a way of creating new technologies and systems by looking at the relationship between technologies, people and environments. It is at the core of user acceptance of a new product or process. In simple terms, human-centred design is the extent to which users find a product or process useful and usable. Understanding user requirements is an integral part of information systems design and is critical to the success of projects.

This thesis gives an outline to the usability and human-centred issues concentrating more fully on user requirements. The theoretical part deals with assessment criteria on the basis of user requirements specification framework. In the case study the assessment criteria is used to evaluate human-centredness of OMT++ development method. The main focus is on early requirements collection and its congruence between human-centred approach and OMT++.

According to the findings of this study the created assessment criteria can direct the evaluation of human-centredness of the method and is capable of pointing out the missing links of the selected method in user requirements collection. Furthermore, it noticed the areas in the method, which need the development. Even if the human-centred approach need more work, it will give better results, which can be seen as more satisfied users.

KEYWORDS: human-centred design, user requirements, OMT++, usability, ISO 13407, ISO 9241-11

TABLE OF CONTENTS:

1. INTRODUCTION 1

2. DEFINITION OF MAIN CONCEPTS 5

2.1 Information systems and HCI 5
2.2 Human-computer Interaction (HCI) 6
2.3 Usability Engineering 8
2.4 Human-centredness 9
2.5 Usability 11
2.6 Usability testing 14
2.7 Summary 16

3. QUALITY IN USE AND USABILITY IN HUMAN-CENTRED DESIGN 18

3.1 Quality in use 18
3.2 Usability in design process 21
3.3 Does usability really matter? 22
3.4 Summary 23

4. REQUIREMENTS IN SYSTEM DEVELOPMENT 24

4.1 Requirements Engineering (RE) 24
4.2 User Requirements 26
4.3 Summary 30

5. HUMAN-CENTRED DESIGN APPROACH AND USER REQUIREMENTS FRAMEWORK 31

5.1 Human-centred design approach 31
5.2 The human-centred process plan 33
5.3 Overview of human-centred design 35
5.4 Human-centred design activities 36
5.5 The framework of user requirements in human-centred design 41
5.6 Summary 46

6. GENERAL OVERVIEW OF OMT++ METHOD 48

6.1 Overview of the OMT++ process 49

6.2 Static and Functional Paths 50
6.4 Feasibility study 52
6.5 Requirements capture and use cases 53
6.6 Summary 56
7. ASSESSMENT OF THE EARLY REQUIREMENTS COLLECTING IN OMT++ 57
7.1 User context and early design 57
7.2 Prototype and user testing 60
7.3 User requirements documentation 62
7.4 Summary 63
8 EVALUATION, DISCUSSION AND CONCLUSIONS 65
8.1 Evaluation of assessment criteria 65
8.2 Discussion 66
8.3 Conclusions 67
REFERENCES 70

1. INTRODUCTION

Over the years, it has been acknowledged that the standard way of representing user requirements in the functional requirements specification document is often inadequate. When developing computer based information system, it is necessary to understand how the end-users perform their work and interact with the environment. It is also necessary that the users' requirements are described in a way that all participants can comprehend.

However, specifying the user requirements is not so simple to achieve. How can a system be designed before the future situation is known? End-users may not appreciate the benefits that future system can offer them, and the design team may find it difficult to integrate user opinion into the design process, and thus may concentrate only on the technical requirements of the system (Maguire, 1997). It seems that general opinion among the designers is still that users don't know what they want, and when they do, they all want something different.

In general, systems analysis methods are suitable for developing several components of the information system, but they are not sufficient to meet the needs of the user interface desinger. The essential principles of human-centred design are to make user issues central in the design process, to carry out early testing and evaluation with users and to design iteratively.

Methodological approaches to usability engineering have been evolving since the 1970s. An early reference to a usability engineering methodology was offered by Gould and Lewis (1985). According to these researchers, any system designed for people to use should be easy to learn, useful and be easy and pleasant to use. They recommended three principles of design: early focus on users and tasks, empirical measurement and iterative design (ibid.). According to Gould and Lewis (ibid.) these principles are argued to be obvious by designers. However, the principles are seldom applied in system design even if

it is indicated that the principles can lead to usable systems (ibid.). Concerning the issue of users and tasks, it seems that 15 years later the situation has not changed much. According to Kramer et al. (2000) current methods of software system design do not adequately capture the needs and values of end users. Furthermore, it seems that although designers argued already in the mid-80's that it is obvious that early focus should be on users and tasks, two decades later, Karat et al. (2000) reminds us about this same issue that success in design is based on understanding users, their tasks, and the context.

Since Gould and Lewis (1985), others have offered general frameworks for usability engineering (e.g. Nielsen, 1992; Mayhew, 1998; Faulkner & Culwin, 2000). In 1999 International Standard Organisation (ISO) published its first edition from a standard called "Human-centred design processes for interactive systems" (ISO 13407, 1999). Before that, in 1998, there was already a standard for usability called "Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11: Guidance on usability" (ISO 9241-11, 1998). Both standards focus on making systems usable. ISO 13407 addresses development process and ISO 9241-11 more detailed human factors issues.

These publications show that although very necessary, gathering information on user requirements is not a simple and easy task to achieve because it requires co-operation with the actual users of the system who may not necessary be programmers, but may be experts of their own field and language. Therefore, to study user requirements and human-centred design we are confronted with questions such as:

- **How can human-centred design approach improve a developing process when collecting user requirements?**

In addition to the major question, the study will aims of answering the follwing sub-questions:

❖ What does human-centredness mean in design process?

- Does the requirements phase of OMT++ design method fulfill human-centred design criteria?

As shown from the central questions idenfied, therefore, the general aim of this thesis is concerned with issues related to user requirements in human-centred design with focus on the above questions.

In order to achieve the aims of this study, I will make use of earlier research in the area of human-computer interaction. I will apply the ISO 13407 standard (1999), which describes the principles of human-centred design and gives general guidelines for human-centred design process. In this thesis, I will use user requirements framework developed by Maguire (1997) based on ISO 9241-11 and ISO 13407.

The study will concentrate on a single case study related to the methodology of OMT++ (Jaaksi, 1997; Jaaksi, et al., 1999). The goal is to find out the extent to which OMT++ supports human-centred design principles in requirements capture. Because of the limited scope of my thesis, I have future outlined the objective of the study only to requirements phase and especially user requirements. The reason for selecting this phase is mainly due to the fact that it is the first and crucial one in the development process and the importance of user requirements is often overlooked.

The major topic of this thesis is about how human-centred design process helps specify user requirements in system development. Chapter 2 gives an outline on basic concepts used in this thesis. Chapter 3 describes quality and usability issues in human-centred design. Chapter 4 describes the general requirements point of view in system development focusing specifically on user requirements. Chapter 5 outlines human-centred design approach and ISO 13407 standard. The used framework is described more fully in this chapter. In chapter 6 I will deal with the case study, whereby I will concentrate on one specific method called OMT++ and its characteristics when dealing with user requirements.

Chapter 7 contains assessment of OMT++ method and especially its requirements collecting phase. Chapter 8 evaluate the assessment criteria and will put forward the discussion and conclusions.

2. DEFINITION OF MAIN CONCEPTS

The target of this research is user requirements and human-centred design, both of which consist of various aspects of user issues. In order to fully understand the complexity of user requirements and human-centred design this chapter gives background information on the basic concepts related to user issues.

2.1 Information systems and HCI

Information systems (IS) research field arose in the late 1960s as mainframe systems proliferated. Management interest in understanding how best to use computers led to an academic focus divided between computer science and business or management schools. Human-computer interaction (HCI) field arose much later in the early 1980s as multi-tasking minicomputers and personal computer systems became widespread. Interest in personal information processing led to an academic focus divided between computer science and experimental psychology. As Grudin (1992) noted concepts, like 'system', 'user', 'task', and 'task analysis' have different meanings in the IS and HCI fields, which makes communication across fields difficult even when this is recognized. However, Ehn and Löwgren (1997) has pointed out that HCI and IS development have evolved from similar backgrounds, via largely independent paths, to a common point of intersection.

Most traditional methods for systems development are based on a waterfall model. Analysis, design, coding, testing and maintenance are all performed in a sequence. A disadvantage with such a model is that the users are not able to test the system until it is complete. Usability problems are discovered late in the process, whereas the possibilities for improvements are limited. Instead of a waterfall model, a spiral model is suggested where analysis, design and evaluation are performed in an iterative way. Problems with the system can then be discovered early in the process, making it easier to remedy mistakes

and make changes.

Software design is concerned with the form and function of a software system and with the structure of the process that produces that system. Software engineers believe that most of their work lies in the process that generates a system having the form and function specified by the customer. The standard engineering design process offers little to connect the actions of designers with the concerns of users. Human-centred design of computer systems is based on understanding the domain of work in which people are engaged and in which they interact with computers, and programming computers to facilitate human action (Denning & Dargan, 1996).

Methods for systems development can be divided into function-oriented and object-oriented methods. Today, object-oriented methods and techniques are common in systems development Object-Oriented Software Engineering (OOSE), as described by Jacobson and his colleagues (1992), has a human-centred philosophy. Despite its user-centred ideas, many fundamental usability goals are still not addressed in OOSE. Recent work in the field of usability engineering (Mayhew, 1999; Rosson & Carroll, 1995) has attempted specifically to integrate usability engineering techniques with object-oriented design and development.

2.2 Human-computer Interaction (HCI)

During the technology explosion of the 1970s the notion of the user interface became a general concern to both system designers and researchers. Computer companies became aware that if they could somehow improve the physical aspects of the user interface they would get an advance from this on the markets. This new dimension was exploited by calling a system "user-friendly". In practice, this often meant tidying up the screen displays to make them more aesthetically pleasing. According to Preece et al. (1994) academic researchers, in contrast, were concerned about how the use of computers might

enrich the work and personal lives of people. Later on other aspects of users, like training issues, working practices, management and organizational issues, included to the field, which described by the term human-computer interaction (HCI).

Human-computer interaction or computer-human interaction (CHI) is an interdisciplinary area concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them. It is emerging as a specialty concern within several disciplines, each with different emphases:

- computer science (application design and engineering of human interfaces),
- psychology (the application of theories of cognitive processes and the empirical analysis of user behavior),
- sociology and anthropology (interactions between technology, work, and organization), and industrial design (interactive products). (ACM SIGCHI, 1992).

Human-computer interaction is concerned with

- the joint performance of tasks by humans and machines;
- the structure of communication between human and machine;
- human capabilities to use machines;
- algorithms and programming of the interface itself;
- engineering concerns that arise in designing and building interfaces;
- the process of specification, design, and implementation of interfaces, and design trade-offs.

Human-computer interaction thus has science, engineering, and design aspects (ACM SIGCHI, 1992). The aim of most efforts by HCI is to achieve more usable computer systems (Ehn & Löwgren, 1997). Since HCI has more scientific approach it needs also practical side in order to get studied practices to function.

2.3 Usability Engineering

More practical side of the HCI represents usability engineering. This concept refers to the entire process of producing usable products. However, there are different opinions about this definition. For example Nielsen (1993) refers to usability engineering as more of a process of evaluation and redesign. Additionally, Mayhew (1999) defines usability engineerig as an interdiscipline that provides structured methods for achieving usability. Most of all usability engineering is concerned about engineering and design.

Characteristic for usability engineering is that it is iterative design and evaluation process and stresses the importance of user involvement and feedback. In other words it is based on human-centred approach.

In software engineering (SE) the focus is on understanding, controlling, managing and improving software products and processes based on engineering principles. In usability engineering the focus is on understanding the usability of software products, deriving criteria for "good" human-computer interfaces, and devising methods and tools for designing and implementing such interfaces.

The historical development of the measurement practices used in the two disciplines has different origins with different perspecitves. The need to assess and predict the performance of software per se was recognized much earlier than the importance of its usability. Consequently, software engineering measurement techniques have been in existence longer and the discpline is more advanced than usability evaluation. Quantitative measurement tends to predominate in SE, whereas qualitative techniques as well as quantitative ones are common in HCI (Preece & Rombach, 1994).

2.4 Human-centredness

According to Ainger et al. (1995) human-centredness is not a concept that can be defined in absolute terms. It has developed in different European countries, combining local cultural traditions with a reaction to Taylorism. As Ainger et al. (1995) points out in designing and implementing computer systems the goal should be to enhance human skills and abilities, rather than to attempt to replace them. This is the basis of the human-centred process. Human-centred is a term better applied to processes than to systems. A tool is not human-centred by virtue of possessing a high level of interaction. Much depends on how the tool is used.

Gill (1996) points out that the human-centred tradition provides theoretical and methdological frameworks for the social and cultural shaping of technologies emphasising human-machine symbiosis, creativity and innovation, participatory and cooperative design, and the tacit dimension of knowledge. These issues are very much part of the human-centred debates whose origins lie in the European human-centred movements of the 1970s.

Over the years the human-centred tradition has built upon the socio-technical approach and has shaped the human-factors approaches towards user-centred, user-involved, and cooperative design approaches (Gill, 1996). The idea of human-machine symbiosis (Figure 1) emphasises the collaboration between human capabilities and machine, rather than the separation, which is embedded in the tradition of HCI (Figure 2). The user-centred approach (Figure 3) transcends the linear relationship between the human and the computer, and argues for the tripartite interrelationships between the human, the computer and the job.

The human-centred approach, is, however, limited by its focus on user practice and user modelling which focuses on a narrow view of human activity, and on the separation between the user and the designer of technological systems. From the human-centred perspective the crucial limitation of the human-centred

approach is its neglect of the social setting and culture of the workplace. The user-involved approach (Figure 4) fills this gap by emphasising the multiplicity of relationships between the social system, the technology and organisational systems.

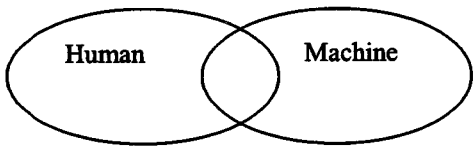

FIGURE 1. Human-machine symbiosis: collaboration (Gill, 1996, p. 3).

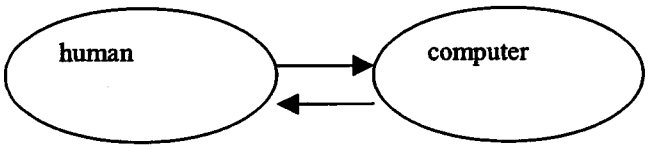

FIGURE 2. Human-computer interaction: separation (Gill, 1996, p. 3).

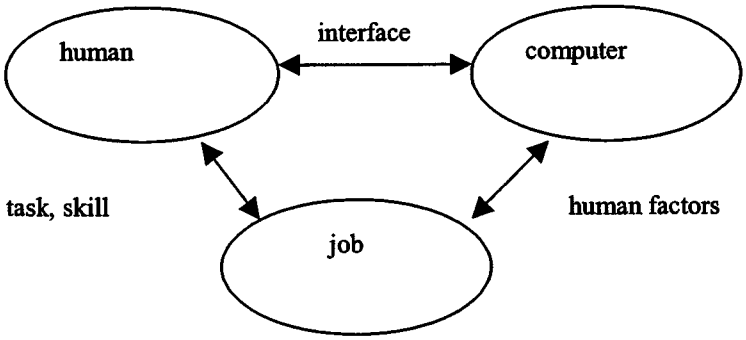

FIGURE 3. User-centred approach: user practice, user model (Gill, 1996, p. 3).

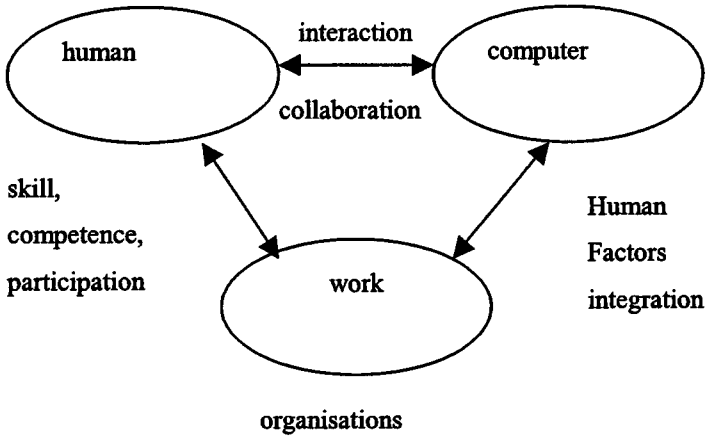

FIGURE 4. User-involved approach: social settings, organisational culture (Gill, 1996, p. 4).

These human-centred approaches provide theoretical and methodological frameworks for designing information and communication technologies (Gill, 1996).

2.5 Usability

"Building usability into a system requires more than knowledge of what is good. It requires more than an empirical method for discovering problems and solutions. It requires more than support from upper management and an openness on the part of developers. It even requires more than money and time. Building usability into a product requires an explicit engineering process. That engineering process is not logically different than any other engineering process. It involves empirical definition, specification of levels to be achieved, appropriate methods, early delivery of a functional system, and the willingness to change that system. Together these principles convert usability from a 'last minute add on' to an integral part of product development. Only when usability engineering is as much part of software development as scheduling can we expect to regularly produce products in which usability is more than an advertising claim."

Dennis Wixon and John Whiteside (1985) p. 147

There is a Finnish adage, which says "the dear child has many names." This adage can be applied to the concept of usability. The most common synonyms used for describing usability are user friendly, easy to use, availability, easy to learn, etc. However, as Nielsen (1993) has pointed out usability is only one important part of a bigger picture of system acceptability (Figure 5).

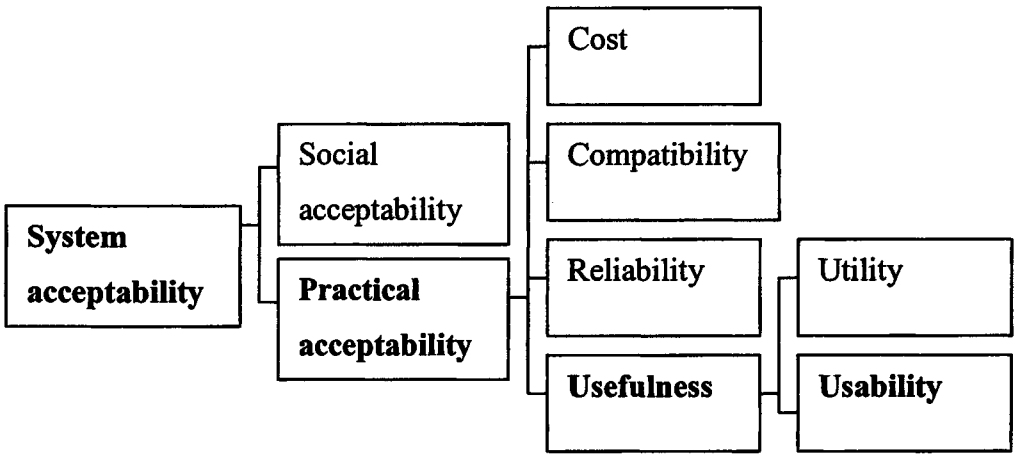

FIGURE 5. A model of the attributes of system acceptability (According to Nielsen, 1993, p. 23).

In the old days, when computer vendors realize that users can be more than an inconvenience, the term of choice to "good" system was "user friendly". As Nielsen (1993) has noted that the term "user friendly" is not appropriate. First of all, "users don't need machines to be friendly to them, they just need machines that will not stand in their way when they try to get their work done" (p. 23). Secondly, different users have different needs in the sense that a machine can be "friendly" to one user but tedious to another. Instead, Nielsen proposes the term usefulness that relates to whether the system can be used to achieve some desired goal.

Usefulness can be divided into two categories: utility and usability (Grudin, 1992). Utility corresponds to whether the needed functionality is included in the system. Usability is more closely related to how well a user could use this

functionality. As a contribution to the design process usability is most often interpreted as relating to skills in interface design which complement other design objectives such as functionality, efficiency and reliability. This is a narrow product-oriented view of usability, which suggests that usability can be designed into a product. In this sense usability is closely related to ease of use, which is probably the most common way the term is used. It is for instance consistent with Nielsen's (1993) view. In this sense one can talk about a system, which is usable but not useful. However, for this very reason, it is not a very good way to conceptualise usability. What really counts is whether a user can achieve their intended goal when they use the product (Bevan, 1995).

International Standard Organisation 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" (ISO 9241-11, 1998, p. 2). The attributes, which a product requires for usability depends on the nature of the user, task and environment. A product has no intrinsic usability, only a capability to be used in a particular context. According to Bevan (1995) usability is determined not only by the product but also by the context in which it is used. This means that there is no such thing as a "usable product" or "unusable product". For instance a product, which is unusable by inexperienced users may be quite usable by trained users (ibid.).

In ISO's broad definition effectiveness refers to the extent to which a goal or task is achieved. Indicators of effectiveness include quality of solution and error rates. Efficiency, meanwhile, refers to the amount of effort required to accomplish a goal. In the other words, efficiency is the relation between the accuracy and completeness with which users achieve certain goals and the resources expended in achieving them. Efficiency can be measured by task completion time and learning time. Satisfaction refers to the level of comfort and positive attitudes that users feel when using a system. Users' satisfaction can be measured by attitude rating scales.

According to Frøkjær, et al. (2000) the correlations between aspects of effectiveness, efficiency, and satisfaction are not well understood. They have pointed out that the correlations among the usability aspects depend in a complex way on the application domain, the user's experience, and the use context. Unless domain specific studies suggest otherwise, effectiveness, efficiency, and satisfaction should be considered independent aspect of usability and all be included in usability testing (ibid.).

The ultimate goal in software development is that the user community accepts the software. In other words, the software must be usable. According to Karat (1997) the acceptability of any system is not dependent solely on surface interface features, but on the way a system fits within a context of use.

2.6 Usability testing

To produce evidence that product is of a satisfactory quality and meets the user's expectations—in other words is usable—usability evaluation methods and heuristics can be used (Faulkner & Culwin, 2000). Usability testing and laboratories appeared in the early 1980's (Shneiderman, 1998). Experience has shown that companies found many benefits to usability testing including shortened product development time. Usability testing is an important part of the software development lifecycle in industry. The goal of most of the evaluation is to catch usability problems and/or errors in implementation of requirements as early in the software development process as possible. Problems found early in the software lifecycle are easier and less costly to correct than problems found during the operations phase.

There have been a number of trends in usability evaluation and measurement. In the 1970s and early 1980s traditional laboratory testing was predominant. Laboratory testing is valuable for examining and comparing differences in one or more independent variables. As in all kinds of testing one needs to pay attention to the issues of reliability and validity (Preece & Rombach, 1994).

Reliability of usability tests is a problem because of the huge invidual differences between test users. Validity is a question of whether the usability test in fact measures something of relevance to usability of real systems in real use outside the laboratory. Whereas reliability can be addressed with statistical tests, a high level of validity requires methodological understanding of the test method one is using. Typical validity problems involve using the wrong users or giving them the wrong tasks or not including time constraints and social influences (Nielsen, 1993).

Although the influence of usability professionals during design is often restricted to user interface issues, usability is frequetly evaluated by testing a prototype of the product with users. This leads to a number of problems. This type of human-based evaluation is often left till late in design when there is only scope to make minor changes to the user interface (Bevan, 1995). If evaluation reveals deeper problems with the functionality or basic dialogue design, this may be outside the responsibility of the usability professional.

According to Bevan (1995) the best solution to these problems is to adopt a human-centred approach to design with a continual cycle of human-based evaluation. If usability evaluation is left until just before release, there will be no chance to make any significant changes in design to correct deficiencies. In order to achieve a usable product it is important to begin the cycle of understanding, specifying and evaluating usability by using simple mock-ups at the earliest stages of design.

Most current usability evaluation practices focus on providing design feedback to improve usability. The most common methods for testing users involve some form of co-operative evaluation, requiring active intervention from an observer to probe usability problems with users.

To obtain reliable measures, the context of evaluation must closely match the intended context of usage. This means that:

- it is essential to have a complete understanding of the exact context in which the product will be used,
- it is essential to replicate the important aspects of this context for the evaluation, and
- the user should work in realistic conditions without interruption from an observer, in order to accurately replicate the intended context of use. (Bevan, 1995).

Evaluation is concerned with gathering data about the usability of a design or product by a specified group of users for a particular activity within a specified evaluations that provide answers to questions that inform design in some way and form an integral part of the human-centred design process (Preece et al., 1994). Most kind of evaluations can be described as one of the following:

- observing and monitoring users' interactions,
- collecting users' opinions,
- experiments or benchmark tests,
- interpreting naturally-occuring interactions or
- predicting the usability of a product (Preece et al., 1994).

The term heuristic evaluation came into use at the end of the 1980s. Heuristic evaluation is the most informal method and involves having usability specialists judge whether each dialogue element follows established usability principles (the "heuristics"). Heuristics are provided to help evaluators identify usability problems when inspecting a user interface design. Heuristic evaluation is a very efficient usability engineering method. However, there are major individual and group differences between the performance of evaluators (Preece & Rombach, 1994; Nielsen, 1993).

2.7 Summary

Concepts are established slowly in new areas. Human-computer interaction research and usability issues are comparatively new from a technological point

of view of information systems. However, when designing systems with user interface we have to take into consideration the differences between human and machine. Human beings are good at things that come naturally to us: representing knowledge, practicing judgement, creativity, experiencing emotions and so on. All of these are difficult for the machine. Machines are precise, repetitive and reliable by their operations. This could lead to the conclusion that if the strengths of machines are combined with human skills then many complex tasks can be accomplished. The complexity of technology results in designers spending almost all their creative energies on the technical issues, with little time and interest on the human issues. Designers' failure to implement technology, which is suitable for the human way of working have led to many disasters. The main goal should be to make tools that fit and complement human needs. This goal can only be achieved through design approach, which takes into consideration the human side of the technology use. The following chapters will outline more deeply on the human side of design.

3. QUALITY IN USE AND USABILITY IN HUMAN-CENTRED DESIGN

Generally there are two different views on usability. One is a "top-down" approach in which usability is interpreted as a quality objective and the other is a product-oriented "bottom-up" view, which identifies usability with ease of use. However, to achieve the overall objective of usability and quality in use requires a human-centred approach to design. The following is a description of these two approaches, quality in use and usability.

3.1 Quality in use

In order to evaluate software it is necessary to select relevant quality characteristics. This can be done using a quality model, which breaks software quality down into different characteristics. According to Bevan and Azuma (1997) ISO/IEC 9126 provides a general-purpose model, which defines six broad categories of characteristics of the software: functionality, reliability, usability, efficiency, maintainability and portability. These can be further broken down into subcharacteristics, which have measurable attributes.

ISO 9000 is concerned with quality assurance to provide confidence that a product will satisfy given requirements. Interpreted literally, this puts in the hands of the person producing the requirements specification. Requirements are determined by the real world, which determines what the needs are. Needs relate very strongly to quality in use.

Most approaches to quality do not deal explicitly with user-perceived quality. User-perceived quality is regarded as an intrinsically inaccurate judgement of product quality. However, there is a more fundamental reason for being concerned with user-perceived quality. Products can only have quality in

relation to their intended purpose.

Bevan and Azuma (1997) point out that software product quality can be measured internally (typically by static measures of the code), or externally (typically by measuring the behaviour of the code when executed). The objective is for the product to have the required effect in a particular context of use. Quality in use is the user's view of quality. Achieving quality in use is dependent on meeting criteria for external measures of the relevant quality sub-characteristics, which in turn is dependent on achieving related criteria for the associated internal measures (Figure 6). In other words, the internal attributes are said to be indicators of the external characteristics (Bevan & Azuma, 1997). In the same way, external subcharacteristics will influence the observed quality in use. A failure in quality in use can be traced to external sub-characteristics and the associated internal attributes, which have to be changed.

FIGURE 6. - Relationship between different types of quality (Bevan, 1997, p. 4)

ISO 14598-1 defines external quality as "the extent to which a product satisfies stated and implied needs when used under specified conditions" (ISO 14598-1, 1999, p. 2). This moves the focus of quality from the product in isolation to the satisfaction of the needs of particular users in particular situtations. The purpose of a product is to help users achieve particular goals, which leads to the definition of quality in use in ISO 14598-1 (1999) as "the extent to which a product by specified users meets their needs to achieve specified goals with effectiveness, productivity and satisfaction in specified context of use" (p. 4).

Internal metrics can be applied to a non-executable software product (such as a specification or source code) during designing and coding. The primary purpose of these internal metrics is to ensure that the required external quality is achieved. Internal metrics provide users, evaluators, testers, and developers with the benefit that they are able to evaluate software product quality and address quality issues early before the software product becomes executable.

External metrics use measures of a software product derived from measures of the behaviour of the system of which it is a part, by testing, operating and observing the executable software or system. Before acquiring or using a software product it should be evaluated using metrics based on business objectives related to the use, exploitation, and management of the product in a specified organisational and technical environment. External metrics provide users, evaluators, testers, and developers with the benefit that they are able to evaluate software product quality during testing or operation (Bevan, 1997).

According to Bevan (1997) the relationship of quality in use to the other software quality characteristics depends on the type of user:

- the end user for whom quality in use is mainly a result of functionality, reliability, usability and efficiency,
- the person maintaining the software for whom quality in use is a result of maintainability,
- the person porting the software for whom quality in use is a result of portability.

The difference between usability and the quality of a work system in use is a matter of focus. When usability is evaluated the focus is on improving a product while the other components of the work system are treated as given. It is the aim to improve the quality of the overall work system in use, any part of the work system may be the subject of design or evaluation (Bevan, 1995).

3.2 Usability in design process

There are many aspects to usability, which must be taken into account if a system is to be good. According to Gould et al. (1997) designers focus too much on only one aspect of usability, e.g. knitting pregiven functions together into a user interface. This narrow focus almost guarantees missing the point that usability is made up of many important factors, which are often mutually dependent. Engineers are too busy building and sadly far too many still see the user as a pusher of buttons at best, or an embarrassing inconvenience at worst (ibid.).

To make matters worse, an increasing number of tools such as Visual Basic, Visual C++, Delphi and various Java IDEs, have gained commercial acceptance. The ease with which these tools allow the visual aspects of a Graphical User Interface (GUI) to be constructed undermine the complexities involved in the development of an interface which takes into consideration both the principles of sound software engineering and user needs. In other words, the use of these tools is not governed by usability considerations nor even of the principles of software engineering. This can lead to a situation where a seemingly sophisticated visual interface contains many usability faults, inadequate application functionality and has not been developed with rigorous software engineering practices (Faulkner & Culwin, 2000).

Faulkner and Culwin (2000) criticise that many software engineers seem to have no understanding of what average people can reasonably be expected to do. Systems are developed with deficient regard to the human users and developers act as if anyone is able to do anything, anywhere and at any time. According to them (ibid.) this difference attitude reflects industry practice where the production of software is akin to two people making clothes for a third one. One person knows what the customer looks like, what they want to wear and where they will wear it. The other one knows everything about cutting shapes out and sew up them together, but starts before the size and preferences of the

customer have been communicated or even established. The two tailors then haggle over how to take bits in and let bits out eventually leaving with an awfully fitted garment. Despite this the customer is required to thank and pay the tailors.

Designing for usability should be seen as involving both attending to high-level principles and selection of particular approaches that are determined by a context of use that is broader than we have generally attended to. The acceptability of any software product is not dependent solely on surface interface features, but on the way a system fits within a user context. While to some this might seem obvious, Karat (1997) believes that "the community of system designers—including usability specialists—is still struggling with hard questions of how to turn this obvious fact into specific approaches for dealing with use context in design." As a first step towards focusing on the attention of the designer on the needs of the user and away from a concentration of the hardware, there has been an increasing emphasis on taking a "human-centred" approach to design (Norman & Draper, 1986).

3.3 Does usability really matter?

While there is a wide acceptance of usability principles in system development companies, the majority of development projects are done with little or no usability expertise in house. Also, despite evidence of the benefits of employing human-centred design methods to produce more usable products, the result is that most computer systems are unnecessarily difficult to use. There are a number of possible factors, which appear to underly the lack of influence of usability practices and methods including:

- conservative attitudes to the idea of taking up the ideas of usability and human-centred design,
- problems of integrating usability tools and methods within the design process,

- usability activities taking place too late to have a real influence,
- over-formalised procedures that are soundly based but take too long to apply to be effective (Maguire, 2000).

Furthermore, development schedules are contracting and complexity is increasing at a rapid rate. The number of lines of code is increasing and the toolset for software development is becoming far more restrictive. Another factor that is affecting usability professionals, is the growing audience of users. Whereas the user population for a software product may have previously been measured in the thousands, it is now, especially with Web-site development, for products being measured in the millions. (Maguire, 2000).

The question therefore is does usability really matter? It does, but it seems that a lot needs to be done before human-centred approach could become a general part of the software development just like coding or testing. As Chrusch (2000) points out there still are many myths of usability, which prevents the usability issues from becoming everyday life in system development.

3.4 Summary

The overall quality of the system is seen to be very crucial. However, quality is viewed mainly from the perspective of the system or process. Quality in use concept has extended this traditional view of product quality. Quality in use provides a link to usability and human-centred design approach. Quality in use is the objective, software product quality is the means of achieving it. First of all, the purpose of designing an interactive system is to meet the needs of the user.

4. REQUIREMENTS IN SYSTEM DEVELOPMENT

Requirements are the starting point for the all system development. It is also seen as the most critical phase, which affect to the whole development process. Errors in requirements cost a lot and also the cost of their repair rises as the lifecycle advances. The following sections give an outline of Requirements Engineering (RE) and especially human requirements point of view.

4.1 Requirements Engineering (RE)

The high cost of errors incurred during requirements analysis and many system failures have been attributed to mistakes in the early phases of system development. Requirements engineering (RE) aims to address this problem but the scope of what this entails is not clear (Sutcliffe, 1996). The traditional "waterfall" model of the system development process conceives of requirements gathering as a single step. Users, customers, and/or marketing representatives meet with designers, and describe what they want and/or react to what is proposed (Carroll et al., 1997).

According to Pohl (1996) and Sutcliffe (1996) there is no common definition of RE, and existing definitions differ in their focus. It is widely agreed that the primary product of the RE process is the requirements specification, which should state what a system should do and not how it should do it. The (so-called) formal methods focus mostly on representational aspects and provide hardly any guidance for the development of a specification. Pohl (1996) points out that existing methods ignore the fact that RE is an iterative process in which the RE team learns about the current and/or future reality, e.g. the customer gets an understanding of his/her real needs.

Zave and Jackson (1997) stress that the idea that a requirements specification should include a model of the environment has been familiar for a long time, but according to them, they are the first to propose that requirements should

contain *nothing but* information about the environment. They define the *environment* as the portion of the real world relevant to the software development project. The environment of Zave and Jackson are quite general and more detailed definition for this environment can be found from ISO 9241-11 (1998) and its definition for context of use, where it is seen as "users, tasks, equipment, and the physical and social environments in which a product is used (p. 2)".

Requirements frequently start with a vague statement of intent. The first problem is to establish the boundary of investigation and the scope of the intended system. Unfortunately, this is rarely an easy process as users may have different visions of the system or only partial and incomplete ideas about what they want. Input is characteristically informal in its representation, imprecise and personal as requirements are initially held by individuals and frequently conflict with one another. However, the desired output from RE is very different. It should be a complete system specification, within the constraints of available resources, using a formal language, and agreed by all involved (Sutcliffe, 1996).

For the most part the techniques for fact gathering activity have been borrowed from systems analysis, e.g. interviews, observation, questionnaires. Analysis and modelling generally follow top-down approaches, concentrating on goal decomposition. Modelling activity consumes the output from analysis, structures facts and represents them in a notation. Validation is a key activity in RE. Validation implies getting users to understand the implications of a requirements specification and then agree that it accurately reflects their wishes (Sutcliffe, 1996).

Requirements gathering or analysis is the process of finding out what a customer requires from a software system. Requirements gathering is concerned with understanding needs. A requirement is something that the product must do or a quality that the product must have (Robertson &

Robertson, 1999). Requirements of a system can be considered in different categories. Preece et al. (1994) use division to the functional, data and usability requirements. Functional requirements specify what the system must do. Whereas data requirements specify the structure of the system and the data that must be available for processing to be successful. Usability requirements specify the acceptable level of user performance and satisfaction with the system.

Robertson, et al. (1999) uses different division for the requirements. They divide requirements only to two categories, which are functional and user requirements. Robertson et al. (1999) gives similar definition for the functional requirements as Preece at al. (1994), but they see the objective of the user requirements process as identifying who the users of the system will be, to understand the charachteristics of these users, and to determine the usability requirements for the system.

Requirements engineering still lacks consensus as to what exactly constitutes software requirements engineering as a whole but presents a model in terms of three stages: elicitation, representation, and validation. These terms are likely to be more familiar to the software engineer but it will be seen that they map onto the terms used in the definition of user-based requirements engineering. At present there is no commonly accepted interdisciplinary framework which takes account of the wide range of criteria and practices relevant to studies of both software engineering and human-computer interaction (Preece & Rombach, 1994).

4.2 User Requirements

The role of the user into the lifecycle of a requirements document is classically viewed as (1) to provide input and (2) to sign off the end result. Comment from end users may not achieve its purpose for a number of reasons; chief among them being that the non-technical end user may not understand the notations

and language used in a software engineering requirements specification, and may be unable to visualise or predict the end result of a development process based on such a specification.

Figure 7 presents three paradigms according to Grudin (1995) for software project development based on when users and developers are identified. The left edge of each horizontal bar represents the point when many of the project's developers or eventual users are known. This framework best applies to projects that begin with a decision to build, modify, or replace a system or application and include a planned completion or delivery date. The three development paradigms describe a large set of projects that have significantly influenced interactive systems development.

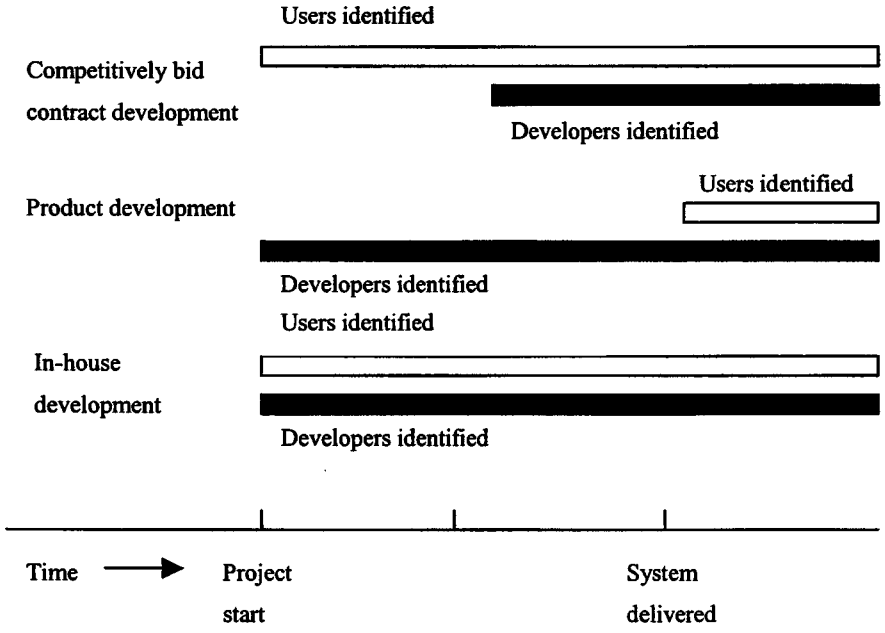

FIGURE 7. Project time lines with points of user and developer identification. (According to Grudin, 1995, p. 295).

In contract development or software acquisition, the user organization is known

from the outset, but the development organization is identified after a contract is awarded. The clearest cases involve competitive bidding. The essence of contract development is that the users are identified before the developers. In product development, also called off-the-shelf software development, the developers are known from the outset, but the users often remain unknown until the product is marketed. Finally, in in-house development both the users and the developers are known at the project outset (Grudin, 1995).

When collecting the user requirements these different paradigms can affect the quality and validation of the user requirements. For example in product development, it is not possible to collect real user data. There can be done surveys among stakeholders, but even at the best the user requirements are only wild guesses. Even in the contract development the problem can be that developers are subcontractors, who work on the basis of ready-made requirements specifications. In this case it is possible that developers do not even meet the end users at the any phase of the development.

Projects may not be pure expressions of one paradigm. It is not surprising that conditions for user participation vary across these development contexts. The timing gap between user and developer involvement is an obstacle to collaboration. Each of the paradigms or contexts in Figure 7 has contributed to contemporary practices of developing interactive systems. Contract development has been central to the evolution of software project management methods, product development has focused attention on computer interfaces to individual users, and in-house development has explored social and organizational aspects of system use (Grudin, 1995).

Understanding user requirements is a first and integral part of information systems design. According to a survey by Standish Group of 365 IT executive managers for their opinions on why projects succeed, three major reasons behind the success of a project which emerged were: 1) user involvement, 2) executive management support, and 3) a clear statement of requirements. On

the other hand, opinions, concerning why projects are impaired and ultimately canceled, ranked incomplete requirements and lack of user involvement at the top of the list. (The Standish Group International, Inc., 1995).

When developing computer based information system, it is necessary to understand how the end-users perform their work and interact with the environment. It is also necessary that the users' requirements are described in a way that all participants can comprehend. Modelling the users' work normally consists of the two major activities of analysis and documentation.

Systems software developers face the particular challenge of satisfying the diverse and sometimes contradictory needs of users, ranging from experts (professional programmers) to application developers to novices ("black box" users) with only basic programming skills. Overall, enhanced usability will determine whether users accept these systems. More specifically, user requirements include:

- easy access and use,
- easy application engineering,
- portability and scalability of applications and development tools,
- performance and fault tolerance of applications, and
- customization and adaptability (Decker, 1997).

The following human-centred methods are current in software engineering treatments of human-based requirements work:

- interviews,
- surveys,
- naturalistic observation,
- brain storming,
- focus groups, and
- ethnographic approaches.

These methods are all relevant to the early user context analysis phase of

human-based requirements work. However, they do not support in any strong way the feasibility and prototyping phase, nor do they support the important user requirements synthesis phase.

A view of how different aspects of requirements (business and market; user; and functional) can be successfully integrated as a spiral process in a realistic commercial environment is given in (Dayton, et al. 1998), which describes how users, usability engineers, system engineers, and developers are able to rapidly co-ordinate their efforts to produce a GUI design using an iterative methodology known as the Bridge.

4.3 Summary

Requirements, all kinds, are important, when developing a computer-based system. The so-called hard requirements tend to be more interesting and stressed among the developers. However, recent development has risen the soft requirements to the same level with hard and more technical sound requirements. The issue is that all needs, hard or soft, should have as important. Many times the problem has been that the soft side of requirements has been forgotten and this means that the user has been forgotten.

5. HUMAN-CENTRED DESIGN APPROACH AND USER REQUIREMENTS FRAMEWORK

There are many approaches to design systems, which have generally been described under the heading of "human-centred design." Success in designing affordances into the interface is based on understanding the user, the user's tasks, and the context in which the user accomplishes tasks and goals. When these aspects of the context of use are taken into account, it becomes possible to design a system the user understands, appreciates, and uses.

This chapter provides a description of the human-centred design approach based on ISO 13407 (1999). The main purpose of the chapter is to give an idea of the planning and management of human-centred design. However, it does not provide detailed covereage of the methods and techniques required for human-centred design. The principles of human-centred design approach will provide a framework for user requirements and then I will form assessment criteria for applying framework to the process of gathering user requirements.

5.1 Human-centred design approach

Human-centred design can be thought of as a cognitive task that requires the designer to encode and transform information in the course of application definition. Human-centred design entails the creation of a particular "representational system" through a process that begins with user data and ends with a hardware and software system for some end user task. The gap between user data and concrete design is a gap between "represented world " and a "representing world" (Graefe, 1998). Bridging the gap requires the definition of modellings of represented and representing worlds and their correspondences.

An IBM usability research group stressed "an early focus on users" in the 1970s, and an influential paper by Gould and Lewis (1983) recommended that

"typical users become part of the design team from the very outset when their perspectives can have the most influence, rather than using them to "review", "sign off", or "agree" to the design before it is coded." A user focus was further emphasized in User Centered System Design, an influential 1986 collection of research papers. Furthermore, in the 1970s and 1980s Europeans experimented with greater user involvement in systems design. In particular, the Scandinavian "participatory design" approach, in which users collaborate as full development team members, attracted attention (Grudin, 1995).

Landauer (1993) suggests that computers have failed to deliver the gains in productivity that they once promised and we once expected of them. He blames this failure on the problem that systems are still far from easy to use. He thinks that the best way forward is to ensure that systems are usable by adopting what he labels a UCD approach. Landauer (1999) defines UCD as user-centred design, user centred development, and user centred deployment. According to him they do not need separate acronyms because at the base they are all the same.

According to Karat et al. (2000) one might assume there is progress because we are developing more natural technologies—speech recognition and output, and advanced visual interfaces. But working with these and other technologies one thing has become very clear. It is not simply the technology employed that makes a system fit its user. Speech recognition systems have been rejected because they are not natural for interaction and visually appealing systems that fail because users do not think they provide the function they are seeking (ibid.).

The main cost to achieve the benefits of human-centred design is that the manager's task initially appears more difficult. Project planning has to allow for iteration and for incorporating user feedback. More time will also be required for effective communication between design team participants and for reconciling potential conflicts and trade-offs. However, project managers will benefit from the additional creativity and ideas from an extended development team and skill

base. In addition, informal communication and discussion of usage issues early on in the project will result in a better design and significant savings at later stages when changes are more costly (Maguire, 1997).

5.2 The human-centred process plan

Different organisations are at different levels of usability maturity - from not recognising usability as an issue, to having processes in place, which ensure the development of consistently usable systems. Organisations can integrate the principles of ISO 13407 into their development process incrementally, to achieve an appropriate maturity level. The first step is to plan which methods are expected to be used at different stages of development. This will depend on the business case for usability, and will take account of the budget, timescales, resources, skills and other constraints. Planning may also include assessing the usability capability maturity of the organisation and identifying where improvements are required. The stages are according to Bevan & Curson (1998):

- ignorance: We don't have problems with usability,
- uncertainty: We don't know why we have problems with usability,
- awakening: Must we always have problems with usability,
- enlightenment: Through management commitment and improvement of human-centred processes we are identifying and resolving our problems, and
- wisdom: Usability defect prevention is a routine part of our operation.

If the developing organisation has a quality system and associated quality plans for software development, then a specific quality plan should be included for the human-centred design process covering both the type of software development process adopted and the quality control measures. This may form part of a broader software quality plan which sets criteria for usability both as an attribute of the software and as a quality of the overall system (Kirakowski, 1996).

If the company does not have a quality system in place, then a plan should be developed to specify how the human-centred activities fit into the overall system development process.

The plan should identify:

- the human-centred design process activities i.e. understanding and identifying context of use, specifying user and organisational requirements, producing prototypes and evaluating designs according to user criteria,
- procedures for integrating these activities with other design activities,
- the individuals and the organisation(s) responsible for the human-centred design activities and the range of skills and viewpoints they provide,
- effective procedures for establishing feedback and communication on human-centred design activities as they affect other design activities,
- appropriate milestones during the design and development process, e.g. through specification of lifecycle documents (such as Usability Requirement Specification as in ISO 9241 Part 11 Annex C),
- characteristics (or attributes) which will be subject to quality control and measurements during the whole process,
- suitable timescales to allow feedback to be incorporated into the design schedule,
- the human-centred evaluation methods to be used at each stage in the lifecycle (Kirakowski, 1996).

In order to decide wheather to adopt a human-centred approach, it is necessary to assess the benefits and the costs. Bevan (1995) points out that human-centred design can potentially provide benefits in four areas:

- reduced production costs: the overall development times and costs can be reduced by avoiding over design and reducing the number of changes

required late in design,

- reduced support costs: systems which are easier to use require less training, less user support and less subsequent maintenance,
- reduced costs in use: systems better matched to user needs improve productivity and the quality of actions and decisions. Easier to use systems reduce stress and enable workers to handle a wider variety of tasks,
- improved product quality: human-centred design results in products which have a higher quality of use and are more competitive a market which is demanding easier to use systems (Bevan, 1996).

This human-centred design process plan should form part of the overall project quality plan and should also be subject to the same project disciplines (e.g. responsibilities, change control etc.) as other key activities to ensure it is followed through and implemented effectively (Kirakowski, 1996).

5.3 Overview of human-centred design

Gould and Lewis (1985) recommended three principles of human-centred design. These were early focus on users and tasks, empirical measurement and iterative design. General principles of human-centred design in ISO 13407 (1999) are along the same lines with Gould and Lewis. According to ISO 13407 a human-centred approach is characterized by:

1. the active involvement of users and a clear understanding of user and task requirements,
2. an appropriate allocation of function between users and technology,
3. the iteration of design solutions, and
4. multi-disciplinary design (ISO 13407, 1999).

ISO 13407 (1999) advocates a process-oriented approach to usability, whereby usable systems are achieved as the result of a human-centred design process. The standard provides a framework for applying human-centred design and evaluation techniques, and is intended to supplement existing lifecycle models.

ISO 13407 specifies types of activity to be performed during the development of an interactive system, but does not demand nor recommend particular techniques or methods.

There are many industry and proprietary standard methods for the design of computer based interactive systems. However, ISO 13407 (1999) does not assume any one standard design process. It is complementary to existing design methods and provides a human-centred perspective that can be integrated into different forms of design process in a way that is appropriate to the particular context.

5.4 Human-centred design activities

There are four human-centred design activities that need to take place at all stages during a project. These are to:

- understand and specify the context of use,
- specify the user and organisational requirements,
- produce design solutions, and
- evaluate designs against requirements (ISO 13407, 1999).

The iterative nature of these activities is illustrated in Figure 8. The process involves iterating until the objectives are satisfied.

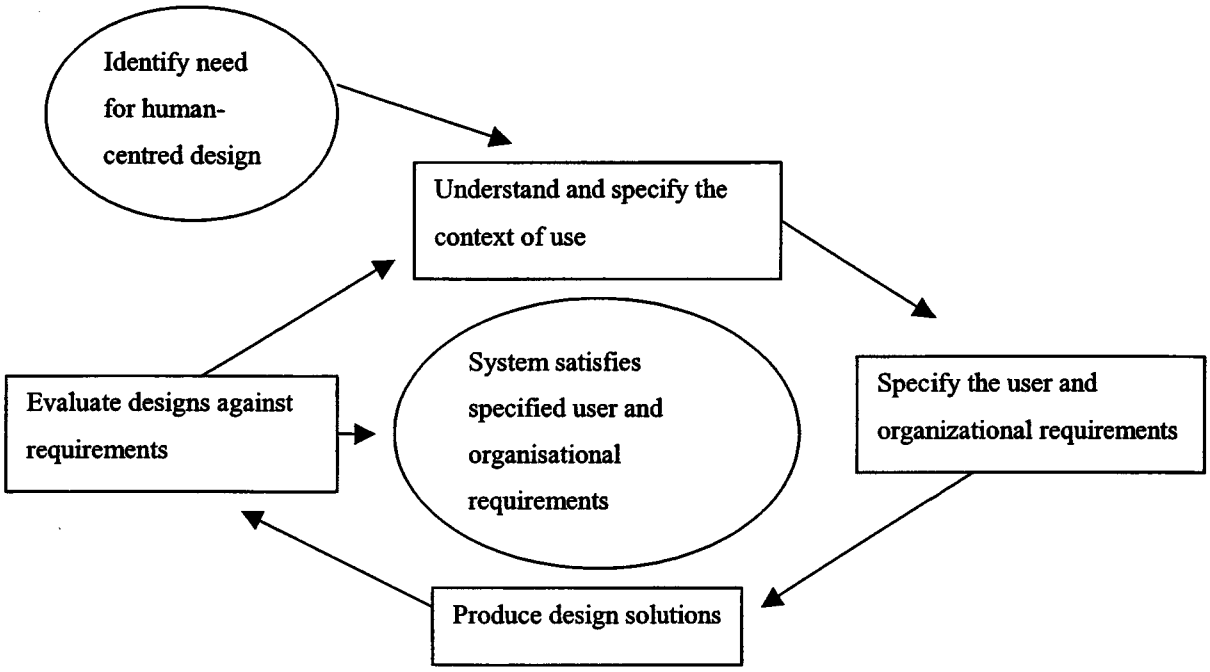

FIGURE 8. The interdependence of human-centred design activites (ISO 13407, 1999, p. 6).

At a high level the activities of identifying the context of use, setting requirements, designing a solution, and evaluating could be seen as representing the overall design process that continues until the system satisfies specified user and organizational requirements.

At a more detailed level these activities happen iteratively during human-centred design, so that the criterion is really until the system sufficiently satisfies specified user and organizational requirements to meet the needs at this stage of the development process. So in practice the activities of identifying the context of use, setting requirements, designing a solution and evaluating should be repeated several times. Post-delivery evaluations may well identify changes in user requirements that lead to a new cycle of design and evaluation. According to ISO 13407 (1999) it may be necessary to specify criteria both for the minimum acceptable levels and for the target levels to be achieved.

Understand and specify the context of use

The characteristics of the users, tasks, and the organisational and physical environment define the context in which the product is used. It is important to understand and identify the details of this context in order to guide early design decisions, and to provide a basis for evaluation. If an existing product is to be upgraded or enhanced, the context will already be well understood. If there are extensive results from user feedback, help desk reports and other data, these provide a basis for prioritising user requirements for system modifications and changes. For new products or systems, it is necessary to gather information about its context of use. The context in which the product is to be used should be identified in terms of (ISO 13407, 1999):

1. **the characteristics of the intended users**. Relevant characteristics of the users can include knowledge, skill, experience, education, training, physical attributes, habits, and capabilities. If necessary, define the characteristics of different types of user, for example with different levels of experience or performing different roles (maintainers, installers etc.),
2. **the tasks the users are to perform**. The description should include the overall goals of use of the system. The characteristics of tasks that can influence quality in use in typical scenarios should be described, e.g. the frequency and the duration of performance. The description should include the allocation of activities and operational steps between the human and technological resources. Tasks should not be described solely in terms of the functions or features provided by a product or system,
3. **the environment in which the users are to use the product**. The description of the hardware, software and materials can be in terms of a set of products, one or more of which can be the focus of human-centred specification or evaluation, or it can be in terms of a set of attributes or performance characteristics of the hardware, software and other materials. Relevant characteristics of the physical and social environment also need to

be described. Aspects, which may need to be described, include attributes of the wider technical environment (e.g. a local area network), the physical environment (e.g. workplace, furniture), the ambient environment (e.g. temperature, humidity), the legislative environment (e.g. laws, ordinances, directives and standards) and the social and cultural environment (e.g. work practices, organisational structure and attitudes).

The output from this activity should be a description of the relevant characteristics of the users, tasks and environment which identifies what aspects have an important impact on the system design.

This description is unlikely to be a single output that is issued once. It is more often a "working document" that is first produced in outline terms and is then reviewed, maintained, extended, and updated during the design and development process.

There are different methods, which can be used for collecting information about the context of use. In the first instance it will usually be necessary to gather together a group of experts in the different aspects of the context to discuss and agree the details of the intended context of use.

Specify the user and organisational requirements

Requirements elicitation and analysis is widely accepted to be the most crucial part of software development. Indeed, the success of the human-centred approach largely depends on how well this activity is done.

In most design processes, there is a major activity specifying the functional and other requirements for the product or system. For human-centred design, it is essential to extend this activity to include an explicit statement of user and organisational requirements, in relation to the context of use description. According to ISO 13407 (1999) the following aspects should be considered in order to identify relevant requirements:

- required performance of the new system against operational and financial objectives,
- relevant statutory or legislative requirements, including safety and health,
- the users' jobs,
- task performance,
- work design and organization,
- management of change, including training and personnel to be involved,
- feasibility of operation and maintenance,
- the human-computer interface and workstation design.

Objectives can be set with appropriate trade-offs identified between the different requirements. The requirements should be stated in terms that permit subsequent testing.

Produce design solutions

The next stage is to create potential design solutions by drawing on the established state of the art and the experience and knowledge of the participants. The process therefore involves:

- using existing knowledge to develop proposed multi-disciplinary design solutions,
- making those design solutions more concrete (using simulations, models, mock-ups etc.),
- showing the solutions to users and allowing them to perform tasks (or simulated tasks),
- using this feedback to improve the designs and iterating this process until the human-centred design goals are met,
- managing the iteration of design solutions.

Evaluate designs against requirements

Evaluation is an essential step in human-centred design and should take place at all stages in the system lifecycle. Evaluation can be used to:

- provide feedback which can be used to improve design,
- assess whether user and organisational objectives have been achieved,
- monitor long term use of the product or system (ISO 13407, 1999).

Early in design the emphasis is on obtaining feedback that can be used to guide design, while later when a realistic prototype is available it is possible to measure whether user and organisational objectives have been achieved.

In the early stages of the development and design process, changes are relatively inexpensive. The longer the process has progressed and the more fully the system is defined, the more expensive the introduction of changes is. It is therefore important to start evaluation as early as possible.

5.5 The framework of user requirements in human-centred design

A general approach to specify user requirements needs to be flexible to meet different situations e.g. generic or custom system development, new systems or developments of existing systems. Traditional software engineering tends to take an idealistic view to requirements. It essentially looks at requirements in an abstract way rather than contextualised as part of tasks to perform.

Maguire (1998) has developed a framework based on ISO 13407 (1999) for requirements engineering called RESPECT. It emphasizes the importance of obtaining a complete understanding of user needs, and validating the emerging requirements against potential real world scenarios of usage. According to Maguire (1998) the three main phases of the user requirements specification

framework are:

1. user context and early desing,
2. prototype and user test, and
3. user requirements documentation.

Each phase is broken down into a series of numbered sections. The aim of the method is to collect data (e.g. user characteristics) and to generate items for each section. The process is flexible and the order in which each section is considered can vary as appropriate to the situation.

The development of user requirements will be an iterative process. Items of information are noted, then modified or firmed up. The method of data collection is also flexible. While in some situations requirements can be generated through discussion, in others, the use of particular techniques and methods may be needed (Maguire, 1998). Inside every stage of the framework all human-centred activities (context, requirements, design and test) described in section 5.4 in this thesis is performed.

Phase 1 (Figure 9) consists of the first iteration around the human-centred design loop. The goal is to understand the initial project requirement and performing an analysis of the main user groups, their tasks and working environments. Contextual factors may highlight a need for particular user requirements. User goals and tasks are identified and current process for each goal is reviewed. A list of design ideas or a design concept may be produced, represented and reviewed in order to decide whether they form a good basis for meeting the user goals.

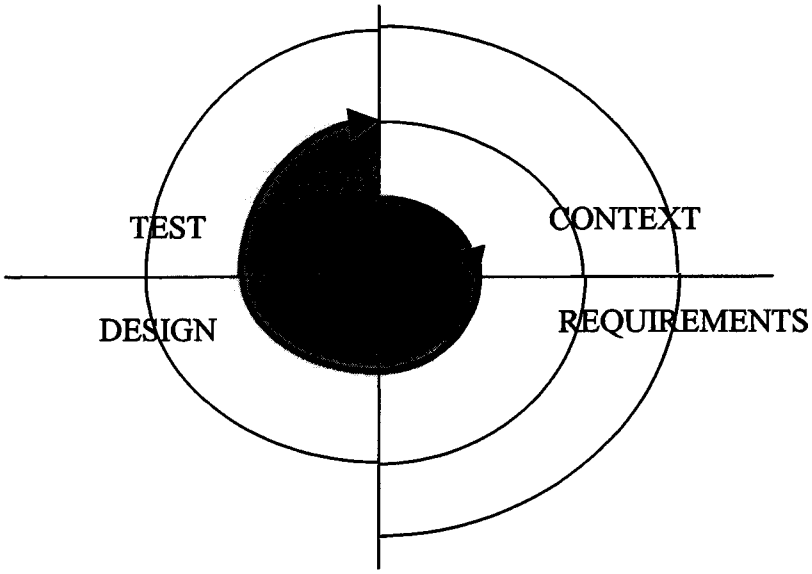

FIGURE 9. Phase 1 User context and Early design (Maguire, 1998, p. 17)

Phase 2 (Figure 10) is the second iteration around the human-centred design loop. The first part of phase 2 is concerned to identify task scenarios that will be used as a way of testing the system design. Each scenario represents a user goal set within a particular context. For each scenario, a set of steps representing an interactive process is developed. An interactive prototype will be produced and used to test the system concept against the scenarios with potential users. A review is carried out of the tasks that each user will carry out for all user goals.

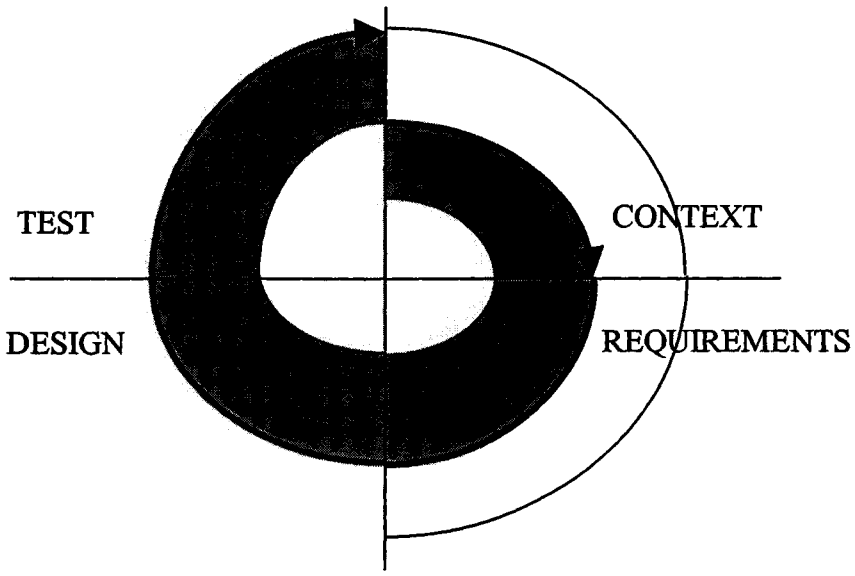

FIGURE 10. Phase 2 Prototyping and User testing (Maguire, 1998, p. 49)

Stage 3 (Figure 11) is the third iteration. This phase include stating requirements for the system in general including the general characteristics of the system and the organisational structure on which it is based. An outline plan for implementing the user requirements within the design process is also produced. It will also include a list of internal guides or external standards that should be reffered to during implementation. The previous two stages will have produced a series of potential user requirements that should be considered as a basis for agreed user requirements, which will form the basis of the user requirements specification. The first draft of user requirements specification document will be written during this phase.

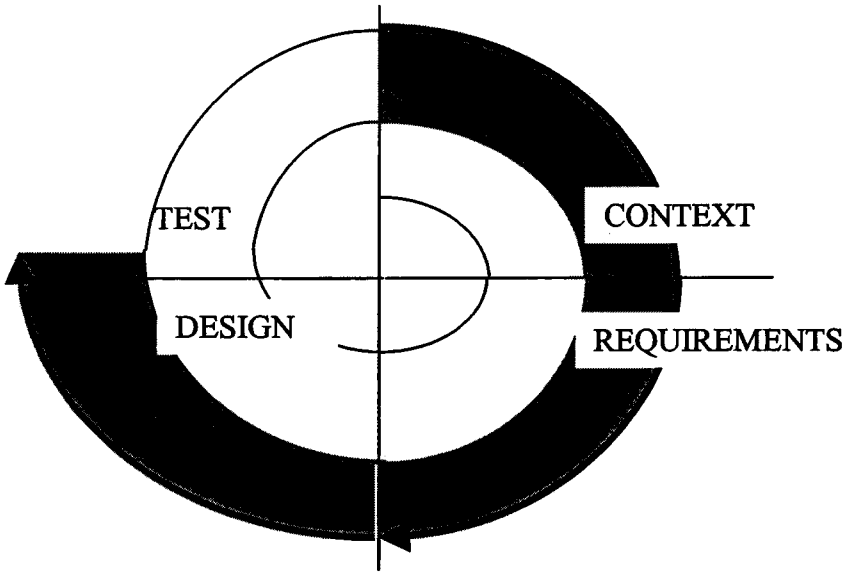

FIGURE 11. Phase 3 User Requirements Documentation (Maguire, 1998, p. 67)

One of the key aims of the framework is to encourage the take-up of more human-oriented approaches to requirements gathering and specification. However, it is also important to relate the procedures to existing requirements engineering processes. Furthermore, collecting user requirements using this framework will demonstrate that a user-driven approach to capturing user requirements has been adopted, and will supports the requirements of the ISO standards 9241-11 and 13407 (Maguire, 1997).

Requirements phase is already defined in many existing methods and many companies have their own applied methods, which are quite difficult to change. To find out in which level the used method supports the ideas of human-centred user requirements approach, the following table gives some assessment criteria for it. The criteria are based on stages of the framework and principles of human-centred design.

STAGE	Assessment criteria
1. USER CONTEXT AND EARLY DESIGN	How does the method define the different users or stakeholders and their characteristics for the new system?
	Are the current tasks or the future system tasks described?
	Have the characteristics of the environment been considered?
	Is there a phase for producing the new design ideas?
	How are the design ideas and concepts been tested?
2. PROTOTYPE AND USER TEST	Are the general usability goals been set to use as a basis for establishing user requirements?
	Has the balance between user actions and system actions been considered properly?
	Is there a prototype and has it been tested with users?
	Have the costs and benefits to the users of the new system been considered to ensure that the benefits at least match the costs?
3. USER REQUIREMENTS DOCUMENTATION	Have the user requirements been assembled into suitable form for input to system development?

Table 1. Criteria for applying framework to user requirements process

5.6 Summary

Human-centred design is an approach to interactive system development, which focuses on making systems usable and safe for their users. The benefits include increased productivity, enhanced quality of work, reductions in support and training costs. Although there is a substantial body of human factors and interaction knowledge about how such design processes can be organised, much of this information is not yet widely applied.

Typically the development process starts with requirements gathering. All kinds of requirements - business, user, functional and technical - must be developed to ensure the success of the system. Historically the importance of user requirements is often overlooked. However, understanding user requirements is an integral part of information systems design and is critical to the success of project.

There are many design methods for computer systems. These design methods typically address more fully technical and engineering perspectives than human issues. It is not an easy task to apply new methods and techniques to the existing development processes. However, human-centred design approach allows organizations to incorporate human-centred design into their existing internal procedures and development standards and this inclusion can be done gradually by starting e.g. the first and most important phase, requirements analysis.

The following chapter gives an outline of one of the design methods called OMT++. This method stresses the importance of the end user. However, it is not a so-called human-centred design method. The chapter will emphasise the requirements gathering of OMT++ method on the basis of the assessment criteria developed in foregoing chapter.

6. GENERAL OVERVIEW OF OMT++ METHOD

OMT++ is based on a method called Object Modelling Technique, OMT (Rumbaugh et al., 1991 in Jaaksi, 1997). The aims of OMT++ is the construction of object-oriented software systems. An object-oriented method must support three aspects of the system to be developed (Jaaksi, 1997). First of all, the method must model the requirements for the system and its functionality, i.e. what the system provides to the end user. Secondly, the method must model the objects that constitute the system; what the objects are and how they relate to each other. Thirdly, the method must model how objects collaborate in order to provide desired functionality.

Object-oriented Software Engineering, i.e. the OOSE method (Jacobson et al., 1992 in Jaaksi, 1997) is one of the key sources of OMT++. The way of modelling the system and its communication with external agents with event traces is borrowed from the Fusion method (Coleman, et al., 1994 in Jaaksi, 1997). In Fusion these specifications are called scenarios. In addition to the ideas from commercial methods, there are elements whose development can be traced to Nokia Telecommunications.

According to Jaaksi et al. (1999) software systems should be able to deal with events coming from various sources, and the main concern of software developers is developing applications that provide smooth communication with the outside world. On the other hand, although events come from various sources and a system must respond to them, the final responsibility lies with end users. Therefore the most important task of almost any system is to be of service to the final users.

This case study concentrate on early requirement collection and analysis under feasibility study processes of OMT++. The following is the general description of the OMT++ phases and more detailed description of the requirements capture

according to Jaaksi (1997) and Jaaksi et al. (1999). The main focus is on finding if OMT++ answers to any or all of the human-centred user requirements criteria, which are defined in the previous chapter.

6.1 Overview of the OMT++ process

The notation of OMT++ includes three main elements: natural language, object models, and event traces. Natural language is the main tool to capture requirements, and it is typically used whenever there is a need to communicate with end users.

The OMT++ method consists of four main phases, namely object-oriented analysis, object-oriented design, object-oriented programming, and testing. Each phase provides views on different levels of abstraction to the system to be developed. Also, each phase builds on top of the previous phases, and each phase sets requirements for the following phases.

Object-oriented analysis is the first phase of software development according to OMT++ (Jaaksi, 1997; Jaaksi et al., 1999). The objective of analysis is to collect requirements and analyse the problem at hand. In addition to this, the analysis phase of OMT++ constructs a high-level solution to the problem from the user's point of view. The analysis uses only concepts that are meaningful for the end user. The analysis phase has two subphases, namely requirements capture and requirements specification. The requirements capture subphase collects and documents all requirements and use cases for the system to be developed. The requirements specification subphase analyses and models both the concepts and the external functionality of the system. Thus, the analysis phase collects, analyses, and refines the requirements. The analysis phase produces three documents, namely, the feasibility study, the analysis document, and the user-interface specification document (Jaaksi, 1997; Jaaksi, et al., 1999).

The next phase is the design phase, which consists of architectural design and

detailed design. The design phase uses the artifacts of the analysis phase and produces technical specifications, which can be implemented by programming. During the architectural design phase, the collaborative objects are typically entire executables, components, libraries, and devices, while in the detailed design phase the objects of interest are C++ or Java classes, for example. (Jaaksi, 1997; Jaaksi, et al. 1999).

The outcome of the programming phase depends on the programming language used. The programming phase includes coding as well as compiling, linking, and debugging (Jaaksi et al., 1999). Finally, the testing phase tests the system to find errors in design and programming and to verify that the system meets the requirements. The testing phase also includes the final integration of the components (Jaaksi, 1997).

6.2 Static and Functional Paths

The analysis, design, programming, and testing are separate phases. They can be arranged either in a waterfall or iterative manner. OMT++ identify two parallel paths in the process: the static and functional paths (see figure 12). All phases are based on these two perspectives. During the analysis phase, the static entities are the system itself; concepts the system deals with, external devices, and end users.

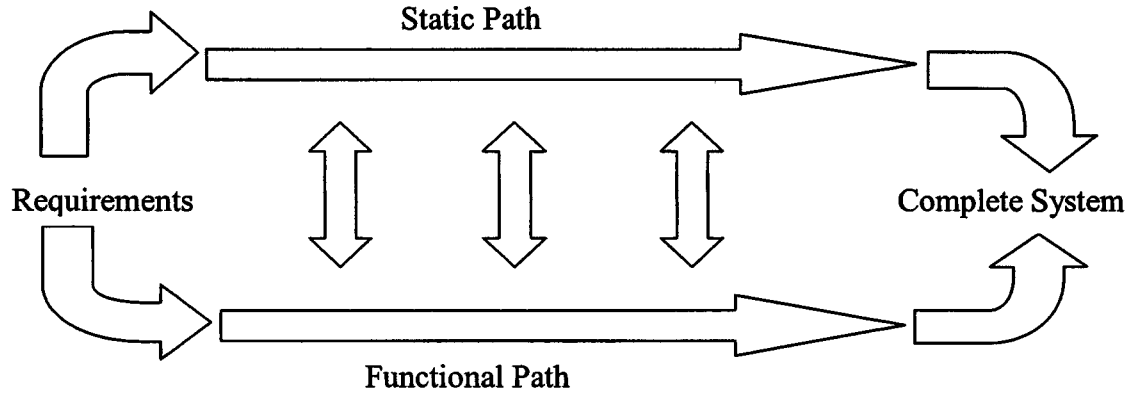

FIGURE 12. The two paths of OMT++ (Jaaksi et al., 1999, 6).

Figure 13 illustrates the main phases of OMT++. Although there are two separate paths, these paths are still closely related. From the notation point of view, the static path uses class, component, and deployment diagrams of UML. On the other hand, the functional path uses sequence diagrams to illustrate the functional and dynamic behavior of the system (Jaaksi et al., 1999).

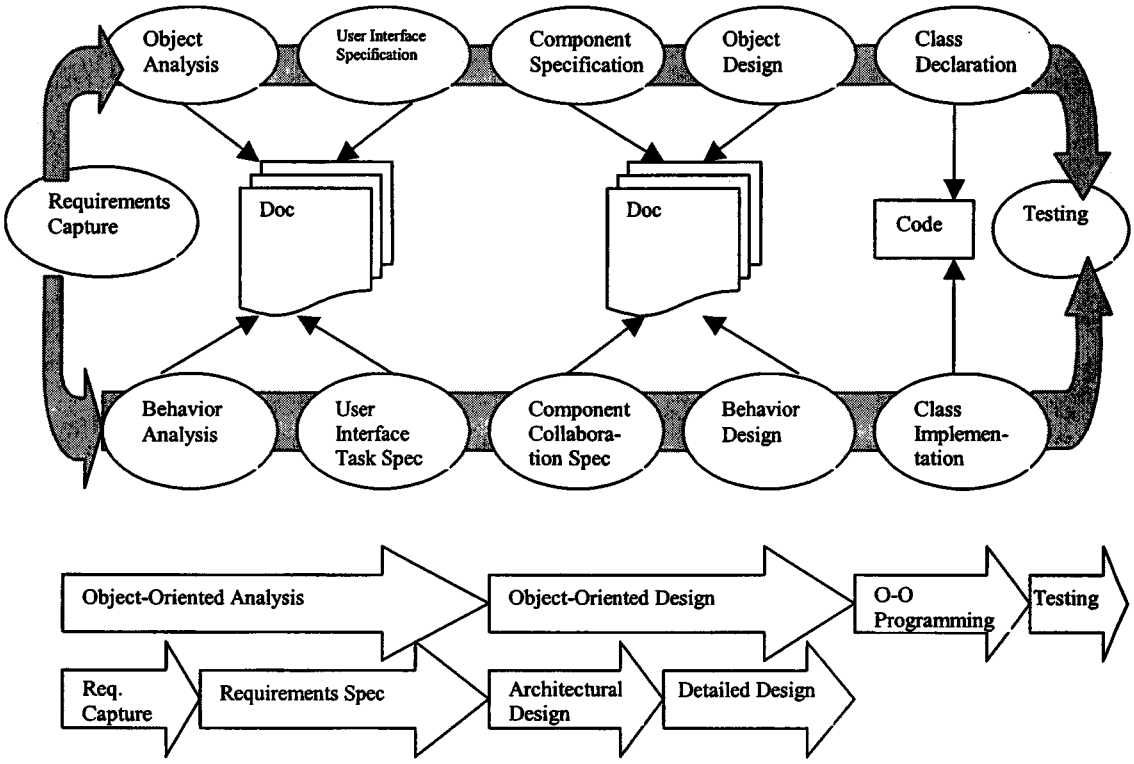

FIGURE 13. The main phases and the two paths of OMT++ (Jaaksi et al., 1999, 7).

These two paths provide different views to the developed system. In each phase, it is first modelled the static entities of the system and then the functional path illustrates how instances of these static entities collaborate in the context of system functionality. The static path uses packages and class, component, and deployment diagrams to illustrate the static properties of the system, and the functional path uses use cases and sequence diagrams to illustrate the functional and dynamic behavior of the system. These two paths complement each other, and they both aim at a complete, tested system. (Jaaksi et al., 1999).

However, during requirements capture and testing the existence of two paths is not emphasised. All other phases are based on these two perspective. According to object paradigm, the output of both paths is implemented into tight independent packages that include both data and behaviour (Jaaksi, 1997).

6.4 Feasibility study

The feasibility of planned software development project or software product is studied concurrently with requirements capture (Jaaksi, 1997). Early requirement collection and analysis is performed under a so-called feasibility study process (Jaaksi et al., 1999). It is an independent phase that takes place before the actual software development. One could also say that the feasibility study phase includes also the first steps of requirements specification since it models some aspects of the functionality of the system. In fact, feasibility study is another name for an early requirements collection, as suggested by Gause and Weinberg (1989, p. 58).

Feasibility study serves two purposes: It collects requirements and screens solution suggestions for a development project and it estimates development efforts based on the solution screenings (Jaaksi et al., 1999). The document, which describes the requirements and use cases is created during the feasibility stage. Thus, this document could also be called the "early requirements document". However, the document also includes preliminary solution suggestion and work estimations and that is why it is called the feasibility study document. I will now concentrate more on the feasibility study's requirements capture and use cases.

6.5 Requirements capture and use cases

Each software project begins with requirements capture. The objective of the requirements capture phase is to collect and document all requirements for a new system or a system enhancement. Before a software development project starts there typically are a great deal of information available. In most cases, ideas, concepts, and requirements are vague and they need a lot of work to be understood. The requirements capture phase collects raw requirements from various sources and documents them explicitly as requirement statements.

In OMT++ requirements are documented in a numbered list and divided into two main groups namely functional requirements and nonfunctional requirements. Functional requirements tell what the system does. They explain how the system functions when observed from outside. In addition, the functional requirements specify how the end user works with the system. Nonfunctional requirements are mainly technical. They tell how closely the functional requirements must be met regarding attributes such as responsiveness, capacity, and usability.

The documented requirements are exemplified and demonstrated in more details as use cases. On one hand, end users and software developers must be able to discuss the requirements and form a common understanding of what kind of system will be developed. On the other hand, software developers need a deeper and more detailed understanding of the functionality of the system. (Jaaksi et al., 1999). Dano et al. (1997) have defined the use cases as "an objective that an actor should achieve by using one complete functionality of the system" (p. 80). However, like any methodological element, use cases need adaptation when taken into use, since organisations, software products, and software projects differ from each other (Jaaksi, 1998). Jaaksi et al. (1999) have defined their own way of using use cases. For them, they are just a means of communication between software developers and users and the object orientation of the use cases is not an issue at all.

In OMT++ use cases are simply textual presentations of the usage of the future systems. Use cases are "short stories" that explain how the end users will use the system and how the system will response to the users actions. Whereas a functional requirement specifies what the system should do, a use case describes how the system should do what it does together with the end user or any other external entity. The internal implementation of the system is never described. The user must be able to use the system according to the documented use cases. Thus, it is important that everybody, including customers, managers, designers, and testers can read and understand the use cases (Jaaksi et al., 1999).

OMT++ presents the usability requirements as a part of use cases. In most cases, the usability requirements are related to the operations performed by the end user. Jaaksi et al. (1999) emphasize that designers should always plan the system for good usability from the very beginning of system development. However, OMT++ does not specify the elements of the user interface with use cases. Too early introduction of buttons, text fields, and other elements would easily obscure the focus from the users' workflows, which is the first thing to understand before user interface design (Jaaksi, 1997; Jaaksi et al., 1999).

Original idea of use cases as "a sequence of transactions between the user and a system", did not give enough guidance for producing use cases with good quality. With this guidance use cases were often too vague or in the other extreme too detailed. These use cases were too limited for communication purposes and too detailed for the early phases of the system development (Jaaksi et al., 1999). To tackle these problems Jaaksi (1998) developed guidelines for finding and writing use cases. The following are the description of 'ten commands' for use cases proposed by Jaaksi (1998):

1. Use cases must specify the most important functional requirements. Use cases are a tool to analyse functional and other requirements. Thus, when a

requirement is defined it should be used one or more use cases.

2. A use case describes something that the designer can be proud of and the customer is willing to pay for. Too broad use cases are either too complex to understand or too vague to be useful, while too narrow use cases are either too detailed or meaningless as such. Each use case must describe something that is beneficial to the end user as such.

3. A use case depicts a typical way of using the system - but nothing more. A use case should not try to cover issues outside of its area, and it should not try to define all possible ways of performing a task.

4. A use case is a play. Anybody who wants to take the role of an actor must be capable of performing as intended in the play just by reading the manuscript, that is, the use case.

5. A use case has a beginning, a main body, and an ending. Each use case should be a complete story.

6. A use case is like an essay written by an elementary school pupil. At a certain age kids tend to write down stories that describe an explicit flow of actions that take place one after another, which is exactly what use cases should do. Speculations about the functional alternatives or implementation possibilities of the future systems are not the part of the use case. Refinements should modify and improve the use cases later in the project.

7. A use case fits in one page. Longer use cases are hard to understand.

8. A use case is loud and clear. Use cases should be so clear and explicit that people who read them, such as customers and software designers, can form strong opinions. If nobody disagrees about the first version of the use case, it is probably too vague and should be made more exlicit.

9. Customers and software designers can sign the use case. Use cases act as contracts between the customer and the developers. Nobody should make any modifications to the use cases without everybody's approval.

10. A use case can be used in system development and system testing. Use cases should be specified so that they can be used later in the project.

OMT++ uses use cases to collect and analyse functional requirements during the requirements capture phase. After this phase OMT++ follows two paths, namely a static class-oriented path and functional behaviour path. OMT++ uses use cases as a source of information both in the static and functional paths. (Jaaksi, 1998b).

6.6 Summary

OMT++ process model consists of four main phases, namely object-oriented analysis, object-oriented design, object-oriented programming, and testing. There are two parallel paths through the process: the upper static path and the lower functional path.

According to Jaaksi et al. (1999) the bigger the development team and the more complicated the system, the more needs to place emphasis on how to gather and understand requirements. Regarded to this the feasibility study is the most important phase in the software lifecycle, as the requirements identified and specified in this phase form the basis for the subsequent phases. Thus, this document could also be called the "early requirements document". The feasbility study describes the system from the technical point of view. Marketing and other such studies must be conducted separately.

7. ASSESSMENT OF THE EARLY REQUIREMENTS COLLECTING IN OMT++

In this chapter I will present an assessement of the early requirements collecting of OMT++ from the human-centred design approach. The assessment is based on the criteria, which are described in the section 5.5. I will concentrate on finding answers to the assessment questions and present some ideas for the development of OMT++ method more human-centred.

7.1 User context and early design

How does the method define the different users or stakeholders and their characteristics for the new system?

Jaaksi et al. (1999) stress that requirements are collected from various sources, like customers, developers, marketing personnel and so forth. However, they do not give any directions for the definition of any specific user groups or stakeholders of the system. The users of the system are defined in use cases as actors and their user role is defined more detailed. However, only the most common and important work process is described as use cases. This can cause that some user groups or stakeholders are not noticed at all during requirements capture.

This view can be critised that the user and stakeholder analysis is the job of marketing department. However, as Maguire (1998) has pointed out there may be indirect users who may be affected by the outcome of a direct user using the technology and most obviously they are not a main interest of the marketing persons.

Constantine (1995) noticed that what is really important in developing a system is not to build software around users, but around uses. Constantine (1995) has

extended Jacobson's "use cases" to apply to user interface design. These use cases are called essential use cases. The main idea is that users interact with a system in various roles. A role is an abstract relationship between user and system. Jaaksi et al. (1999) describe actors similar way, according to them "the use case model uses actors to represent roles that the users can play..." (p. 13).

OMT++ does not say explicitly how to describe users and stakeholders of the system and their characteristics. Use cases give some idea of the users and their roles, but the limitation is that they cover only the most important and frequent tasks and due to this also the most important and frequent users. There can be the indirect users as Maguire (1998) pointed out and in OMT++ they are propably dismissed. To help later stages of the requirements capture the user role map can be used to describe different users role as suggested by Constantine (1995).

Are the current tasks or the future system tasks described?
In OMT++ tasks are partially described. According to Jaaksi et al. (1999) requirements collection is started based on added-value features by discussing with potential users. After an intensive collection phase the requirements statements are written down. These requirements statements create the basis for the analyzing the functionality of the system in terms of user operations in the later stage of the development process. In OMT++ functional requirements can be called tasks and non-functional requirements the framework for functional requirements, which set constraints and limits for the implementation (Jaaksi et al., 1999).

The most important functional requirements are written down as use cases. One potential shortcoming of use cases is that each case only describes one subtask of the total workflow (Jaaksi et al., 1999). The numbered list, as requirements are described, does not give the best possible way to handle the total workflow. Also the division between system and user tasks is omitted in OMT++.

Have the characteristics of the environment been considered?
In ISO 13407 (1999) the environment is defined to include the hardware, software and materials. Also relevant characteristics of the physical and social environment include the definition of ISO's environment.

In ISO 9241-11 (1998) environment is divided to three different phases: organizational, technical and physical environment. And these are further divided to subphases.

There is no specific stress on environmental issues in OMT++. However, non-functional requirements specify how closely the functional requirements must be met regarding attributes such as responsiveness, capacity, and usability (Jaaksi et al., 1999).

Is there a phase for producing the new design ideas?
According to Jaaksi et al. (1999) feasibility study should answer questions like what kind of different implementation solutions can be created and what is the most feasible solution. OMT++ study the feasibility of the planned system concurrently with requirement capture and collected requirements are the basis of different solution alternatives (Jaaksi et al., 1999). However, it should be noticed that OMT++ covers the whole system not just interactive part of it.

According to Maguire (1998) the aim of human-centred user requirements is to develop one or more ideas upon which the new design would be based. He regards each idea as a system concept. According to ISO 13407 (1999) potential design solutions are produced by drawing on the established state of the art, the expierience and knowledge of the participants. Furthermore, using simulations, models and mock-ups or other forms of prototype allows designers to communicate more effectively with users and reduces the need and cost of reworking (ISO 13407, 1999).

How are the design ideas and concepts been analysed?
Feasibility study of OMT++ is typically performed by a group of the best experts and based on alternatives the expert has created (Jaaksi et al., 1999). However, OMT++ does not give any details how these alternatives are analysed.

According to Maguire (1998) this phase consist of the review and evaluation of all the ideas proposed for the new system. ISO 13407 (1999) stresses that evaluation is an essential step in human-centred design and should take place at all stages in the system lifecycle.

7.2 Prototype and user testing

Are the general usability goals been set to use as a basis for establishing user requirements?
In OMT++ usability goals are expressed as non-functional requirements and as a part of use cases. The earlier articles of Jaaksi (1995, 1997, 1998) do not mention usability requirements. In the latest writing of Jaaksi et al. (1999) usability requirements are mentioned as a part of use cases. However, there are no specific details concerning what these usability goals are and how they can be defined.

According to Maguire (1998) the relevant usability goals are defined on the basis of the use. If the users are the general public who are to use an information system on a walk up and use basis the important usability goals are different than the situation that the users are using e.g. an air traffic control system.

According to ISO 9241-11 (1998) in order to specify usability it is necessary to identify the goals and to decompose effectiveness, efficiency and satisfaction and the components of the context of use into sub-components with measurable and verifiable attributes. Effectiveness, efficiency and satisfaction

define the overall usability. There can also be additional usability objectives like learnability, error tolerance and legibility. Usability goals can be related to a system's primary goal, subgoal or secondary goal. Focusing usability objectives on the most important user goals may mean ignoring many functions, but is likely to be the most practical approach (ISO 9241-11, 1998).

Has the balance between user actions and system actions been considered properly?

There is not a division to user and system actions during the feasibility and early requirements capture in OMT++. The point of view is user's view. Later on in the development process, when requirements are analysed further, the operations that the user carries out with the application is identified more detailed. However, there is no clear distinction between user and system tasks as the way Preece et al. (1994) suggest.

According to Maguire (1998) the tasks for which the person is using the system comprises a series of sub-tasks or functions of which some are to be performed by the person and some by the system. During the design of the system consideration should be given to what tasks the system should perform and those that the user should perform. It is important that the person is given an acceptable task load.

Is there a prototype and has it been tested with users?

Jaaksi et al. (1999) do not talk about prototyping in the early requirements and feasibility study phase.They mention prototypes in a user interface phase. They critise prototyping mentioning that prototyping can unnecessarily limit the solution. However, Maguire (1998) means with prototyping more like a task walk through instead of the operational prototype with all the functionality.

According to Maguire (1998) in order to help users visualize the possible system and to clarify user requirements, a rapid prototype or simulation of the system can be developed. The prototype can be tested by 'walking through'

specific system tasks. The basic approach is for the evaluator and the user(s) to select a range of suitable task scenarios. Each task is then gone through how it would be carried out with the system, in a step-by-step fashion. According to ISO 13407 (1999) using simulations, models and mock-ups or other forms of prototype allows designers to communicate more effectively with users.

Have the costs and benefits to the users of the new system been considered to ensure that the benefits at least match the costs?
Jaaksi (1997) points out that, different alternatives about how to meet their requirements are studied. Based on the alternatives are produced solution suggestions and effort estimations. The collected requirements, solution suggestions, and effort estimations form the basis for the next phases.

According to Maguire (1998) an important aspect in weighing up alternative system concept options will be the costs and benefits that users perceive, as well as each option's effectiveness in meeting the business goals. The information collected are the costs and benefits of the system concept form each user group's view.

However, there is difference between these two views. Jaaksi (1997) sees the feasibility study as just a technical one. So it does not include market or cost-benefit analysis as part of it. Instead, Maguire (1998) stresses the importance of the user's view and tries to find out the best alternatives for all user groups.

7.3 User requirements documentation

Have the user requirements been assembled into suitable form for input to system development?
In OMT++ the requirements need to be documented in order to make them traceable. They have to be unambiguous the way that it is easy to be seen as having been fulfilled in the testing phase. The requirements are documented in a list that is divided in two parts: functional and non-functional requirements. If some constraints are found, they are documented in a third separate list. The

requirements must also be classified into mandatory and optional categories. When documenting requirements in this way they can be used as a basis of the analysis document.

According to Maguire (1998) the potential user requirements should be considered as a basis for agreed user requirements, which will form the basis of the user requirements specification. The process and idea is similar with OMT++.

7.4 Summary

The most important task of almost any system is to serve the end users. OMT++ process aims at the construction of object-oriented software systems that the end user can use effectively. On the other hand, the main focus of human-centred design is to make interactive systems usable. Seemingly, the goal is the same.

OMT++ covers the whole system development process and as ISO 13407 (1999) points out the human-centred design process should form only a part of the overall system development. Human-centred design applies to all phases of a project. It can have greater impact in the initial project stages when key decisions about the project design and scope are being defined. During later stages, when major design decisions have been made, a narrower scope is more appropriate.

Even if OMT++ method does not follow all the human-centred activities, it is a good base to build a more human-centred design process by selecting the relevant human-centred design methods and techniques to support the overall system development process.

According to the use case approach taken in OMT++, users perform work by carrying out sequentially related operations on the system. A use case is a specific way of interacting with the system by performing some part of its

functionality. One problem is that the use case model usually is written with the software system as the focus of attention. They mean that the use cases give too little priority to the end users and that each use case is a definition of user actions by software engineers. To overcome these problems it is necessary to model the use cases in participation with the end-users. Otherwise there is a great risk that the application will not support the users efficiently in their work.

8 EVALUATION, DISCUSSION AND CONCLUSIONS

This research tried to find out what human-centred design means when collecting user requirements and how can we evaluate these characteristics in methods other than human-centred ones. Many current development processes do not take a human-centred approach, and thus fail to incorporate feedback from users and identify the full range of user needs.

In this chapter I will evaluate just of all the usefulness of assessment criteria developed during this research followed by the discussion and conclusions.

8.1 Evaluation of assessment criteria

A general approach to specifying user requirements needs to be flexible to meet different situations. It is important to understand that specification of user requirements is an ongoing process. The process is iterative so that items of information are noted, then modified or firmed up. If development method is only assessed with criteria, the answers do not necessary measure the iterative nature of the process.

ISO 13407 (1999) describes the four human-centred design activities that need to take place at all stages during a project: 1) understand and specify the context of use; 2) specify the user and organizational requirements; 3) produce design solutions; and 4) evaluate designs against requirements. These four aspects are taken into consideration in assessment criteria. The assessment criteria gives an idea at least of what kind of information should be collected during early user requirements phase and does the method under evaluation gives that information. Many critical acitivies for user requirements collection are missing from conventional development processes and these criteria are trying to point out those deficiencies.

8.2 Discussion

The main goal of my thesis was to provide guidelines, which can support the capture of the user requirements and designing of usable systems. Many people think that usability can be added to a system just by providing a specific style of interface. However, usability is much deeper than the superficial features of the user interface. While user interface features are important in helping shape the usability of the system, simply employing a good set of widgets does not guarantee usability. What then is usability? It can be thought of as quality of use, a quality of the interaction between user and system. Usability is more difficult to reason about than some other qualities of software products. Usability depends on users, software and the specific tasks people want to perform.

According to Macleod (1994) the main problem is that despite the fact that usability has been acknowledged as an important software quality, it has remained as a fuzzy concept, which has been difficult to evaluate and measure. As a consequence it is often not explicitly identified as a part of user requirements and product specification.

An important distinction to make when considering usability methods is the difference between user interaction and software. Many developers think of their usability methods efforts in terms of "evaluating software" or "evaluating user interface software." Software is the code running behind the interface screen and can be evaluated, but not for the purpose of usability.

Usually, the user interface is designed by software engineers who seldom have the time, interest, or competence needed to design a usable interface. Software engineers and designers have different priorities. Software engineers are usually more interested in fulfilling the functional requirements of the system. The designer, on the other hand, is more concerned about the usability of the

system. Both priorities are of course important why it would be valuable to include a usability expert that could focus on the user interface design only and thereby have better possibilities for gaining experience and skill.

Involving the users in the development project is undoubtedly important. The users are the only experts on the work to be supported by the system and they should therefore have a great influence on the result. However, when working in a human-centred development project it is important to carefully consider how to involve the users in the most beneficial way. The users are experts on their work, not on designing interfaces. The analysis of the users' work should only describe the contents of the users' work, not the style of the interface.

Usually introducing changes into an organisation is a lengthy, costly and complicated process. It requires convincing many people to invest time and money and then demonstrate the benefits versus costs. To get over all the barriers needs a cultural change in the expectations and a change in priorities in development from meeting technical specifications to meeting the widest possible range of user needs. Usability will only be taken seriously when it is part of the acceptance criteria for a system, and this requires a means to specify usability goals and assess their achievement.

8.3 Conclusions

This research focused on user requirements through human-centred design approach, which applies ISO 9241-11 (1998) and ISO 13407 (1999). ISO 9241-11 describes how usability can be specified and measured, and ISO 13407 specifies the human-centred design process, which is necessary to achieve the usability and quality in use goals. For a design process to take a human-centred approach to the identification of requirements, it must include activities that can capture both usability requirements, derived from the capabilities of the end user groups, as well as accessibility requirements for users.

The main research question concentrated on the issue of how human-centred design approach can improve user requirements phase. Traditional approaches to requirements engineering concentrate on identifying functional requirements and ensuring that the developed product meets these requirements. Other non-functional requirements have less importance. Yet from a user perspective, quality in use is critical to successful implementation. According to human-centred design the characteristics of users, tasks and the organizational and physical environment define the context in which the system is used. In order to to guide early design decisions, and to provide a basis of this context, it is important to understand and identify the details of this context. However, this activity is missing from conventional development prosesses.

Requirements analysis is the process of determining what is required of a future system or product. Three kinds of requirement are developed:

- organisation and business requirements,
- system technical requirements, and
- user requirements and functional specification.

All three of these sets of requirements must be carefully developed to ensure the success of the new system. Historically they may have been seen as separate but to achieve a successful design, they should be carried out as part of common business framework (Maguire, 1998). A general approach to specifying user requirements needs to be flexible to meet different situations. However, the main advantages are that developers have a process how to collect user requirements and these requirements can be an integral part of the whole development process. Users can be integrated to the development process already during early requirements capture phase.

In general human-centred design can potentially provide benefits in four areas:

- reduced production costs,
- reduced support costs,
- reduced costs in use, and

- improved product quality (Maguire 1998).

The complete benefits of human-centred design come from calculating the total lifecycle costs of the product including conception, development, implementation, support, use and maintenance. When adopting the human-centredness into the overall design process, the design organization should incorporate human-centred design into their existing internal procedures and development standards. This means that a plan should be developed, which is able to identify how the human-centred design process activities and procedures for integrating these activities with other system development activities.

Based on the findings of this thesis, we can conclude that ISO 13407 together with RESPECT provide a framework, which can help in the capturing of user requirements. However, human-centred design process defined in ISO 13407 focuses on the planning and management of human-centred design. If human-centred design activities are applied to the system development, there is a need to define detailed coverage of the methods and techniques suitable for the process in question.

Human-centred design should result in the ability of particular users to achieve specified goals through the effective and satisfactory use of a product in a given environment. To achieve this objective there is still a need for empirical research how to put this simple goal into the practice.

REFERENCES

Aalto, J-M. & Jaaksi, A. 1994. Object-Oriented Development of Interactive Systems with OMT++. In Proceedings of TOOLS14, Technology of Object-Oriented Languages & Systems, Prentince-Hall, New York, 205 – 218.

ACM SIGCHI. 1992: Curricula for Human-Computer Interaction. http://www.acm.org/sigchi/cdg/cdg2.html. [Checked: 25.4.2001]

Ainger, A., Kaura, R. & Ennals, R. 1995. Executive Guide to Business Success through Human-Centred Systems. Springer-Verlag, London.

Bevan, N. & Macleod, M. 1994. Usability Measurement in Context. Behaviour and Infromation Technology, Vol. 13, No 3, 132-145.

Bevan, N. 1995. Usability is Quality of Use. In Proceedings of the 6th International Conference on Human Computer Interaction, Elsevier Science, New York, 349-354.

Bevan, N. 1996. Integrating usability into the development lifecycle. In Proceedings of 1st International Conference on Applied Ergonomics (ICAE'96). USA Publishing, West Lafayette.
http://www.lboro.ac.uk/eusc/r_usability_papers.html [Checked: 25.4.2001]

Bevan, N. & Azuma, M. 1997. Quality in Use: Incorporating Human Factors into the Software Engineering Lifecycle. In Proceedings of the Third IEEE International Software Engineering Standards Symposium and Forum (ISESS'97), IEEE Computer Society, Los Alamitos, 169-179.
http://www.computer.org/proceedings/isess/7837/7837toc.htm
[Checked: 25.4.2001]

Bevan, N. 1997. Quality and Usability: A New Framework. In E. van Veenendaal & J. McMullan (Eds), Achieving software product quality, Tutein Nolthenius, Netherlands, 25-34.
http://www.lboro.ac.uk/eusc/r_usability_papers.html [Checked: 25.4.2001]

Bevan, N. & Curson, I. 1998. Planning and Implementing User-centred Design. In Proceedings of the conference on CHI 98 summary: human factors in computing systems, ACM, New York, 111 – 112.

Carroll, J.M., Rosson, M.B., Chin, G. & Koenemann, J. 1997. Requirements Development: Stages of Opportunity for Collaborative Needs Discovery. In Proceedings of the conference on Designing interactive systems: processes, practices, methods, and techniques (DIS '97), ACM, New York, 55-64.

Chrusch, M. 2000. The Whiteboard: Seven Great Myths of Usability. Interactions, Vol. 7, No. 5, 13-16.

Constantine, L. 1995. What Do Users Want? Engineering Usability into Software. Reprinted in CHI 2000 Tutorial Notes: L. Constantine & L. Lockwood (Eds), Usage-Centred Design: Practical Abstract Modeling with Use Cases, ACM, New York, 29-40.

Dano, B., Briand, H. & Barbier, F. 1997. A Use Case Driven Requirements Engineering Process. Requirements Engineering, Vol. 2, No. 2, 79-91.

Dayton, T., McFarland, A. & Kramer, J. 1998. Bridging User Needs To Object Oriented GUI Prototype Via Task Object Design. In L.E. Wood (Ed.), User Interface Design: Bridging the Gap from User Requirements to Design, CRC Press, London, 15-56.

Decker, K.M. 1997. Matching User Requirements. IEEE Concurrency, Vol. 1, No. 3, 17-20.

Denning, P. & Dargan, P. 1996. Action-Centered Design. In T. Winograd (Ed.), Bringing Design to Software, Addison-Wesley, New York, 105-120.

Ehn, P. & Löwgren, J. 1997. Design for Quality-in-use: Human-Computer Interaction Meets Information Systems Development. In M.K. Helander, T.K. Landauer, & P. Prabhu (Eds.), Handbook of Human-Computer Interaction, Elsevier Science B.V., Amsterdam, 299-313.

Faulkner, X. & Culwin, F. 2000. Enter the Usability Engineer: Integrating Usability and Software Engineering. In Proceedings of 5th annual SIGCSE/SIGCUE conference on Innovation and technology in computer science education (ITiCSE 2000), ACM, New York, 61 – 64.

Frøkjær, E., Hertzum, M. & Hornbæk, K. 2000. Measuring Usability: Are Effectiveness, Efficiency, and Satisfaction Really Correlated? In Proceedings of the CHI 2000, ACM, New York, 345-352.

Gause, D.C. & Weinberg, G.M. 1989. Exploring requirements: Quality before Design, Dorset House, New York.

Gilb, T. 1997. Requirements-Driven Management: A Planning Language. Cross Talk, Vol. 10, No. 6, (web magazine)
http://www.stsc.hill.af.mil/CrossTalk/1997/jun/requirements.asp.
[Checked: 20.4.2001]

Gill, K.S. 1996. The Foundations of Human-centred Systems. In K.S. Gill (Ed.), Human Machine Symbiosis: The Foundations of Human-centred Systems Design. Springer-Verlag, London, 1-68.

Gould, J.D. & Lewis, C. 1985. Designing for Usability: Key Principles and What Designers Think. Communications of the ACM, Vol. 28, No. 3, 300-311.

Gould, J.D., Boies, S.J. & Ukelson, J. 1997. How to Design Usable Systems. In M.K. Helander, T.K. Landauer & P. Prabhu (Eds.) Handbook of Human-Computer Interaction, Elsevier Science B.V., Amsterdam, 231-254.

Graefe, T.M. 1998. Transforming Representation in User-centred Design. In L.E. Wood (Ed.), User Interface Design: Bridging the Gap from User Requirements to Design, CRC Press, London, 57-79.

Grudin, J. 1992. Utility and usability: research issues and development contexts. Interacting with computers, Vol. 4, No. 2, 209-217.

Grudin, J. 1995. Interactive Systems: Bridging the Gaps Between Developers and Users. In R.M. Baecker, J. Grudin, W.A.S. Buxton & S. Greenberg (Eds.), Readings in Human-Computer Interaction: Towards the Year 2000, Morgan Kaufmann Publishers, Inc., San Francisco, California, 293-303.

Hirsjärvi, S., Remes, P. & Sajavaara, P. 1997. Tutki ja Kirjoita. Kirjayhtymä Oy, Helsinki.

Holtzblatt, K. & Jones, S. 1995. Conducting and Analyzing a Contextual Interview. In R.M. Baecker, J. Grudin, W.A.S. Buxton & S. Greenberg (Eds.), Readings in Human-Computer Interaction: Towards the Year 2000, Morgan Kaufmann Publishers, Inc., San Francisco, California.

ISO 14598-1. 1999. Information technology – Software product evaluation. Part 1: General overview. International Organization for Standardization, Geneva, Switzerland.

ISO 13407. 1999. Human-centred design processes for interactive systems. International Organization for Standardization, Geneva, Switzerland.

ISO 9241-11. 1998. Ergonomic requirements for office work with visual display terminals (VDTs). Part 11: Guidance on usability. International Organization for Standardization, Geneva, Switzerland.

Jaaksi, A. 1995a. Object-Oriented Specification of User Interfaces. Software Practice & Experience, Vol. 25, No. 11, 1203 – 1221.

Jaaksi, A. 1995b. Implementing Interactive Applications in C++. Software Practice & Experience, Vol. 25, No. 3, 271 – 289.

Jaaksi, A. 1997. Object-Oriented Development of Interactive Systems. Tampere University of Technology, Tampere, Ph.D. Thesis.

Jaaksi, A. 1998a A Method for Your First Object-Oriented Project. Journal of Object-Oriented Programming, Vol. 10, No. 9, 58 – 65.

Jaaksi, A. 1998b. Our Cases with Use Cases. Journal of Object-Oriented Programming, Vol. 10, No. 10, 58 – 65.

Jaaksi, A. (Ed.), Aalto, J-M., Aalto, A. & Vättö, K. 1999. Tried and True Development: Industry-Proven Approaches with UML. Cambridge University Press, New York.

Järvinen, P. & Järvinen, A. 1996. Tutkimustyön Metodeista. Opinpaja Oy, Tampere.

Karat, J., 1997. Evolving the Scope of User-Centered Design. Communications of ACM, Vol. 40, No. 7, 33-38.

Karat, J., Karat, C.M. & Ukelson, J. 2000. Affordances, Motivation, and the Design of User Interfaces. Communication of the ACM, Vol. 43, No. 8, 49-51.

Kramer, J., Noronha, S. & Vergo, J. 2000. A User-Centered Design Approach to Personalization. Communications of ACM, Vol. 43, No. 8, 45-48.

Kirakowski, J. (Ed.) 1996. User-Centred Design, version 1.2. INUSE project, the EC Telematics Applications program, England.

Landauer, T. 1993. The Trouble with Computers. MIT Press, Cambridge, Massachusetts.

Madsen, K.H. 1999. The diversity of usability practices. Communications of the ACM, Vol. 42, No. 5. 61-62.

Maguire, M. 2000. Increasing the Influence of Usability Practices within the Design Process. In Extended Abstracts CHI 2000, ACM, New York, 305.

Maguire, M. 1997. RESPECT User-Requirements Framework Handbook. Version 2.21. HUSAT Research Institute, Leics, UK.

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

Mayhew, D.J. 1998. The Usability Engineering Lifecycle. In Proceedings of CHI 1998, ACM, New York, 127-128.

Nielsen, J. 1992. Finding usability problems through heuristic evaluation. In Conference proceedings on Human factors in computing systems: CHI '92, ACM, New York, 373 – 380.

Nielsen, J. 1993. Usability Engineering. Academic Press, Inc., San Diego.

Norman, D. A. 1998. Want Human-Centered Development? Reorganize the Company. In D.A. Norman, The Invisible Computer, Chapter 10, MIT Press, Cambridge, Massachusetts.
http://www.nngroup.com/reports/want_hcd_reorg.html [Checked: 25.4.2001]

Norman, D.A. & Draper, S.W. (Eds.). 1986. User Centred System Design: New Perspectives on Human-Computer Interaction. Lawrence Erlbaum Associates, Publishers, London.

Pohl, K. 1996. Process-Centered Requirements Engineering. John Wiley & Sons Inc., England.

Preece, J. & Rombach, H.D. 1994. A Taxonomy for combining software engineering and human-computer interaction measurement approaches: towards a common framework. International Journal of Human-Computer Studies, Vol. 41, No. 4, 553-583.

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

Robertson, S. & Robertson, J. Mastering the Requirements Process. Addison-Wesley, New York.

Rosson, M.B. & Carroll, J.M. 1995. Integrating Task and Software Development for Object-Orientied Applications. In Proceedings of CHI '95, ACM, New York, 377-384.

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

Sutcliffe, A. 1996. A Conceptual Framework for Requirements Engineering. Requirements Engineering, Vol. 1, No. 1, 170-189.

The Standish Group International, Inc., 1995. Chaos Report. http://standishgroup.com/visitor/ [Checked: 25.4.2001]

Wixon, D. & Whiteside, J. 1985. Engineering for Usability: Lessons from the User Derived Interface. In Proceedings of the CHI'85, ACM, New York, 144-147.

Zave, P. & Jackson, M. 1997. Four Dark Corners of Requirements Engineering. ACM Transactions on Software Engineering and Methodology, Vol. 6, No. 1, p. 1–30.