UPPSALA
UNIVERSITET

Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology 5

Design in Telemedicine

Development and Implementation of Usable Computer Systems

BY
ERIK BORÄLV

ACTA
UNIVERSITATIS
UPSALIENSIS
UPPSALA
2005

ISSN 1651-6214
ISBN 91-554-6133-6
urn:nbn:se:uu:diva-4760

Dissertation presented at Uppsala University to be publicly examined in Rum 1211, Hus 1, Polacksbacken, Uppsala, Friday, February 4, 2005 at 10:15 for the degree of Doctor of Philosophy. The examination will be conducted in Swedish.

Abstract

Borälv, E. 2005. Design in Telemedicine. Development and Implementation of Usable Computer Systems. Acta Universitatis Upsaliensis. *Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology* 5. 54 pp. Uppsala. ISBN 91-554-6133-6

Designing computer systems that effectively support the user is the major goal within human-computer interaction. To achieve this, we must understand and master several tasks. This process must initially deal with the question of knowing what to develop and later, with the question of knowing how to design and develop the system. This view might seem off-target at first, since it does not explicitly mention the goals or functions of the system. However, more often than not, there is no objective goal to aim for that can be formally specified and used as a target criterion that will signal when we have designed an appropriate system. Instead, there is a large set of vague goals – some of which may last through the entire project and some that will not. It is therefore somewhat confounding that most of the current methods of systems development require that these goals are explicitly laid out, in order to steer development.

For researchers in Human-Computer Interaction, the existence of many varying – and possibly conflicting goals – presents is a great challenge. The constructive main focus on producing usable systems is a matter of understanding this complex situation and knowing how to proceed from there.

There are many existing approaches that can be used to carry out this complex development process. This thesis presents one approach, based on the notion that the elements that constitute a successful system are also a part of the solution.

This thesis presents this approach as it is applied to the development of systems for computer-supported work in health care. The projected solution suggests that we need to focus more intently on active user involvement in iterative development that is significantly long-term. The traditional, rather narrow circle of focus that encompasses design, development and evaluation is not sufficient.

Keywords: design, telemedicine, human-computer interaction, usability, development

Erik Borälv, Department of Information Technology, Box 337, Uppsala University, SE-75105 Uppsala, Sweden

ISSN 1651-6214
ISBN 91-554-6133-6
urn:nbn:se:uu:diva-4760 (http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-4760)

Parts of the thesis

This thesis is based on a number of research activities, parts of which have previously been published. The following is a list of the relevant papers and notes on my contribution to them.

1. Domain Specific Style Guides – Design and Implementation. Olsson E, Göransson B, Borälv E, Sandblad B, Proceedings of the Motif & COSE International User Conference, Washington D.C. 1993, pp. 133-139.
2. Usability and Efficiency – the Helios approach to development of user interfaces. Borälv E, Göransson B, Olsson E, Sandblad B, Computer methods and programs in biomedicine, supplement volume 45, December 1994, pp. 47-64.
3. A Teleradiology System Design Case. Borälv E, Göransson B. Conference proceedings of Designing Interactive Systems 1997, ACM's Special Interest Group in Computer-Human Interaction (SIGCHI) in cooperation with the International Federation for Information Processing (IFIPWG 13.2), Amsterdam, 18-20 August 1997. ISBN 0-89791-863-0, pp. 27-30.
4. Design and Evaluation of the CHILI System. Borälv E. Technical report, Department of Information Technology. 2004-056. ISSN 1404-3203.
5. Evaluation and Reflections on the Design of the WeAidU System. Borälv E. Technical report, Department of Information Technology. 2004-057. ISSN 1404-3203.

Important changes in the format

Please note that the original format of the three first papers has been edited to fit into this thesis.

Paper 1

This paper describes how to develop a Style Guide for a specific context of use. It discusses what domain knowledge is and how this knowledge can help us to develop better systems.

My contribution to this paper was to describe how to relate graphical components (widgets) to the domain. The main contribution to the project and this paper was the design and implementation of domain-specific components.

Paper 2

This paper is a continuation of Paper 1. It describes how to use a Style Guide to implement a medical engineering environment in a larger setting. It defines a basic model of development that will make better use of the domain-specific aspects of the Style Guide.

My role here was to define how the domain-specific components fit into a development environment.

Paper 3

In this paper, the underlying design process is presented in the form of a method. It introduces the concept of Design Patterns (or Design Criteria). It describes how criteria and requirements were transformed into a physical interface design. It further shows how the concept "Work Task" (as defined in Papers 1 and 2) can be used as the primary approach when developing a system.

I took part in all design work and method compilation.

Paper 4

This paper summarizes the design approach used in the CHILI system. It contains an evaluation of the system after a number of years in clinical use.

I performed the evaluation and wrote the paper. I was one of the original designers of the system and I still remain involved in the project.

Paper 5

This paper presents the original ideas behind the computer support system called WeAidU. An evaluation of the system is included.

I performed the evaluation and wrote the paper. I was part of the development project 1999-2000, and was involved in the initial design and deployment.

Contents

Introduction 7
Purpose and goal 7
Research and work performed 8
Result 10
Perspective 10
The “right” solution 12
Disposition 14

Methods 15
Viewpoint 15
Research approaches 15
Evaluation 16
Action Research 18
Central problems 19

Human-Computer Interaction 22
Research questions 25
A slice of software development history 27
In the beginning 27
Integrated development environment 27
Development today: a moving target 28
Model-based GUI 29

Medical Informatics 30
Telemedicine and telemedical applications 30
The user and the importance of utility and usability 31

Design 33
Helios (Papers 1 & 2) 34
Project description 34
Solution 35
MEDICUS/CHILI (Papers 3 & 4) 35
Project description 35
Solution 37
WeAidU (Paper 5) 38
Project description 38
Solution 40

Conclusion 42
Guidelines 45
Design methods 46
Development methods 47
More is more, less is better 47

Summary in Swedish 49
Design i telemedicin – utveckling och konstruktion av användbara datorsystem 49

Acknowledgements 50

References 51

Introduction

Purpose and goal

While in high school, my first impression of software development was one of fascination. It appealed to me that one could build systems based on logic, to perform actual and valuable tasks. With structure and careful planning one could make beautiful graphs of x^3 curves or an archive for one's vinyl collection.

Even later, with formal training and education in computer science, this fascination still dominated. Where did this sensation come from and what makes software development interesting? Obviously "hot" items such as fast computers or special effects are, by their nature, interesting to work on. But any problem can be made interesting if it poses the right type of challenges. Vacuuming is not one such problem – but someone challenging a teenager (for instance) to clean up his whole room in ten minutes, redefines this as a problem that could potentially be interesting.

What makes software development interesting is a combination of interesting tools, well-formulated problems and high standards. High standards are said to explain some of Apple's success and good reputation – its president, as well as the entire company, are all about high standards [21]. Almost all of the company's products behave better than expected, are "user friendly" and look good – even many years later. You can tell an iPod is a good product just by looking at the case it comes in.

"High standards" may initially seem to be a strange explanation, especially when left to stand on its own. But, there is a continuance and it is about challenges.

Programming is largely about challenges. It is also an activity where it is possible to experience "flow" – a state of focus that occurs when one is engaged in challenging tasks that demand intense concentration and commitment. According to Mihaly Csikszentmihalyi[1], flow occurs (see *Figure 1*) when a person's skill level is perfectly matched to the challenge level of a task that has clear goals and provides immediate feedback [16].

[1] Pronounced chick-sent-me-high-ee

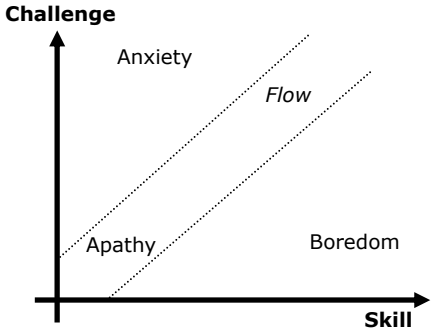

Figure 1. Csikszentmihalyi saw optimal activities in the flow channel moving outward as skills are gained, and certainly before apathy sets in. This is perhaps a parallel to Vygotsky's Zone of Proximal Development, described as "*the distance between the actual development level as determined by independent problem solving and the level of potential development as determined through problem solving under adult guidance or in collaboration with more capable peers.*" [63]

One particular element of programming is user interface design and construction. It requires the skills needed for any kind of software development, in addition to being subject to the constraints that arise when building something for human use. Even in naïve settings, the implications of "human use" are striking; it has to be good enough for the users, meaning that the user interface has to present the various possible actions in an understandable way; the user must be able to use the interface to perform her desired actions; and finally, the user must understand the outcome of the chosen actions. The domain of computer support in medicine – the area dealt with in this thesis – is not at all a naïve setting.

My motivation for working in the fields of Human-Computer Interaction and Telemedicine has been to increase my own skills and to learn how to build systems that are of help to the users. The purpose of my research has been to examine some of the ways that this particular kind of software is designed. The way in which software development is carried out affects usability. I discuss different ways to design and construct software used within the field of telemedicine, in order to increase usability for users in health care.

Research and work performed

The work presented in this thesis relates to three different research and software development projects in the medical domain. In all of the projects, our

department was given the general task of being responsible for usability issues of the resulting system as well as the explicit task of designing the graphical user interfaces

The selected projects had a particular requirement in common: they were to deliver solutions at a certain point in time. They were, in fact, not research projects with concealed commercial qualities – they were exactly the opposite. The implication is that planning and work strategies must ensure that the project delivers in the end and my research activities had to adapt to this situation.

I was part of the three development projects from start to finish (some projects are still ongoing) and the area of user interfaces was my responsibility. The approaches are therefore selected to fit into a setting which is no different from any typical commercial software development project.

The projects span a substantial time frame, starting in 1992 and continuing up until today. Naturally, the work has been affected by the trends and methods that dominated at various times throughout this period.

Helios consisted of a large university consortium. The primary goal of this research project of the CEC (Commission of the European Communities) was to build a software framework suitable for the medical domain, in order to facilitate the development of medical applications. The framework consisted of many modules and each module was a research topic in its own right. Our task and research goal was to examine how to gain and pass on design knowledge in static forms, such as Style Guides and Widgets (reusable visual components). We based our research on the assumption that it is possible to express this kind of design knowledge once and for all – more or less. All necessary design knowledge and advice would be compiled into written advice and complemented by pre-designed, re-usable elements. In this context, my work consisted of looking into how one could transform domain knowledge into static descriptions, such as pre-designed widgets.

For the second project, *Medicus*, the research question remained more or less the same, but we looked at other, more refined ways to acquire design knowledge. The design decisions were not taken in advance, rather they were made during development process. Again, the decisions were documented as static knowledge, in the form of (almost universal) “patterns” that also could be understood and used outside of this project. In this context, I looked at how one could describe the design decisions so they could be understood by others, and how to document them so that the knowledge would not disappear over time.

Representing a combined third project, the *CHILI* and *WeAidU* computer and support systems both reflect a transition away from the generation of static design knowledge. Both projects implicitly reject the idea of upfront design or even requirements. As an alternative, the starting point is more or less a blank paper, and the design and development process is instead adapted to handle design decisions as they occur. However, this adaptation is

more than just trial-and-error and it is not a random process. In this context, I took a very active role in designing or shaping the framework (the overall design) so that subsequent changes in the details were still possible. The initial research concentrated on how to design this complex software, so that it would still be easy to use. Later on in the project, there was a shift in focus towards more overall usability questions.

Result

Based on the knowledge and experience in software engineering, human-computer interaction and health care, my research shows that we need to acquire a deeper understanding of what characteristics the computer support should encompass, in order to succeed in developing usable systems. This understanding can be enhanced by including users in the development process, by producing many intermediary prototypes and by making progress in fine increments. A major conclusion is that the design and development process never ceases – if it does, the system will fail to function in a real-life workplace. The way in which computer support is introduced and brought into use also plays a vital role in determining the outcome. If it is introduced in a way that stakeholders approve of, the better the odds are that it will be accepted in the long run.

The idea is to make the computer system flexible enough so that the cost of late changes is less than the cost of early changes. The way in which software is produced must be set up so that changes do not hinder meaningful development. This challenges the methods that are chosen, but more importantly it challenges the skills and self-confidence of the developers. If the developers feel confident, they are able to re-work the implementation and use the change to improve the internals as well as the resulting system. However, if the developers do not feel that they have the necessary skills then the introduction of new requirements will become a burden that will complicate further advancement.

The process of going from abstract requirements to a visual representation is a key target, as it involves many of the elements that we believe affect the usability of the finished product. It is very likely though that the chosen method must contain a prototype-oriented approach [65] that can steer the process.

Perspective

In academic situations, researchers want to discuss – or at least be familiar with – the various perspectives that other fellow researchers draw on. As you would expect, it makes a difference in the field of Human-Computer Interac-

tion (HCI) if one has a background in Cognitive Psychology or if one's background is in the field of Ethnography, for instance. My formal background is in Computer Science which deals with the technical side of how to build computer systems.

Scandinavian software designers, who wanted to make systems design more participatory and self-ruled, turned to prototyping in the early 1980s [18, 10, 11]. By using prototypes, developers sought a pro-active way for users to develop a joint consensus on what they needed from a computer system [9]. Prototypes provided a common language for developers and users; a way to test solutions iteratively and to implement industrial democracy in the workplace. The same way of reasoning can be seen in the field of architecture.

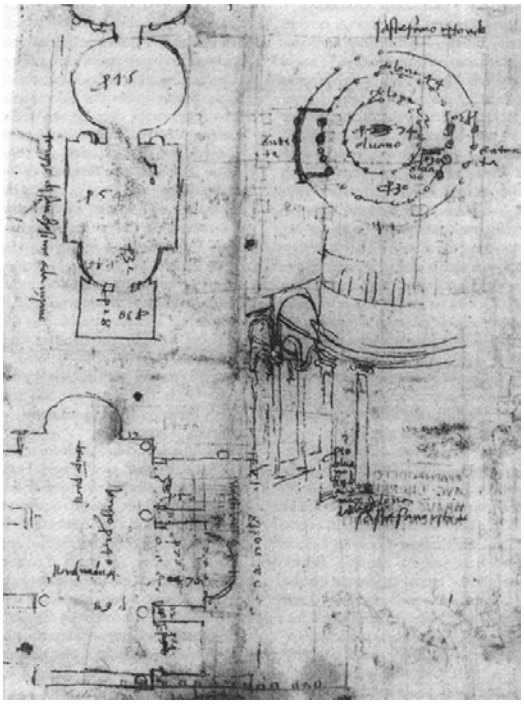

Figure 2. Illustrations from the Trattato di architettura. c. 1470. Biblioteca Nazionale, Turin.

As early as the 15th century, *Francesco di Giorgio* and other master builders in Sienna started collecting drawings that contained examples of working solutions to general problems (see *Figure 2*). More recently, *Christopher Alexander*, a mathematician but foremost an architect, suggested in his thesis "Notes on the Synthesis of Form" from the 60s, that we should use new types representations as tools to facilitate communication between the designers of buildings and the people that were to live in the building [2]. This

kind of tool or language is now known in the computer business as “design patterns.” [1]

This striking view of making computer support more *democratic* inspired others who became interested in user-centered design; information designers began to employ prototyping as a way to encourage user participation and feedback in design approaches. Prototyping is nevertheless seen as a method that meets very different needs in Scandinavia and elsewhere in the world. As a result, diverse development approaches have implemented prototyping quite differently, have deployed it to meet quite different goals, and have tended to understand prototyping results in different ways [58].

This diversity is also reflected in the background of our department – it was established early in the computer age, when focus was often exclusively on computerizing manual (office) work practices. This focus differs from that of many younger HCI institutions that often carry a dissimilar view, and where the objective, in essence, is to look for innovation, pleasure, novel interaction and ground-breaking change, using computer technology. In contrast, our department is still rooted in making everyday work life better in small, steady increments. This is partly a reflection of the domains that we work in. For the most part, these domains are areas where safety-critical aspects and highly specialized work skills are emphasized. The main areas include train traffic control, high-speed ferry operations, train cab operations, health care and dynamic process control.

For my own part, the most attractive aspect of our perspective is the way development is performed. It is always a cooperative effort, where exploration of ideas and solutions play a vital part. Just as Schön [54] talks about *reflection-in-action*, the aim of my research is to investigate the possibilities of translating direct experience from practice into a form that makes sense to the academic audience as well.

The perspective itself, i.e. the background and experience, is essential to this kind of research since the researcher him/herself is the most important research instrument.

The “right” solution

Education, training and scientific methods improve the chances of being able to build usable systems. A usable system is fine, but is not the same thing as the “right” system. As a student, you study theory and in the best case, also train on the practical side – how to put the theory into use. This is a way to ensure that one can solve the given problems. However, being able to solve problems is not always enough if one is particular about how to define usability [26].

Take a common problem in computer science – sorting data. The solution that most people suggest when asked for a sorting algorithm is something

called Bubble Sort[2]. It is the simplest way to sort a list of objects. Unfortunately it is also one of the slowest ways! The Bubble Sort solution is correct, but is not the "right" way to sort data.

> *The problem is not so much the day to day management. Really good hackers are practically self-managing. The problem is, if you're not a hacker, you can't tell who the good hackers are. A similar problem explains why American cars are so ugly. I call it the design paradox. You might think that you could make your products beautiful just by hiring a great designer to design them. But if you yourself don't have good taste, how are you going to recognize a good designer? By definition you can't tell from his portfolio. And you can't go by the awards he's won or the jobs he's had, because in design, as in most fields, those tend to be driven by fashion and schmoozing, with actual ability a distant third. There's no way around it: you can't manage a process intended to produce beautiful things without knowing what beautiful is. American cars are ugly because American car companies are run by people with bad taste.*
>
> Paul Graham, on Great Hackers [21]

The view one should have in regards to solutions, is that the output of design is a *design space* rather than a single solution [39]. The Design Space contains all possible solutions, but all the solutions are not necessarily suitable. This approach contrasts with the traditional concept of design, which assumes that the eventual output is simply a specification or artifact. Only when an association to the Design Space is made, is it possible to analyze just how good the proposed solution is. The right solution is something that needs to be found through exploration and by intentional choice.

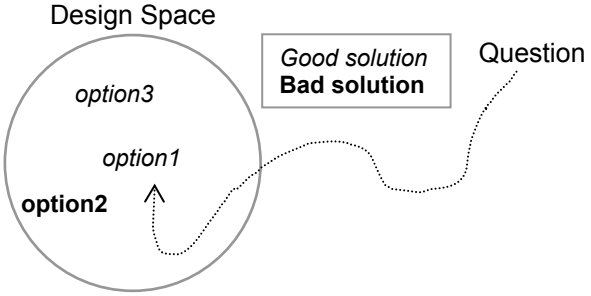

Figure 3. The Design Space (DS) contains all possible Options (solutions) to the Question. For most questions, the DS is large. A novice student or a designer in training is happy just to find any solution in DS, but when striving for high standards, we need to find the "right" or best possible option in the DS.

[2] The idea is to compare two adjacent objects, and to swap them if they are in the wrong order. The algorithm repeats this process until it has gone through all the data without swapping any items. This way of sorting large bodies of data is the most inefficient sorting algorithm in common use.

Sadly, this is a part of the design domain that is said to be indefinable and said to contain a *design paradox* in that we need to be “right” in order to recognize what is “right”.

Disposition

The following chapter describes the research methods used. It explains how to carry out research in live development projects (action research) but also how to use the more traditional scientific methods.

A description then follows of the field of Human-Computer Interaction (HCI) and how I have interpreted this in relation to my work. This section provides a definition of the research question and it also briefly discusses the way software has developed, from the 1980s and onward.

The next chapter, titled Medical Informatics, deals with computers in health care, and discusses what elements characterize this combination. For the most part, my work has been limited to telemedical systems and this is also explained.

The final chapter discusses the topic of design, including the actual work and research performed within the projects.

Finally, there is a discussion about the results and the conclusions that I have made.

Methods

Viewpoint

The main targets for our department's research have always been *real users*, in *real settings*. The focus is on the end user's situation (as opposed to the contractor's situation, the buyer's role, etc.) and the daily work that is to be carried out. This is usually a task filled with many small and delicate problems – problems that if solved, would result in major, overall improvements as compared to the current situation. In other words, we focus on today's work problems and how to solve the most immediate ones. This is in contrast to many other activities in the HCI area that look into the future and try to find completely new directions of problem-solving, or try to introduce new forms of technology to solve the problems. For us, HCI is not about radical innovation. It is about making improvements in the context of an evolving process.

Taking an active role in the projects where we are doing research is another common practice within our department. In these situations, we are not merely providers of HCI expertise. Rather we are more like regular coworkers, if regarded from an outside perspective. In the most ideal of circumstances, we take part in the project from the earliest phase, follow then the initialization of the project, and finally join the development team as the project evolves. This has been the case in all the projects that are presented; I have been a member of the project group from the starting point (when nobody in the group really knows what to do) up until the finish line, when the product is finally deployed or sold.

Research approaches

As a rule, in HCI, there are three predominant ways to conduct research [50]:

1. experimental studies
2. survey studies
3. observational studies

Experimental design is used to control all external variables and vary only those that are being tested. The strength of the experimental study is its ability to clearly localize the effect of a particular design. The method allows the

study of isolated design factors, but requires a situation where variables are controlled, something that is not always feasible.

Survey studies are useful for describing systems, for detecting strong and weak points, and for suggesting improvements. Surveys, questionnaires and interviews provide a structured approach in which the user assesses factors related to the subject of the particular study. User assessments are introspective and are subject to biases but through empirical verification it is possible to establish reliability and validity of user responses. Furthermore, when comparisons are made between different groups of subjects rather than with an absolute criterion, it may not matter that responses are biased.

Observational studies are reasonably easy to conduct but are open to interpretation. In an observational study, one or several systems may be selected and researchers observe the users. Analysis may be on a purely verbal, descriptive level or based on quantitative measures. Conclusions are tentative since researchers have little or no control over conditions interacting with the system [51]. A factor that has to be taken into consideration is when observing work that is partly your own product. In situations like this, one needs to involve more people to avoid biases.

In terms of the conditions we work in (development projects) it is often only feasible to perform surveys and observational studies. The major part of my work has been in the form of observational studies, complemented with surveys and interviews whenever possible. The experimental approach is not used, due to the nature of the work.

Evaluation

Jakob Nielsen categorizes users' experiences in different dimensions or levels [42]:

1. general knowledge about computers
2. expertise in using a specific system
3. knowledge about the domain

He argues that the level will – or should – have implications for the design of the user interface. A user with extensive experience is able to use a computer system in a better, more efficient way than people without experience. An expert user knows more about how the system behaves when using it. The level of experience can also affect the evaluation process. Methods for evaluation are normally separated into:

- *usability testing* methods, where users are involved
- *usability inspection* methods, where users are not involved

An established method for evaluation is performance measurement where the purpose is to determine whether a usability goal has been reached or not. User performance can be measured by having users carry out pre-defined

tasks while observing relevant aspects of the interaction. One could concentrate on the time needed to complete the task, or on error rates, depending on the specific task at hand. The tests can be performed in a live setting, where the user is performing real tasks, or in controlled laboratory settings where more parameters can be controlled. For the kind of systems I have worked on, evaluation requires skilled users because we are mostly interested in the efficiency of daily use.

There are a number of ways one can carry out an evaluation. For the quantitative aspects (performance and efficiency of the system) absolute measures are possible to attain. The results produce a number of direct measures of performance and efficiency.

In terms of qualitative aspects – for example, the quality of the system from the users' point of view – several approaches are needed to cover all relevant areas. These approaches include questionnaires and expert observations [37, 31]. These are the methods that I have used the most frequently.

Heuristic inspection

An expert evaluation or heuristic evaluation differs from traditional evaluations. It is actually more of an inspection than evaluation and it is performed by trained experts.

Established, pre-defined guidelines are often used to classify the findings. During the evaluation session, the evaluator goes through the system several times and inspects the various dialogue elements and compares them with a list of recognized usability principles (the heuristics). These heuristics are general rules that describe common properties of usable interfaces.

The result is a list or groups of issues that have an impact on the usability of the system [43, 42].

Questionnaires

Questionnaires range from informal versions with hand-tailored questions, to standardized versions that are rigorously tested and validated [52].

The Software Usability Measurement Inventory (SUMI) is a proven method for measuring software quality from the end user's point of view [30]. SUMI is a consistent method for assessing the quality of use of a software product or prototype, and it can assist in the detection of usability flaws before a product is shipped. It is backed by an extensive reference database embedded in an effective analysis and report generation tool.

The Software Usability Scale (SUS) was developed as part of the usability engineering program in integrated office systems development at Digital Equipment Co Ltd., Reading, United Kingdom [13]. It is a reliable, low-cost usability scale that can be used for global assessments of systems usability. The SUS is a simple, ten-item scale giving a global view of subjective assessments of usability. This measure can be used to compare usability across

a range of contexts. Just like the SUMI, it is a validated questionnaire, but with a minimal footprint.

Action Research

"*If you want to know how things really are, just try to change them*"

Kurt Lewin, who coined the term action research [36]

My research has a qualitative approach, since almost all of the work is done within a live setting. The primary reason for doing qualitative research is that one wants to gain a deeper knowledge – the *why* and *how* questions – which cannot be measured in numbers. It is also an expression of the fact that knowledge lies within the development projects. We often label the approach as Action Research (AR), although not necessarily adhering to a particular instance or interpretation of AR – though *participatory action research* is similar to our approach [29].

In social psychology, Action Research emerged before and during the Second World War as a form of research in which the researcher learns about certain group processes by actively participating in or manipulating certain aspects of these processes. Action Research has its academic roots in sociology, social psychology, psychology, organizational studies, and education. In AR literature, one finds a varying degree of rejection of the classical notion of scientific research. This is a reminder of AR's origin where resolution of theoretical issues was of less importance, and finding the solution to social and organizational problems was regarded as more important. AR is based mainly on the premise that most things are done within groups and the insight that working within groups has a fundamental effect, not only on the total outcome but also on the individual members of the group. Action-oriented research is said to generate situation-specific knowledge; it does not deal with the mere application of some pre-existing knowledge.

According to Hopkins [25], action research can be described as an informal, qualitative, formative, subjective, interpretive, reflective and experiential model of inquiry in which all individuals involved in the study are informed and contributing participants. The primary intent of Action Research is to provide a framework for qualitative investigations in complex working situations [35]. It consists of four stages, all repeated throughout the duration of the project: reconnaissance & plan, action, observation, and reflection & revision (see *Figure 4*)

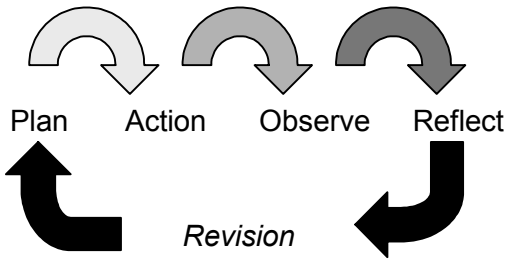

Figure 4. Action Research Stages.

However, during a project it is common to fall back on quantitative methods since they serve as a valuable complement to the free form of observations that otherwise would be the only source of information.

Central problems

One central problem in terms of development is that the heath care world is still not sufficiently computerized. This is not primarily due to a lack of resources, even if this too is an important factor. Instead this insufficiency stems from the difficulty in knowing *what* to computerize and – perhaps even more so – the difficulty in knowing *how* to computerize. Recently, the main criticism has been that the computerizations have been excessively individual. The focus on integration and communication has been lost when many of those involved in computerization have merely been looking to solve their own problems [57].

But simply moving current work practices into the digital domain does not automatically solve any problems or make work better or more efficient. In fact, some computerization has had the opposite effect, for instance, when the selected solution has been inappropriate, or when all aspects of the current situation have not been taken into consideration.

Figure 5. In complex situations there are many technical devices and systems that may interfere with the care process.

One of the primary reasons that this happens is due to the complexity of the situation in the medical domain; a situation that is often too complex to be reduced into a manageable set of constraints that can be implemented into a computer system. There are many reasons for this complexity:

- The care-giving process is strictly focused on the patient. Anything (especially technical systems) that stands in the way of this patient orientation is easily regarded as a severe obstacle.
- In certain circumstances, legislation can be a barrier. It may be a question of economic issues – when someone needs treatment outside her normal service area (in another part of the country, in another country) – or it may be issues of security or confidentiality that prevent rational handling of patient information, etc.
- A hospital is full of professionals with highly specialized skills. Their skills match current needs and thus reflect what they are required to be able to handle in a specific setting. Change the setting and the organization's body of skills will adapt to the change.
- A hospital consists of numerous smaller and often independent departments. Sometimes they overlap in terms of activities, sometimes not. Each department has many local solutions and local technology used to solve their tasks.
- The medical activity in itself is complex and can therefore be difficult to both comprehend and express logically. Much of the internal knowledge

is based on fact, some knowledge is based on experience and some on professional instinct. The health care process is thus very different from biomedical science. The former is more related to the art of medicine, whereas the latter is closely connected to the academic aspects and the biological disciplines of medicine.

- The information that is available can be of poor quality. The process for data collection and for entering data may be of low quality, resulting in poor validity and reliability, as shown by Peterson [49].

Knowing what the problem is that needs to be solved, is the first of the central problems. But the next step, the *how* step, is equally difficult. For even if one knows what the problem is and how it arises, it is still difficult to find a solution. The proposed resolution must also show that real gains will be made in terms of the existing situation, since the existing solution is likely be the first alternative in any competing position. The focus of the medical organization is to treat patients, and if a solution works (although possibly unsatisfactorily) it is likely to remain in use. The health care system is complex and changing from a known solution that works to an unfamiliar system that may or may not be superior, is not always the most obvious and straightforward decision.

Human-Computer Interaction

> *Human-computer interaction is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them.*
>
> ACM SIGCHI Curricula for Human-Computer Interaction [23]

Although the desktop computing revolution has greatly increased the range of possibilities, most users and developers would agree that getting a computer to do specifically what they want it to do is as challenging as ever. Despite the successful introduction of personal computers into nearly all professions, the daily experience of using computers can still generate emotional distress or strain. The world is simply full of poorly designed programs that lack usability in the most elementary ways. If usability aspects are not actively considered during development, then they are not likely to not be considered at all. Genuine usability never happens by pure chance [43].

> *Unfortunately, much of the Web is like an anthill built by ants on LSD: many sites don't fit into the big picture, and are too difficult to use because they deviate from expected norms.*
>
> Jakob Nielsen [44]

The major goal within the field of Human-Computer Interaction (HCI) is to design computer systems that effectively support the user in his/her work task and decision-making. The definition of Usability according to ISO 9241 is:

Usability
: The extent to which a product can be used by specified users to achieve goals with effectiveness, efficiency and satisfaction in a specified context of use.

Effectiveness
: The accuracy and completeness with which users achieve goals.

Efficiency
: The resources expended in relation to the accuracy and completeness with which users achieve goals.

To achieve genuine usability, we must understand and master many tasks (see *Figure 6*). A working system alone is not sufficient, because the system is going to be used by a large number of different people with different needs and capabilities. The conditions of work are likely to change, and the world surrounding the system and its users is definitely going to change. In addition, it is not a question of technical issues alone. Social aspects also play a vital role in any complex work situation [46].

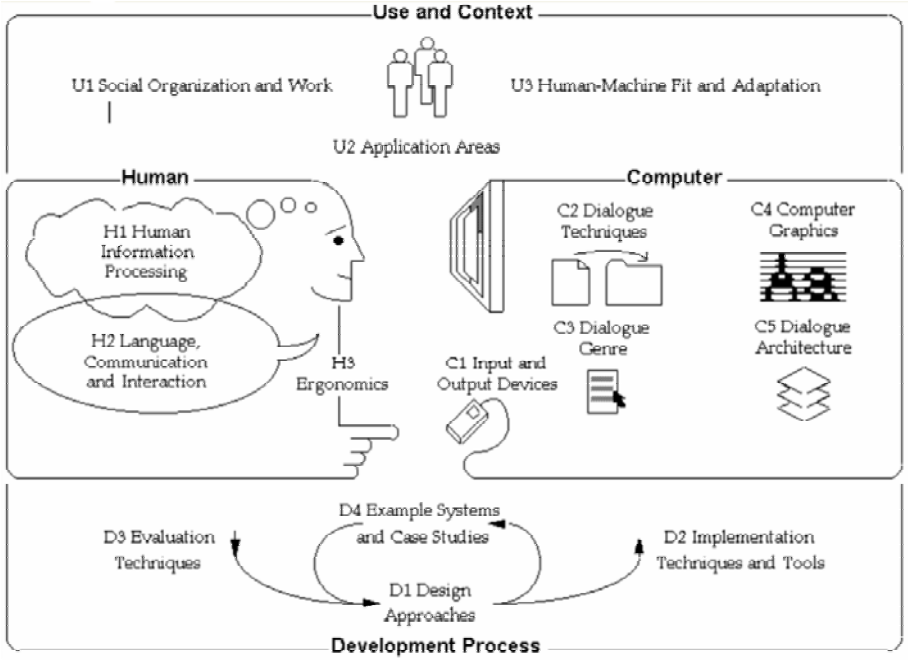

Figure 6. SIGCHI view of HCI. [23]

Being able to understand how users think and perceive information and how people co-operate during work is the first step when dividing the overall development process into smaller elements. It is at this point that the development process first starts.

Much of this initial work within HCI is treated by the underlying disciplines, for example cognitive psychology and software engineering. Those fields could be described as being objective since measurable goals and facts are often in focus. Examples of such facts include the human mind's capabilities in terms of memory capacity, the information flow before decision making, the error rates at a given speed of processing etc. Much of the knowledge in this area forms the basis and theory for the other related HCI research areas.

The next step deals with how to apply these basic facts and considerations in practice, in order to be able to develop computer applications and systems

with high usability [42]. Software design embraces many aspects: function, safety, human interface, ergonomics, graphics, algorithms, and data structure. Correspondingly, these various aspects of software design invariably have an impact on one another. The development process is also equally about what to develop and how to develop it. This *how* is addressed by paying attention to the way software is developed. The solution is to involve end-users and other stakeholders earlier in the process and in more stages of the development process [20]. The proposed solution, combined with modern development strategies, such as prototype-driven development, produces a superior final result [34].

One key purpose of Human-Computer Interaction (HCI) is to ensure that computer support is valuable and useful to users. Research within the field of HCI has contributed to this in many different ways.

At the outset, much of the work within HCI dealt with safety and basic operational issues; many systems and machines were potentially dangerous if used incorrectly, or they could fail, leading to dire economic consequences. When computers grew to be the most common machines that users interacted with, other kinds of problems became more important. Reading and understanding information presented on visual displays units was – and still is – one major research area. Later the actual interaction with visual information became the problem in focus.

Computers and interactive systems now play a very large and fundamental role for many professionals. Many workplaces and work tasks are based completely upon the use of computers; information is stored in computers, only computers can access information, only through the use of computers is information modified and shared, and decisions are channeled though actions taken on computers. The penetration of computer systems into modern working life is indeed remarkable. In January 2004, approximately 96 per cent of all Swedish enterprises with ten or more employees used computers [60].

In all the above cases, the purpose of the computer system is to help and assist in the actual work that is being performed. Ideally, the computer is there to support the user. Computer support, if done properly, is a useful and productive tool that makes work safer, more meaningful, more productive, and even more fun.

However, the state of computer systems is not quite this perfect. Many systems fail to support the work task and instead become burdens themselves, producing more work for the user. Being mechanical in character and encoded to follow certain pre-arranged rules, the computer systems are not likely to perform well at all times, given the changing and non-static nature of companies and work tasks. The clash between the deterministic computer and the not-so-deterministic human being is bound to cause some problems.

However, the situation is not all bad. Computer science knows a lot about how to build systems that are flexible and adaptable. We also know a lot

about human beings, human psychology and the way people work. The computer industry, computer science, HCI and other stakeholders have matured greatly, and today's computer systems have made improvements from both a technical and a usability point of view.

Still, more than 50 percent of software development projects fail to meet their economical and functional requirements. As many as 31 percent of the projects are cancelled before they are completed, since they do not meet the requirements there were originally outlined [59].

Many things can be said about why software development fails and why it is so difficult to succeed. The most common reasons attributed to breakdown are the failure to include end-users in the process, the incomplete and/or changing requirements and the lack of support from management. From a HCI point of view, the lack of usability is a key concern. Most computer users recognize the problems involved in using computers, based on their own experience.

Although the desktop computing revolution has greatly increased the range of possibilities, most users would agree that getting a computer to do what they want it to do, is as challenging as ever. The daily experience of using computers can still produce emotional distress or tension.

Research questions

> *But by searching for that magic metaphor you will be making one of the biggest mistakes in user interface design. Searching for that guiding metaphor is like searching for the correct steam engine to power your airplane, or searching for a good dinosaur on which to ride to work.*
>
> Alan Cooper [14]

Designing computer systems that effectively support the user, is the major goal within human-computer interaction. To achieve this, we must understand and master several tasks. These tasks first address the question of knowing in what direction development should head, and later, the question of knowing in what manner to develop the system.

This view might seem off-target at first, since it does not mention the direct goals or actual functions of the system. However, more often than not, there is no objective goal to aim for that can be formally specified and used as indicator to recognize when we have an appropriate system. This is somewhat confounding since most of the current methods of systems development (see *Figure 7*) focus on and require that these goals are made explicit in order to steer development.

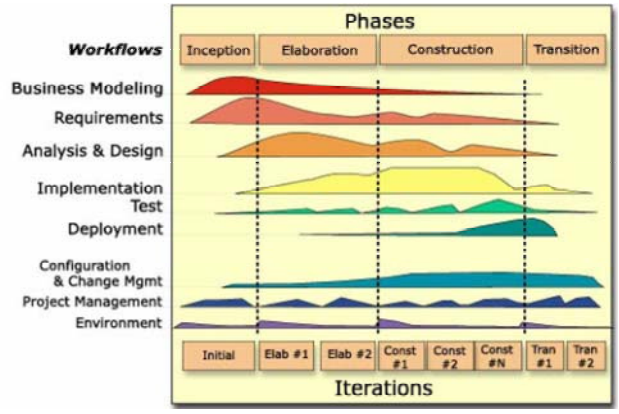

Figure 7. Overview of the flow in systems development, for the IBM Rational Unified Process® [32]. The picture shows a large investment on upfront design early in the project.

When looked at carefully, most computer systems have so many objectives and purposes – technical, organizational or strategic – that limiting the measurement of success of a particular system to an imperfect set of objective goals, is a strategy that does not necessarily lead software development in the right direction. Instead, a successful system can be described in terms of a well-balanced matrix, consisting of many different types of values. There are the objective goals, the functional capabilities and usability aspects – quite often possible to measure and assess. This thesis deals with systems for human interactive use, so it will also discuss values such as pleasure, emotion, long-term expertise effects and the system's potential to grow and expand. These "soft" goals contribute to making the picture more complete in comparison to the picture drawn when merely traditional objective goals are looked at.

For HCI researchers, this situation – which encompasses many varying and possibly conflicting goals, presents a great challenge. The constructive focus on producing usable systems is a matter of understanding this intricate situation and knowing how to proceed from there.

Many approaches exist that can be used to solve this complex development task. The research presented here is targeted towards finding ways – methods and practices – to design and develop computer systems (especially in the medical domain) that will give users the support and the usable system that they need.

In order to succeed in developing an efficient system and user interface, time and effort must be spent analyzing the task, the work contents and the utilization of information in this work task. In short, the application must only do two things: provide the right actions at the right moment and show a

sufficient amount of information. If this can be accomplished, we are in a good position to find a satisfactory solution [45].

But the available screen space will, in all cases, be less than desired and therefore we cannot show all the information we would like to see. Also, we are never exactly sure of what the next action should be, or precisely what information should be displayed. Therefore information has to be organized in a structured way so that we can achieve the best possible compromise between all the elements that affect the system's overall usability.

A slice of software development history

In the beginning

Not too long ago, development of computer applications required only a small number of tools: a text editor to write the code, a complier to transform the programming code into executable code, and perhaps some kind of debugger to systematically trace errors in the system. These tools had little or nothing to do with each other. They were tools in their own right, developed for the specific tasks that they were to carry out. Each tool had to be learned separately. To be able to enter text efficiently, one had to first learn to understand text input, and then learn a little bit about how text editors are built, and then finally learn how to modify the editor to suit one's specific needs. Since the tools were few in number, this could be done with modest effort.

The development flowed from one tool to the other, in a kind of iterative circle of development.

Integrated development environment

> *To paraphrase Fred Brook's wonderful essay "No Silver Bullet," well over half of the time you spend working on a project (on the order of 70 percent) is spent thinking, and no tool, no matter how advanced, can think for you. Consequently, even if a tool did everything except the thinking for you -- if it wrote 100 percent of the code, wrote 100 percent of the documentation, did 100 percent of the testing, burned the CD-ROMs, put them in boxes, and mailed them to your customers -- the best you could hope for would be a 30 percent improvement in productivity. In order to do better than that, you have to change the way you think.*
>
> Allen Holub [24]

Somewhere along the way software designers came to the insight that we might need to integrate the tools in order to narrow the development circle. This would expedite the whole development process. Developers would no longer need to learn all the details of each tool. This integrated development

environment (IDE) combined the minimum set of used tools – the editor, the compiler and debugger – into one streamlined developer environment.

IDEs have been tremendously successful. This success coincided with advances in graphical user interfaces. GUIs have always been (rightfully so) considered very difficult to implement. For a long time, computer graphics was slow because it lacked today's hardware support. Therefore many delicate techniques had to be used and combined to achieve proper speed and usefulness. IDEs made aspects of this work a lot easier; one did not have to learn all the details of the underlying library of graphical components, one could use a visual layout tool to construct the user interface, and one did not have to write any code for this part of the application. The tool would take responsibility for most things related to the user interface. Since there was a library of graphical components from the IDE manufacturer, the manufacturer would see to it that the user interface achieved the desired speed.

This success sparked the development of new IDEs and today we find old utilities packaged into the IDE family, complete with a new user interface and tight integration with other IDEs.

Today's IDEs contain a lot more than their predecessors. While competing to provide IDEs to address every imaginable developer need in one all-inclusive IDE, vendors fail to focus on improving the functionality of each particular tool. Instead, it seems that vendors are providing features that first "cover the earth" by bundling many constituent components, while offering little value to the software development process. The current state of affairs reveals a constant and continual expansion in which IDEs are getting larger and larger. But vendors still have a long way to go before they achieve one, single, rational goal: to make the lives of developers easier, enabling development teams to be more productive and thereby leading to higher-quality software products.

Development today: a moving target

Development used to be simple, but inefficient. Now it has turned efficient but at the same time so difficult that the promise of efficiency is in danger. Development consists of many programming languages, and many operating systems, but the real moving parts come from the underlying technologies upon which we build our systems. First of all, developers probably need to know a minimum of basic programming languages: C++, Java, VisualBasic, and C# [17]. Developers also need to have a reasonable understanding of several operating environments: Unix, Linux, MS-Windows, and MacOS. Next in line come the various data description representations such as HTML, XML, XSLT, DTD, CSS and then there are also the scripting languages that you probably need: JavaScript, CGI, JSP, ASP, Perl, and PHP.

> The most awful part of this problem is that when you've finally got your head around all the various parts involved in today's denouement, you must ensure that all these different parties dance nicely together when they meet up at the resulting debutante's ball.

Model-based GUI

Much of the complexity and problems that arise when implementing user interfaces, originate from the connections between the interface and the underlying system. With a model-based view, one takes the following stance:

> *Model-based interface development is a new paradigm for developing interfaces that offers solutions to the main shortcomings of current tools. The model-based paradigm uses a central knowledge base to store a description of all aspects of an interface design. This central description is called a model, and typically contains information about the tasks that users are expected to perform using the application, the data of the application, the commands that users can perform, the presentation and behavior of the interface, and the characteristics of users.* [41]

If these connections are programmed separate from the interface specification, the result is lots of effort being wasted on writing error prone codes that must be rewritten whenever the layout changes and whenever the underlying system changes. Therefore it is important that these connections form an integrated part of the declarative specification language.

Medical Informatics

> *Medical Informatics is both an Art and a Science.*
>
> *Science: where methods are conceived and experimentally validated by means of computer models and formalisms.*
>
> *Art: where computer processing systems are built and assessed.*
>
> Handbook of Medical Informatics [6]

To a great extent, the practice of health care is an information management task. A physician's decision-making is based upon expert knowledge, information from the individual patient, and information from many previous patients – the latter being known as experience [62]. Decision-making is often very difficult, not only due to the fact that the required expert knowledge in each individual medical field is enormous, and growing daily. Difficulty in making decisions is also due to the fact that the information available for the individual patient is multi-disciplinary, imprecise and very often incomplete [8].

Medical Informatics is located at the intersection of information technology and the different disciplines of medicine and health care. The role of Medical Informatics is to deal with the common set of problems and solutions that relates to both medicine and computer science.

Telemedicine and telemedical applications

> *Telemedicine is conceived of as an integrated system of health-care delivery that employs telecommunications and computer technology as a substitute for face-to-face contact between provider and client.* [4]

The introduction of telecommunication (computers, the internet, mobile phones, and wireless communication) has already changed the way we communicate with each other – not only in emergency situations but also in everyday, routine communication. This change will also affect the way we communicate in our professional lives.

Within medicine, particularly within its most specialized areas, access to domain experts is limited for many practical reasons. Access to medical data such as electronic medical records, patient images, and laboratory results, is also often limited by physical properties. In general, the main difficulty is that medical data is not available or accessible in a certain situation – either because it is physically not there, or because the expert is not where the data is, or because the data cannot be accessed without using time-consuming manual procedures [56].

In order to address some of these issues, much effort has been invested in making medical data "portable" by storing it in computers instead of on film plates and paper. This resulted initially in a plethora of storage formats that was as useless as paper used to be, in the situations were data was needed elsewhere. With an increased computer maturity and international attempts at defining medical standards (for terminology, medical record data structures, protocols for data exchange, etc) the situation is improving and Telemedicine is rapidly gaining favor over previously used procedures [3].

The user and the importance of utility and usability

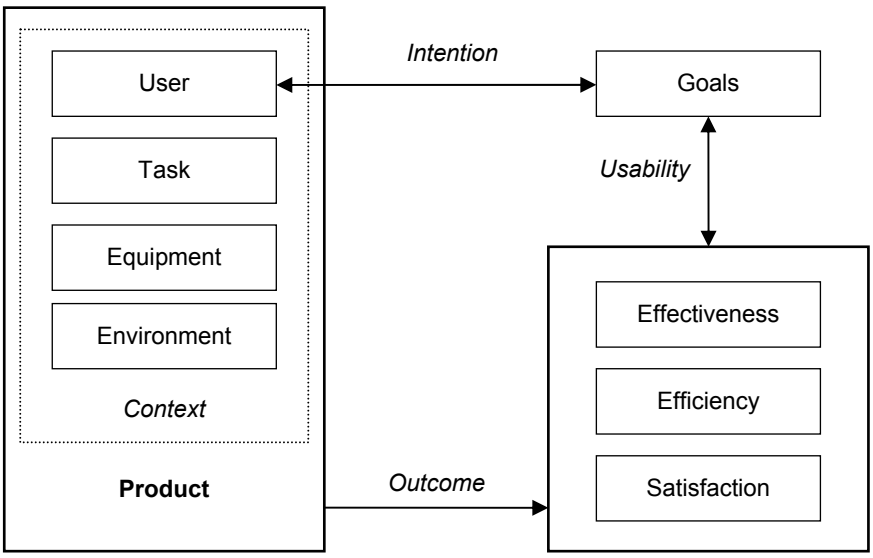

Figure 8. The ISO 9241 usability framework.

In health care, as in many other work situations where computerized information systems are used, the purpose of the work performed by the involved professionals is never to operate the computer. The computer is a tool that

will be accepted and used only as long as it efficiently supports the efforts to provide good health care for the patient.

This means, among other things, that the user interface for the information system must be designed to optimize the health care work activities as such and not simply to optimize the handling of the computer as a tool [55]. In *Figure 8* we see what the framework for measuring usability looks like.

Design

Design is the (early) stage in development process when one plans out in the mind the qualities and inherent features of a future artifact.

Sometimes the graphical user interface (GUI) is mistaken for being the design element of computer systems. The GUI is about visuals, but there are dynamic aspects as well as aspects of interaction that need to be taken into account. Using a traditional HCI point of view, the interchange of information between system and user is part of the design, in addition to the GUI [45]. But even this perspective may be a too limited. For instance, a nurse who hands over a paper-based patient record to another nurse can merely be seen as someone just passing over information. But below the surface, there are many possible interpretations of this act. One interpretation is that the responsibility for the patient has changed hands at the same time. From an information-processing perspective it is possible to just concentrate on the transferred information itself.

I subscribe to the view that design is a "reflective conversation" as defined by Schön [65]. He describes this relationship of *reflection in action* as the shift that happens when something interrupts the flow of the designer, who then shifts to a more conscious mode of analysis. This has been called a reflective conversation with materials. This perspective is closely related to my previous examples of the Scandinavian participatory approach and exploration of the design space using prototypes.

Design is everywhere today. Everything around us – products, environments, even services – is designed with different objectives in mind. There are academic degrees in design and more design awards than one can count. The year 2005 has been designated as the "Year of design" by the Swedish government. There are examples of when the notion of design has been taken too far, when policies and behavior have been called objects of design. It is questionable if everything can be labeled design in this way. Although design is hard to define, introducing some form of limitations will help us talk about good or bad design.

When building computer systems, we work from a perspective that shapes the questions that will be asked and the kinds of solutions that are sought. Winograd considers the perspective of design as vital [64], and argues for a language/action perspective:

> "*One useful way to identify a perspective is by its declaration of what people do. From a language/action perspective we say that People act through language. As a contrast, consider the more predominant perspective that People process information and make decisions. Of course everyone in an organization can be described as doing both, but there is a difference of focus*".

A number of different approaches have been used in an attempt to reach the goal of a working system. Each of the projects described in the papers that are discussed in this thesis, has a different solution. The various approaches are the result of our own broadened experience and to some extent they also reflect the varying needs of the projects.

Helios (Papers 1 & 2)

Project description

Helios was a research project within the framework of the AIM (Advanced Informatics in Medicine) program of the CEC (Commission of the European Communities). It was a large project with seven partners from different European countries.

In the Helios project, the overall undertaking was rather complex from a development point of view. The Engineering Environment contained many state of the art technologies, like natural language processing, images processing, automatic routing of communication and decision support. The goal was to build an advanced software tool that, in turn, was going to be used as a tool to construct complete medical applications. The project contained a number of teams of developers and general stakeholders.

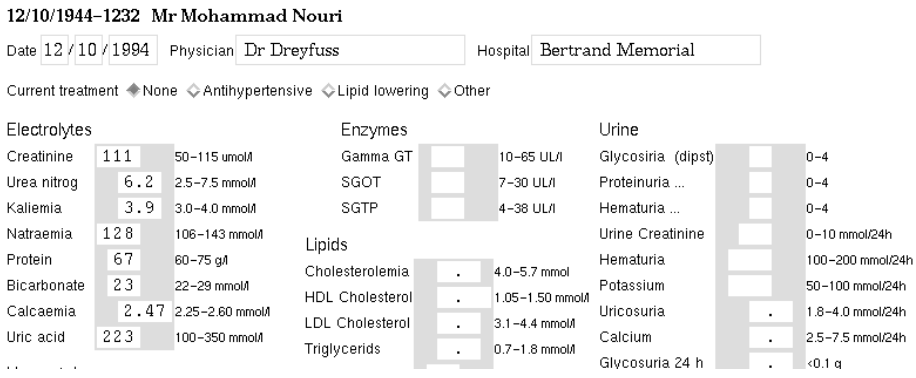

Figure 9. Design example from the Helios project showing part of an interface element for a clinical test form.

Solution

The solution suggested by our team was to produce static knowledge that could be agreed upon. This was also the research question: how could this knowledge be described and acquired? Ideally, the knowledge could be finalized once and for all. The solution would form a collection of guidelines and rules that would help a developer who was looking for design support. This knowledge base included basic cognitive facts on how humans work when interacting with computer systems. It is quite possible to find vital, static knowledge from cognitive psychology – i.e. we will always have a certain long-term memory capacity. Cognitive psychology can describe and help us understand how we function as humans.

This static knowledge was combined with knowledge about the target domain – in this case, medical systems. The knowledge about the domain came from previous experience, field studies and trials. All of this was then combined into a medical style guide – the idea being that it would be possible to make something similar to a cookbook, which a developer could use as a practical guide when developing a new medical system. Since some parts of the target domain are static we could even present design work that could be described as static. For example – the medical domain will always be patient-centered. Therefore we are able to produce a number of designs and implementations that will deal with frequent tasks that relate to patients. The patient card (with basic information about the patient: name, sex, age, etc.) could be designed before any specific application development had started. In *Figure 9* we see examples of static design that can serve as a basis for the development of additional forms.

This reasoning could be applied further and re-usable visual components (called widgets) could be developed to better suit the target domain. One example of such widgets is text entry fields in which the input can be verified before accepted as valid input. Another example is navigation controls that could hold and control many objects of a similar type (patient cards for example) [12].

In conclusion, the core idea was that as designers and HCI experts, we could describe the foundation for our own knowledge so that it could be used by others. Our explanations would take the form of prescriptions and advice. A lot of upfront design could be provided, as examples of functioning graphical user interface objects.

MEDICUS/CHILI (Papers 3 & 4)

Project description

CHILI is a general-purpose radiology workstation with teleradiology and telecardiology functions. CHILI is the successor of MEDICUS. Both appli-

cations were developed for the same purposes, by virtually the same group of people.

The context in the MEDICUS project was very different, as compared to the sizable Helios project. MEDICUS and CHILI both had a small and well-acquainted group, made up predominantly of software developers. Again, the goal was complex: a teleradiology station with the ability to handle on-line communication and application sharing. In this context, we wanted to take the same research standpoint used earlier, and examine if there were other, better forms of describing HCI knowledge. The purpose was to ensure that usability would be included in the development process.

The main use of the application would be to view images for interpretation and consultation purposes. Teleradiology is a means of electronically transmitting radiographic patient images and consultative text from one location to another. The final design proposal – a prototype – for application development can be seen in *Figure 10*.

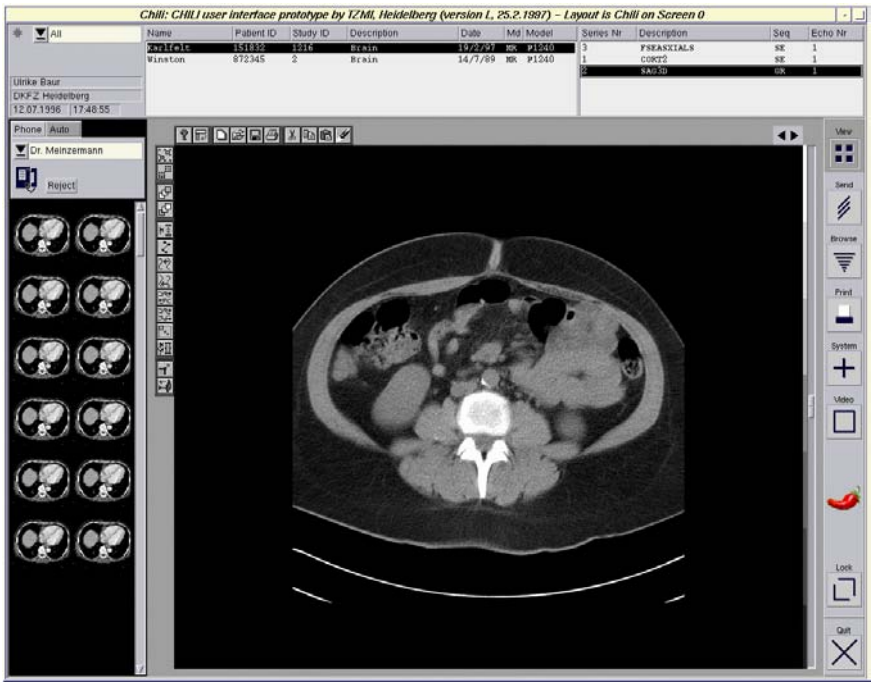

Figure 10. The original CHILI design proposal. This is the final design specification before implementation started.

Solution

The selected approach focused on elements that were the most important according to a particular ranking. We call these criteria, and the whole process is referred to as criteria-based design.

> *We assert that the product of user interface design should be not only the interface itself but also a rationale for why the interface is the way it is. We describe a representation for design based around a semi-formal notation which allows us explicitly to represent alternative design options and reasons for choosing among them.*
>
> MacLean on arguments behind the artifact. [40]

The selected strategy was partly influenced by the current methods that were in vogue: Object Modeling Technique [53] and Design Patterns [19]. OMT was a predecessor to Rational Unified process [27].

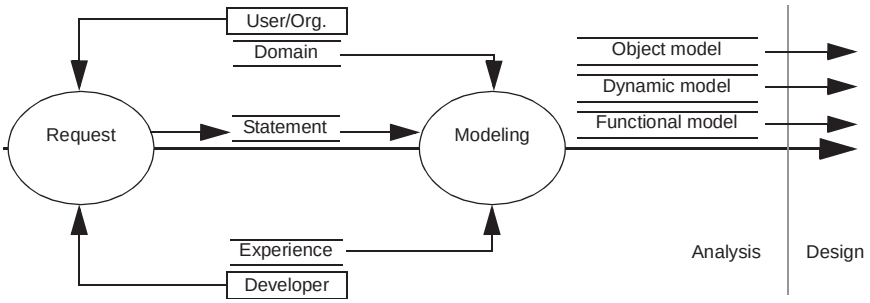

Figure 11. The OMT model overview.

We modified the OMT model (see *Figure 11*) to better suit our purposes, which were not primarily to construct the system, but rather to specify and design how the system should work. The outcome is a design proposal but it also includes the rationale for the proposed solution. The resulting model is seen in *Figure 12*.

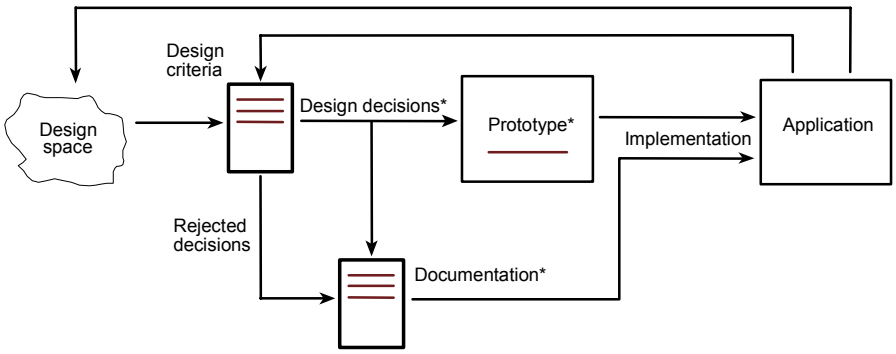

Figure 12. Our re-engineered OMT model that retains information about underlying design reasons as well as information about other attempted solutions that were rejected.

The model deals with how to manage conflicting options and how to preserve the knowledge about the selections that are made. The result is both a prototype and documentation. The resulting artifacts are marked with an (*) in *Figure 12*.

A short summary of the method is as follows:

- The process starts off with an initial set of design criteria. The whole team must agree on these criteria.
- During the design process, solutions are drawn from a very large set of possible solutions referred to as the design space.
- If a proposed solution is consistent with all other chosen solutions and it is supported by the design criteria, it is implemented in the prototype. At the same time, the solution is transferred to the design documentation
- If a solution is rejected, the proposed solution, plus the reasons why it was rejected are moved to the design documentation.
- The final prototype plus the design documentation serves as a basis for the development team as they start implementing the application.
- Formal and informal evaluations of the application both influence the original criteria and the design space.
- The process is repeated until all necessary functionality is incorporated in the design.

WeAidU (Paper 5)

Project description

WeAidU offers physicians automatic medical decision support, where they can obtain a second opinion in a matter of seconds. It is aimed at the cardiological market and uses AI and image analysis validated by medical ex-

perts. A correct diagnosis can, for example, help prevent heart attacks which are the leading cause of death in the world today.

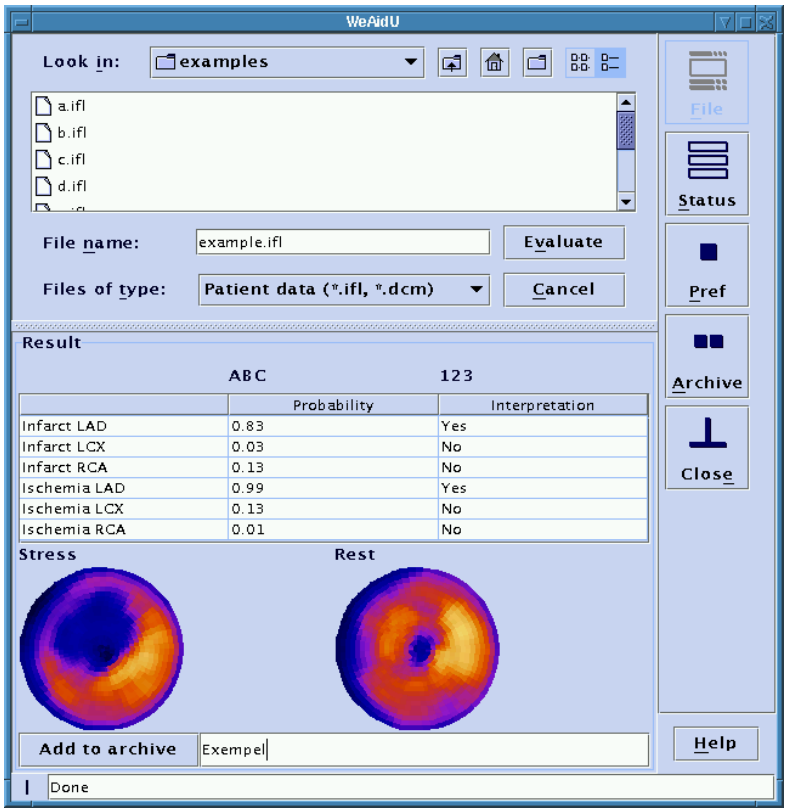

Figure 13. The first production version of the WeAidU application.

The aim of WeAidU was to develop an Internet-based intelligent system that could provide near instantaneous professional aid in clinical decision-making. The system involves artificial neural networks trained to interpret myocardial perfusion scintigrams. This makes it possible to enhance the effectiveness of clinical decision-making in the case of coronary artery disease. The system is designed to work on the Internet.

The following images show two generations of the same decision support. In *Figure 13* we see the first attempt. The second solution in *Figure 14* shows a more traditional solution with pull-down menus.

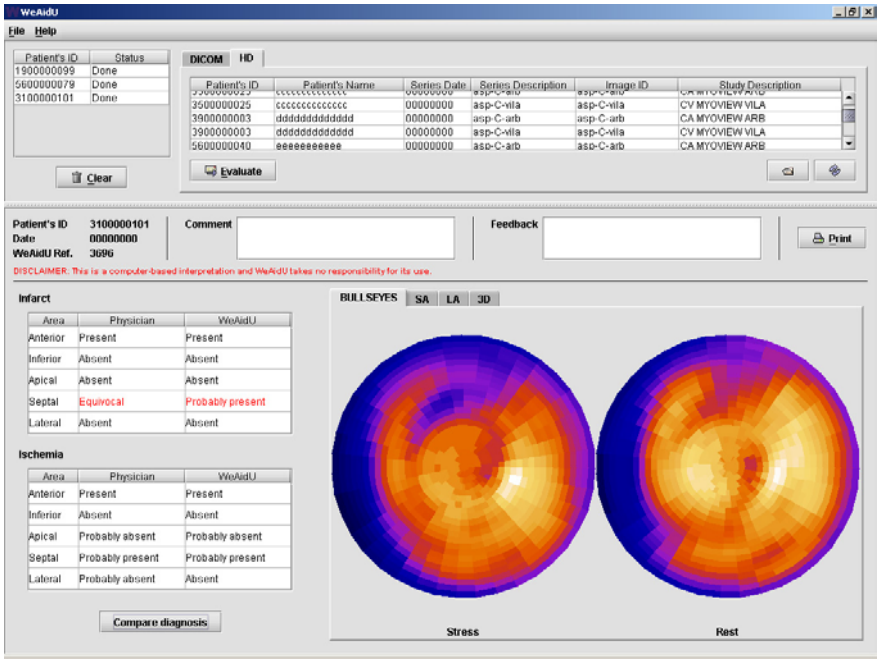

Figure 14. The current version of the WeAidU application.

Solution

The WeAidU approach is an exploratory strategy. It is based on the notion that initially we cannot accurately know what to develop and therefore we do care about requirements! The process is characterized by the spirit of "finding out the requirements as you go." The reason for this is that we actually want a transition from the status quo, and a preliminary study of today's situation may not bring us to an appropriate solution [15]. Because of this stance, the project's work methods meant targeting the most difficult issues at hand, on a day-to-day basis.

In this context, my focus and interest has been to observe and study how the development process is reflected in the final system. Since the way software is developed affects the outcome, it is of special interest to examine how usability is affected.

The method is an interactive as well as an iterative process, where the repeated cycle consists of the following steps:

- Abstraction
- Structure
- Representation
- Detail

Abstraction includes the study of the target domain, in order to form some kind of an elementary understanding of the domain. That general understanding is then categorized into a structure that is possible to manipulate. A structure can include the relationship between involved elements and users' interests. A transformation from the structure to a visual representation then follows. This contains the most important properties of the desired solution. In the early stages of development, the representation is often a prototype that is at some primal level of maturity. Afterwards, details are added to the visual representation [65].

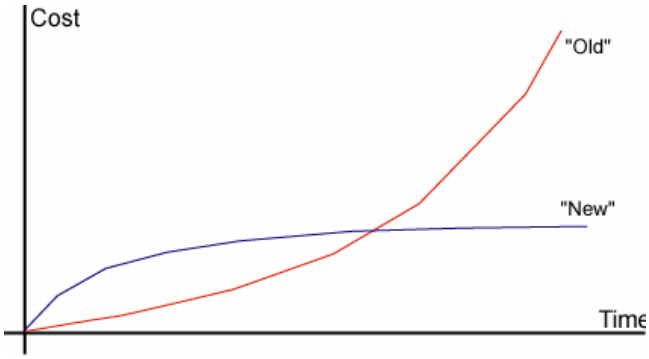

Figure 15. Old and new way of estimating cost of changes vs. time.

The technique described is similar to the ideas found in XP – eXtreme Programming [5]. One of the driving forces in XP is the insight that it is more efficient, both in the short run and the long run, to only solve the problems at hand, instead of solving all the potential problems that might arise. For a trained programmer or designer this can be very frustrating, since this insight conflicts somewhat with the "golden standard" of developing the best possible solution – in which all similar problems can be solved with a single elegant solution. The reason we can ignore a full study before starting development, can be explained by the old "truth" about estimation of cost vs. time (see *Figure 15*) which claims that "there is a small cost for early changes, and a large cost for late changes." This statement does not hold true for all forms of development, and not only because of improved tools and more powerful computers. If the whole development cycle is based upon change and streamlined to handle changes (early ones or later ones), then the result is a flattened cost curve.

Advances in hardware and more suitable programming tools are a help in this process but they are not the most important cause of change. Instead, it is the shift in methodology and a change in attitude towards how software ought to be produced, that can bring about this effect.

Conclusion

Based on the knowledge and experience in the target domain, my research shows that in order to succeed, we need to acquire a deeper understanding about what to develop. This understanding can be enhanced by including users in the development process, by producing many intermediary prototypes and by making progress in fine increments [22, 48]. The way that the computer support is introduced and brought into use also plays a vital role in determining the outcome.

We also need to define what a successful system is, since this is not solely based on measurable objectives – as put forth, for example, in the ISO usability definitions.

A good and usable system in medicine can be said to posses the following characteristics:

- Efficiency
- Flexibility with regard to local work practice
- Ability to minimize cognitive load
- Ability to support learning within the medical domain
- Easy to learn and use
- Suits the work process
- "Obvious" because is does the "right" thing
- Supports the care delivery process
- Is concerned with the human aspects of the medical profession

Although not a strict definition, this list of attributes will help us discuss and talk about the nature of our goals.

Embrace change

Not enough attention has been given to the dynamics of design – in other words, to things that change. From the very start we know that work content and practice will change over time. We also know that the profiles of the users will change, for example when more and more people become computer knowledgeable. It is also true that the computer systems themselves change over time, sometimes in the most fundamental ways.

The web browser can serve as an example of a changing application from recent years. At the start, there was a simple web browser that could only display plain web pages as text. However, after a period of time, it could show complex designs and plug-ins from other applications such as video.

As new technology entered the scene, the same design had to cope with the introduction of e-mail into the same application. Still later, a calendar function was fitted into the very same system.

Change is an on-going process in the life of any computer system. We cannot always control the direction of change, but nevertheless, we can control the effect that change has on the design. Learning to embrace change will help us increase our flexibility, eliminate stress and – in the end – in the midst of uncertainty, result in better solutions.

The idea is to make the computer system, including the entire development process, flexible enough that the cost of late changes is less than that of early changes.

This might sound confusing, but early changes, or requirements if you like, are not always the obvious cheap solution. Beck [5] explains this shift in modern systems development by explaining how we always say that we are going to write requirements and then implement the software according to the requirements – but in fact, we never do. The requirements always change, or for some reason are considered unclear, because the needs of the users are never precisely known from the start. When that occurs, the cost of retro-fitting the design is inevitably high.

In order to be able to create a situation where change is possible, one has to be well-prepared. The way software is produced has to be set up so that changes do not hinder any new development. This addresses the methods used, but more importantly, it addresses the skills and self-confidence of the developers. If the developers feel confident, then they are able to re-work the implementation and use the change to improve the internals as well as the resulting system. If the developers do not feel they have the necessary skills, the introduction of new requirements will become a burden that will complicate any further advancement.

Problem statement

Embracing the whole development cycle in detail is truly a giant task. Science can address well-formulated problems but not, as Schön puts it, "*messy, indeterminate situations*" like those we often face in a real-world project [54]. The main problem with developing a usable system needs to be defined accurately. A key question is what elements determine how the result is going to turn out. Still, a holistic approach would bring much more useful knowledge into the hands of those who need it. It is my sentiment that the whole is not always the sum of its parts. In this sense, a birds-eye view of the development process could bring about something more valuable and usable. From an industrial point of view, methods and development techniques for usable systems are regarded as the most important issue within HCI research [28]. The focus of the work then has to be directed more towards distributing knowledge instead of participating in the development process.

Strategy

The process of going from abstract requirements to a visual representation is a key target, as it involves many of the elements that we believe affect the usability of the finished product. Exactly how one should do this and what kind of methods could be used are still questions we should examine more closely. It is very likely though that this process must include a prototype-oriented approach [65] that steers the process.

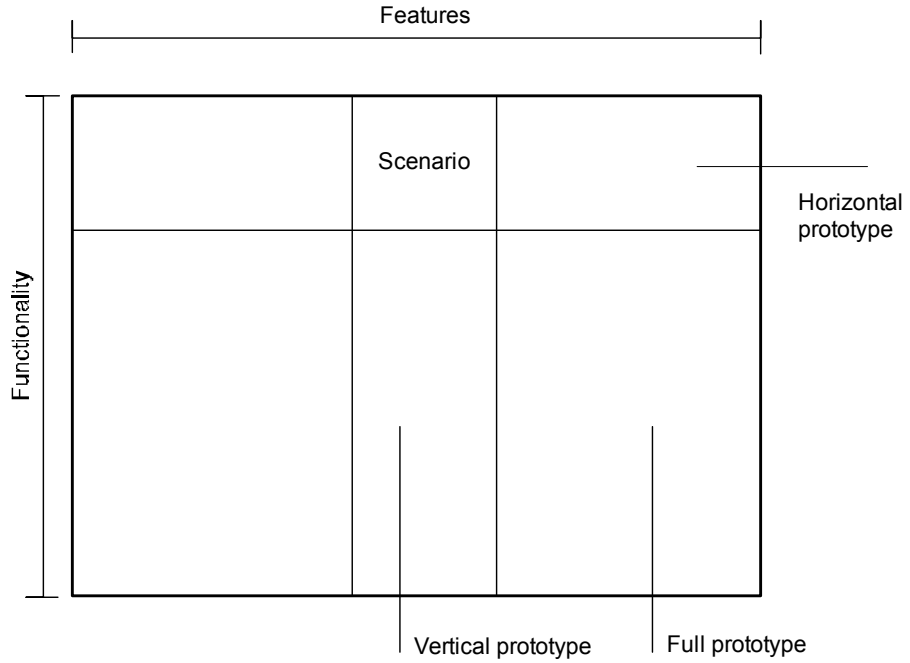

Figure 16. Relationship between different prototyping techniques.

By using prototypes, one is liberated from the many intrinsic problems of software development. In *Figure 16* we see some potential uses for prototypes. For example, prototypes can be used to capture a work situation in a certain scenario, or to mimic the full functionality of some chosen feature in a vertical prototype. Apart from its traditional areas of use (i.e. testing and benchmarking), a prototype can facilitate communication between stakeholders and serve as documentation of design decisions.

Figure 17. Bringing it all together…
Published with kind permission of the artist [7].

In this thesis, three ways of developing Telemedicine systems with good usability are described. The first approach is based on formalized knowledge in the form of a domain-specific style guide. The style guide contains a mix of objective HCI knowledge and heuristic guidelines. The second approach is based on the role of the user interface designer; it presents a method of designing the user interface in such a way that the knowledge and rationale behind the design is preserved throughout the entire development cycle. The third and latest effort is a heuristic method that redefines the approach to software development. Its standpoint is to utilize technical changes that software development has gone through during the last decade. These technical changes are used to modify the development process so that continual changes in the development process are possible.

Some conclusions now follow, derived from each of the attempted solutions:

Guidelines

Today, Style Guides and similar design tools (guidelines and standards) are very much in demand from industry. Some of the popularity is probably a result of the recent focus on quality assurance and the desire to develop systems according to pre-defined or well-established methods. Furthermore, guidelines have a high level of applicability (as opposed to standards) and can be used from the very start of a project [47].

Even though guidelines are based primarily on accepted practice and provide broad applicability, they are still the cause of some controversy. Indeed, some ask the question whether guidelines really help in the process of developing usable systems. The main argument is that since guidelines are described on a general level (e.g. "strive for consistency") they still need some kind of interpretation to be of use in a specific situation. But if an active interpretation is needed, then a developer lacking knowledge from the field of HCI, cannot make this interpretation correctly and the overall purpose of the guideline is not fulfilled.

> "*...consider the ability to write grammatical English. The skills that enable one to construct a grammatical sentence are the same skills necessary to recognize a grammatical sentence, and thus are the same skills necessary to determine if a grammatical mistake has been made. In short, the same knowledge that underlies the ability to produce correct judgment is also the knowledge that underlies the ability to recognize correct judgment. To lack the former is to be deficient in the latter*". [33]

Although there are limitations to the use and applicability of guidelines there are strong arguments for them. The question of whether the result is improved by the use of guidelines is perhaps irrelevant; if the only source of HCI knowledge within a project is gained from guidelines, then it is better than nothing at all.

Design methods

Improving usability measures by putting forward yet another method is possibly counter-productive; part of the explanation why systems have poor usability is the lack of simple and clear development strategies. Adding yet another method to the plethora of existing rules may not bring us closer to the goal of more usable systems or efficient development.

The design method (the modified OMT model) described in this thesis is a conceptual one that attempts to bridge the gap between the initial design and the final system. Having established all the correct requirements and achieving a good design is of little use if the final implementation does not take these results into consideration. The major benefit derived from the proposed design method is that it addresses this issue – so the purpose is not simply to introduce itself as yet another method. It is the preservation of both design and rationale that is important to sustain. Given the complexity of any user interface design, much can be gained if underlying design knowledge can be maintained throughout the project. My work has convinced me that if this kind of design decision can be made comprehensible to all members of the development group, then the final system will come a lot closer to achieving its originally intended design and purpose.

Development methods

What if new computer languages and development tools delivered the long promised, more efficient development cycles? What if the cost of implementing a feature on the first day of the project was the same as if it was implemented on the last day of a project? What if we were not restricted by hardware performance or software restrictions to using only the existing set of interface components? What if we were free to choose the solutions that best suited our needs instead of being limited to the existing solutions?

It is advantageous to make changes in the way software is developed, but many decisions in the software industry are based on tradition and proven practice. Possible changes are not always considered in a serious manner. We are not (yet) close to answering all the questions listed above, even if we are making progress. The latest effort described in this thesis is an attempt to practice the proposals put forward by the eXtreme Programming movement.

Embracing change is not easy. It is not uncomplicated to ignore the uncomfortable feeling of not knowing what to develop and simply defer unresolved questions to a later point in the process, when we might know more. Instead, a lot of effort is spent on developing functions that answer questions like "wouldn't it be nice to be able to do this?" This is sometimes called "creeping featuritis" – that is to say, the tendency for anything complicated to become even more complicated because of the fear of just keeping things as simple as possible. In this sense, the attempted solution that is presented has perhaps not been a successful one. However much progress has been made, primarily in terms of the way software is meant to function in hospital settings in the future.

Bringing about a change in the way software is developed is a difficult task. The traditional techniques have become so natural for us that it will take time to achieve change. While not included in the scope of this thesis, it is important to note that economic aspects play a vital part here. A mentality of sufficiency is good business, whereas a mentality of scarcity creates its own waste.

More is more, less is better

> *The two most important tools an architect has are the eraser in the drawing room and the sledge hammer on the construction site.*
>
> Frank Lloyd Wright

While a certain percentage of the population will always be "gadget nerds" who cannot get enough of functionality in a device, most people long for a TV recorder whose programming is intuitive, a wireless access point that is

easy to configure, or a cell phone whose instruction book has fewer than 100 pages.

In a recent survey, Microsoft found that most consumers use only 10% of the features offered in Microsoft Word. In other words, 90% of the features of this frequently used software get in the way of the features that people actually want. This violates all principles of good design.

The Google Company provides ways to search through a huge amount of information on the Internet. To do this, they use advanced search algorithms and some 100,000 computers (the exact number is kept secret) to serve the users. In contrast to this complexity, the current word count on the Google.com website is 27.

What this boils down to, is that there is a place for computer systems that solve the problems we want to be solved, without adding anything more to the problem. Uncomplicated as this conclusion may seem, it encompasses more than first meets the eye.

Contrary to one's first impression, there is no contradiction between a simple design and its utility. Simplicity hinges as much on cutting away non-essential features as on adding helpful ones. The tempting and obvious path is to add features to a design, since this growth process provides a sense of improvement. But we must be aware of what values we actually desire; in essence, *more* means a quantitative approach. *Less* could mean a qualitative enhancement. Anyone can make things bigger and more complex. It takes a touch of courage and a lot of skill to move in the opposite direction.

All evidence indicates that size and complexity are major factors when a software project fails. From the Standish Group's failure statistics we know that a stunning 98% of large and complex IT projects (those costing over $10 million) fall short of their goals [61].

Many benefits can be achieved by maintaining a view that promotes simplicity. For instance, a simple design always takes less time to finish than a complex one. So, as long as it works, always do the simplest thing possible.

> *The trouble with simple living is that, though it can be joyful, rich, and creative, it isn't simple.*
>
> Doris Janzen Longacre [38]

Summary in Swedish

Design i telemedicin – utveckling och konstruktion av användbara datorsystem.

Inom människa-datorinteraktion så är det ett övergripande mål att utveckla fungerande datorsystem som stöder användarna väl. För att klara av att nå ett sådant mål behöver vi kunskap och förmågor inom flera områden.

En utvecklingsprocess måste börja med att man först skaffar sig en (viss) förståelse för vad som ska utvecklas, och senare, funderar över hur man bäst designar och utvecklar systemet. En sådan syn kan uppfattas som ogenomtänkt, då den inte alls tar upp eller specificerar funktioner, det vill säga det som systemet i själva verket ska utföra eller beräkna. Dock, i praktiken finns det inget sådant korrekt mål, som kan uttryckas som en slags funktionell specifikation. Istället finns det en uppsjö vaga målsättningar – några som överlever hela projekttiden, andra inte. Det är i sammanhanget därför bekymmersamt att majoriteten av dagens utvecklingsmetoder bygger på att objektiva mål och kravspecifikationer tas fram, och att de initiala kraven används för att styra hela den fortsatta utvecklingsprocessen.

För forskare i människa-datorinteraktion är mängden krav som ställs på datorsystem en stor utmaning. Många av kraven innehåller inbördes konflikter och kan dessutom verka som varandras motsatser. Huvudmålet att utveckla användbara system handlar till stor del om att kunna hantera just denna komplexa situation och förstå hur man ska lösa konflikterna på bästa sätt.

Det finns många olika angreppssätt och metoder för att utveckla datorstöd. Denna avhandling utvecklar ett angreppssätt, baserat på vetskapen att faktorer och egenskaper hos datorsystemen som påverkar upplevd och uppmätt användbarhet också måste vara del av utvecklingsprocessen. Avhandlingen presenterar denna metod tillämpat på utveckling av datorstöd för sjukvård. Den föreslagna lösningen visar på betydelsen av att användare är delaktiga i en stegvis upprepande utveckling där vikt läggs vid utforskande av lösningar på lång sikt. De vanligare, relativt resultatinriktade systemutvecklingsmetoderna som väljer att se utvecklingsprocessen som design, utveckling och utvärdering är inte tillräckliga för att ge god användbarhet i sammanhanget.

Acknowledgements

Over the years there have been many people and organizations involved in my work.

First, I would like to thank my supervisor Bengt Sandblad.

I would also like to thank the colleagues in my department, whom I have always enjoyed working with. I would especially like to mention Bengt Göransson, Eva Olsson and Mats Johnson.

On many occasions I have been working on site with fellow researchers at the Medical and Biological Informatics department of the German Cancer Research Center in Heidelberg, Germany. There are many persons involved, but I particularly want to say thanks to a few of them: Prof. Dr. H.P. Meinzer, for all of the invited assemblies and longstanding cooperation, Dr. Uwe Engelmann, for rewarding travels around Europe and all our late night sessions, Andre Schröter, for all his technical expertise, ex-member Dr. Athanasios M. Demiris, for many different things, including squash games. It has always been enjoyable. Also many thanks to the whole CHILI team, both old and new members.

I also want to thank the WeAidU team, especially Lars Edenbrandt and Andreas Järund for interesting work with the start up of a new type of application.

I wish to thank the sponsors that made it all possible: Swedish National Board for Industrial and Technical Development (NUTEK) for my participation in WeAidU, Advanced Informatics in Medicine Program of the Commission of the European Communities for my participation in Helios and the European Commission/Information Society Technologies for my participation in CHILI and MELISA (Multiplatform e-Publishing for Leisure and Interactive Sports Advertising).

References

1. Alexander Christopher, Ishikawa Sara, Silverstein Murray. (1977). A Pattern Language: Towns, Buildings, Construction. Oxford University Press. ISBN: 0195019199.
2. Alexander Christopher. (1970). Notes on the Synthesis of Form. Harvard University Press. ISBN: 0674627512.
3. Ball M.J, Douglas J, Garets D. (1999). Strategies and Technologies for Healthcare Information: Theory Into Practice. Springer Verlag. ISBN: 0387984429.
4. Bashshur RL. (1995). On the definition and evaluation of telemedicine. Telemed J. Spring;1(1):19-30.
5. Beck K. (1999). Extreme Programming Explained: Embrace Change. Addison-Wesley Pub Co. ISBN: 0201616416.
6. Bemmel van J, Musen M. (1997). Handbook of Medical Informatics. Springer Verlag. ISBN: 3540633510.
7. Berglin J. (2002). Lagom Berglin : Jan Berglin Samlade Teckningar 1999-2002. Ordfront Galago AB, Sweden. ISBN 9189248392.
8. Berner S.E, Ball M.J. (1998). Clinical Decision Support Systems: Theory and Practice. Springer Verlag. ISBN: 0387985751.
9. Beyer H, Holtzblatt K. (1998). Contextual design: defining customer-centered systems, Morgan Kaufmann Publishers Inc., San Francisco, CA.
10. Bjerknes G, Ehn P, Kyng M. (1989). Computers and Democracy. A Scandinavian Challenge, Aldershot: Avebury, 250 s, ISBN:0-566-05476-0
11. Bødker, S. (1991). Through the Interface: A Human Acitivity Approach to User Interface Design., Hillsdale NJ, Larwence Erlbaum Associates, Publishers, ISBN 0-8058-0571-0.
12. Borälv E. (1994). New Interface Primitives - extending the OSF/Motif widget set. Report no. 47, CMD, Uppsala University.
13. Brooke, J. A. (1996). "Quick and dirty" usability scale. In Jordan, P., Thomas, B., and Weerdmeester, B. (Eds.), Usability Evaluation in Industry. UK: Taylor and Francis.
14. Cooper A, (1995). The myth of metaphor, http://www.cooper.com/articles/art_myth_of_metaphor.htm. Page visited 2004 Dec 21.
15. Cooper A. (1995). About Face: The Essentials of User Interface Design. IDG Books Worldwide Inc. ISBN: 1568843224.
16. Csikszentmihalyi M. (2000). Beyond Boredom and Anxiety: Experiencing Flow in Work and Play. Jossey Bass Wiley. ISBN: 0787951404.
17. Eckel B. (2000). Thinking in Java. Prentice Hall Computer Books. ISBN: 0130273635.
18. Ehn, P. (1989). Work-oriented design of computer artifacts, Hillsdale, NJ: Lawrence Erlbaum Associates, 496s, ISBN 91-86158-45-7
19. Gamma E, Helm R, Johnson R, Vlissides J. (1995). Design Patterns. Addison-Wesley Pub Co. ISBN: 0201633612.

20. Göransson, B. (2004). User-Centred Systems Design: Designing Usable Interactive Systems in Practice. Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology, ISSN 1104-232X ; 981.
21. Graham P. (2004). Hackers and Painters: Essays on the Art of Programming. O'Reilly UK. ISBN: 0596006624.
22. Heiskanen A, Newman M. (1997). Bridging the gap between information systems research and practice: the reflective practitioner as a researcher. International Conference on Information Systems. Proceedings of the eighteenth international conference on Information systems. Atlanta, Georgia, United States. pp. 121 - 132. ISBN:ICIS1997-X.
23. Hewett, Baecker, Card, Carey, Gasen, Mantei, Perlman, Strong and Verplank. (1996). ACM SIGCHI Curricula for Human-Computer Interaction, http://www.acm.org/sigchi/cdg/. Page visited 2004 Dec 21.
24. Holub A, (1999). Building user interfaces for object-oriented systems, http://www.javaworld.com/javaworld/jw-07-1999/jw-07-toolbox-p5.html. Page visited 2004 Dec 21.
25. Hopkins, D. (1985). A teacher's guide to classroom research. Philadelphia: Open University Press.
26. ISO/IS 9241-11. (1998). Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11: Guidance on usability. First edition 1998-03-15, ref. number ISO 9241–11:1998(E), International Organization for Standardization, Geneve.
27. Jacobson I. (1994). Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley Pub Co. ISBN: 0201544350.
28. Katzeff C, Svärd P.O. (1995). Användbarhet i praktiken. Stockholm, Sweden: Swedish Institute for Systems Development.
29. Kemmis S, McTaggart R. (1988). The Action Research Reader. Third edition. Deakin University Press, Victoria.
30. Kirakowski J, Porteous M, Corbett M. (1993). Software Usability Measurement Inventory questionnaire. SUMI End-user Handbook. Human Factors Research Group, Ireland. http://www.ucc.ie/hfrg/questionnaires/sumi/. Page visited 2004 Dec 21.
31. Kirakowski J. Questionnaires in Usability Engineering, A List of Frequently Asked Questions (3rd Ed.), Human Factors Research Group, Cork, Ireland. http://www.ucc.ie/hfrg/resources/qfaq1.html. Page visited 2004 Dec 21.
32. Kruchten P. (2003). Rational Unified Process: An Introduction. Addison-Wesley. ISBN 0321197704.
33. Kruger J, Dunning D. (1999). Unskilled and Unaware of It: How Difficulties in Recognizing One's Own Incompetence Lead to Inflated Self-Assessments. Journal of Personality and Social Psychology 77:6 (Dec. 1999), pp. 1121-1134. http://www.apa.org/journals/psp/psp7761121.html.
34. Laurel B. (1990). Art of Human-Computer Interface Design. Addison-Wesley Pub Co. ISBN: 0201517973.
35. Lewin K. (1946). Frontiers in Group Dynamics: II. Channels of Group Life; Social Planning and Action Research. Human Relations (1:2), pp. 143-153.
36. Lewin K. (1958). Resolving Social Conflicts, Harper.
37. Lewis R. J. (1995). Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use. International Journal of Human-Computer Interaction. v.7 n.1 p.57-78.
38. Longacre Janzen Doris. (1980). Living More with Less. Herald Press. ISBN: 0836119304.

39. MacLean A, McKerlie1 D. (1995). Design space analysis and use-representations. In Scenario-Based Design: Envisioning Work and Technology in System Development. Edited by J. M. Carroll. Wiley, New York.
40. MacLean A., Young R. M., Moran T. P. (1989). Design rationale: the argument behind the artifact. Proceedings of the SIGCHI conference on Human factors in computing systems: Wings for the mind, Volume 20 Issue SI.
41. Model Based Interface Development, http://smi-web.stanford.edu/projects/mecano/model-based.htm. Page visited 2004 Dec 21.
42. Nielsen J, Mack R.L. (1994). Usability Inspection Methods. John Wiley & Sons. ISBN: 0471018775.
43. Nielsen J. (1994). Usability Engineering. Morgan Kaufmann Publishers. ISBN: 0125184069.
44. Nielsen J. (2004). The Need for Web Design Standards, http://www.useit.com/alertbox/20040913.html. Page visited 2004 Dec 21.
45. Norman D. A. (1990). The Design of Everyday Things. Currency/Doubleday. ISBN: 0385267746.
46. Öhman Persson J. (2004). The Obvious & The Essential: Interpreting Software Development & Organizational Change. Uppsala Dissertations from the Faculty of Science and Technology, ISSN 1104-2516 ; 57.
47. Olsson, E. (1999). Providing design knowledge to system developers by domain-specific style guides. Licentiate thesis. IT/Human-Computer Interaction, Uppsala University.
48. Olsson, E. (2004). Designing Work Support Systems – For and With Skilled Users. Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology, ISSN 1104-232X ; 983.
49. Petersson H. (2003). On Information Quality in Primary Health Care Registries. Linköping studies in science and technology. Dissertations ; 805. ISBN 91-7373-612-0.
50. Preece J, Rogers Y, Sharp H, Benyon D. (1994). Human-Computer Interaction. Addison-Wesley Pub Co. ISBN: 0201627698.
51. Rapoport, R.N. (1970). Three Dilemmas in Action Research. Human Relations, (23:4), 1970, pp. 499-513.
52. Root W.R, Draper S. (1983). Questionnaires as a Software Evaluation Tool. Proceedings of ACM CHI'83 Conference on Human Factors in Computing Systems 1983. p.83-87.
53. Rumbaugh J, Blaha M, Premerlani W, Eddy F, Lorenson F. (1991). Object-Oriented Modeling and Design. Prentice Hall. ISBN: 0136298419.
54. Schön D. A. (1984). The Reflective Practitioner: How Professionals Think in Action. Basic Books. ISBN: 0465068782.
55. Shneiderman B. (1997). Designing the User Interface: Strategies for Effective Human-Computer Interaction. Addison-Wesley Pub Co. ISBN: 0201694972.
56. Shortliffe E.H, Wiederhold G, Perreault L, Fagan L. (2000). Medical Informatics: Computer Applications in Heath Care and Biomedicine. Springer Verlag. ISBN: 0387984720.
57. Specialtiding: IT i vården. (2004). Dagens Medicin och Computer Sweden. 17 November 2004.
58. Spinuzzi C. (2002). A Scandinavian Challenge, a US Response: Methodological Assumptions in Scandinavian and US Prototyping Approaches. ACM Special Interest Group for Design of Communications. Proceedings of the 20th annual international conference on Computer documentation. Toronto, Ontario, Canada. pp. 208 – 215. ISBN:1-58113-543-2

59. Standish Group (1995). The CHAOS report. Available at http://www.scs.carleton.ca/~beau/PM/Standish-Report.html, and http://www.standishgroup.com/sample_research/chaos_1994_1.php.
60. Statistiska centralbyrån/Statistics Sweden. (2004). Företagens användning av datorer och Internet 2004/Use of ICT in Swedish enterprises 2004. SCB, Enheten för forskning och informationsteknik, Stockholm. ISBN 91-618-1261-7.
61. The Standish Group. (2004). Unfinished Voyages. http://www.standishgroup.com/sample_research/unfinished_voyages_1.php. Page visited 2004 Dec 21.
62. Van Bemmel J., Gremy F, Zvarova J. (1985). Medical Decision Making. North-Holland. ISBN: 0444878408.
63. Vygotsky, L.S. (1980). Mind and society: The development of higher mental processes. Cambridge, MA: Harvard University Press.
64. Winograd T. (1987). A Language/Action Perspective on the Design of Cooperative Work. Human-Computer Interaction 3:1 (1987-88), 3-30.
65. Winograd T. (1996). Bringing Design to Software. Addison-Wesley Pub Co. ISBN: 0201854910.

Acta Universitatis Upsaliensis

Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology 5

Editor: The Dean of the Faculty of Science and Technology

A doctoral dissertation from the Faculty of Science and Technology, Uppsala University, is usually a summary of a number of papers. A few copies of the complete dissertation are kept at major Swedish research libraries, while the summary alone is distributed internationally through the series Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology. (Prior to January, 2005, the series was published under the title "Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology".)

Distribution: publications.uu.se
urn:nbn:se:uu:diva-4760

ACTA
UNIVERSITATIS
UPSALIENSIS
UPPSALA
2005