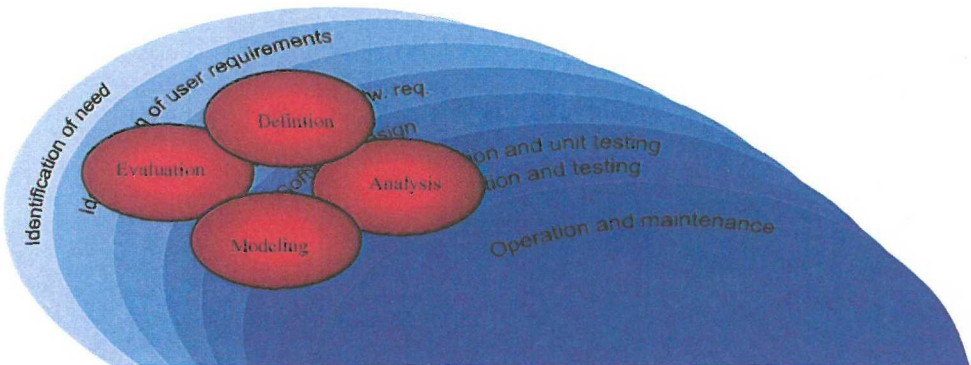


# DOCTORAL THESIS

## USABILITY WORK AND INDUSTRIAL SYSTEM DEVELOPMENT



by  
TOMMY NORDQVIST

*"Fly before you buy"*

Voss, 1993

Department of Human Work Sciences  
Division of Engineering Psychology

**Doctoral Thesis 1997:18**

# **Usability Work and Industrial System Development**

**by**

**Tommy Nordqvist**

Division of Engineering Psychology  
Department of Human Work Science  
Luleå University of Technology

## **Akademisk avhandling / Dissertation**

för avläggande av filosofie doktorsexamen i ämnet teknisk psykologi,  
som med vederbörligt tillstånd av tekniska fakultetsnämnden vid Luleå  
tekniska universitet kommer att offentligen försvaras,

i sal F341 vid Luleå tekniska universitet  
tisdag den 3 juni 1997 kl 13.00.

## **Handledare / Supervisors**

Prof Kjell Ohlsson, Luleå tekniska universitet  
Prof Jonas Löfgren, IDA Linköpings universitet

## **Fakultetsopponent / Faculty opponent**

Prof Martin Helander, IKP Linköpings universitet

## **Betygsnämnd / Examination board**

Prof Berndt Brehmer, FHS, Stockholm  
Docent Hans Marmolin, UI-design, Linköping  
Prof Houshang Shanavaz, Luleå tekniska universitet

## **Ordförande / Chairman**

Prof Kjell Ohlsson



## **ABSTRACT**

This dissertation is about usability work and industrial system development. The first part of the thesis utilizes present descriptions of the industrial system development process to illustrate activities performed.

Following this description is a definition of usability work, together with a number of methods and techniques deemed suitable for usability work. The methods and techniques are analyzed with respect to when they should be used, how to utilize them, outcome from utilization, need for supplementing methods, practical experiences and my own experiences from usability work and industrial system development.

Based on these descriptions and analyses a preliminary model for integration of usability work and industrial system development, together with some preliminary experiences, are presented. Use and outcome from the application of usability work methods are described for each system development activity.

Next is need for further and more comprehensive integration of usability work and industrial system development discussed. Also traditional computer support in system development is described followed by a brief discussion of its relevance for usability work. A number of simple computer-based tools aimed at studying the possibility to support development of computer systems, primarily user interface development, are then presented.

Finally, possible future work is discussed, mainly focused on computer supported usability work.

# CONTENTS

<b>PREFACE</b> .....	
<b>ACKNOWLEDGMENTS</b> .....	
<b>1. INTRODUCTION</b> .....	<b>1</b>
<b>2. INDUSTRIAL SYSTEM DEVELOPMENT: AN OVERVIEW</b> .....	<b>4</b>
2.1 ACTIVITIES IN INDUSTRIAL SYSTEM DEVELOPMENT.....	4
2.1.1 Identification of Need .....	5
2.1.2 Identification of User Requirements (Requirements Definition) .....	6
2.1.3 Overall Design of the System.....	12
2.1.4 Identification of Software Requirements (Software Requirements Analysis).....	12
2.1.5 Software Design.....	13
2.1.6 Implementation and Unit Testing .....	14
2.1.7 Integration and Testing .....	14
2.1.8 Operation and Maintenance.....	15
2.2 CONCLUSIONS .....	16
<b>3. USABILITY WORK</b> .....	<b>17</b>
3.1 A DEFINITION .....	17
3.2 EXAMPLES OF METHODS/TECHNIQUES FOR USABILITY WORK .....	18
3.2.1 Business Analysis (RASP).....	20
3.2.2 Task Analysis, (KAT).....	26
3.2.3 Usability Specification.....	33
3.2.4 Heuristic Evaluation.....	35
3.2.5 Cognitive Walkthrough.....	38
3.2.6 Use of Guidelines and Styleguides .....	41
3.2.7 Prototyping.....	44
3.2.8 Contextual Design .....	49
3.2.9 Use Testing (Usability Testing) .....	52
3.3 CONCLUSIONS .....	56
3.3.1 Business Analysis (RASP).....	57
3.3.2 Task Analysis (KAT) .....	60
3.3.3 Usability Specification.....	63
3.3.4 Heuristic Evaluation.....	66
3.3.5 Cognitive Walkthrough.....	69
3.3.6 Use of Guidelines and Styleguides .....	71
3.3.7 Prototyping.....	73
3.3.8 Contextual Design .....	76
3.3.9 Use Testing (Usability Testing): .....	78
<b>4. USABILITY WORK AND INDUSTRIAL SYSTEM DEVELOPMENT</b> .....	<b>81</b>
4.1 INTRODUCTION.....	81
4.2 INTEGRATION OF USABILITY WORK AND INDUSTRIAL SYSTEM DEVELOPMENT: A PRELIMINARY MODEL.....	83
4.3 AN EXAMPLE OF INTEGRATION OF USABILITY WORK AND INDUSTRIAL SYSTEM DEVELOPMENT .....	84
4.3.1 Identification of Need .....	84
4.3.2 Identification of User Requirements (Requirements Definition) .....	86
4.3.3 Overall Design of the System.....	88
4.3.4 Identification of Software Requirements (Software Requirements Analysis).....	89
4.3.5 Software Design.....	90
4.3.6 Implementation and Unit Testing .....	91
4.3.7 Integration and Testing .....	93
4.3.8 Operation and Maintenance.....	94

4.4 CONCLUSIONS .....	94
4.4.1 <i>The Model</i> .....	94
4.4.2 <i>Experiences</i> .....	95
<b>5. THE NEED FOR EXTENDED INTEGRATION OF USABILITY WORK AND INDUSTRIAL SYSTEM DEVELOPMENT .....</b>	<b>100</b>
5.1 INTRODUCTION.....	100
5.2 FURTHER INTEGRATION OF USABILITY WORK.....	101
5.2.1 <i>Role of Business Analysis</i> .....	103
5.2.2 <i>Role of Task Analysis</i> .....	104
5.2.3 <i>Role of Usability Specification</i> .....	105
5.2.4 <i>Roles of Heuristic Evaluation, Cognitive Walkthrough (Jogthrough), Use of Guidelines and Styleguides</i> .....	106
5.2.5 <i>Role of Prototyping</i> .....	106
5.2.6 <i>Role of Contextual Design</i> .....	107
5.2.7 <i>Role of Use Testing</i> .....	107
5.3 ADDITIONAL METHODS NEEDED .....	107
5.4 FURTHER DEVELOPMENT OF METHODS FOR USABILITY WORK .....	108
5.5 PRACTICAL EVALUATION .....	110
5.6 THE NEED FOR COMPUTER SUPPORT IN USABILITY WORK .....	111
<b>6. TRADITIONAL COMPUTER SUPPORT IN SYSTEM DEVELOPMENT .....</b>	<b>114</b>
6.1 INTRODUCTION.....	114
6.2 CASE SYSTEMS .....	114
6.2.1 <i>Analysis and Design Workbenches</i> .....	118
6.2.2 <i>Programming Workbenches</i> .....	119
6.2.3 <i>Testing Workbenches</i> .....	120
6.2.4 <i>Conclusions</i> .....	121
6.3 USER INTERFACE TOOLS .....	121
6.3.1 <i>Toolkits</i> .....	123
6.3.2 <i>Interface Builders</i> .....	123
6.3.3 <i>User Interface Management Systems</i> .....	123
6.3.4 <i>Application Frameworks</i> .....	126
6.3.5 <i>Conclusions</i> .....	126
6.4 TRADITIONAL COMPUTER SUPPORT IN SYSTEM DEVELOPMENT AND ITS RELEVANCE FOR USABILITY WORK .....	129
<b>7. SUMMARY OF THE STUDIES .....</b>	<b>132</b>
7.1 INTRODUCTION.....	132
7.2 STUDY 1: A KNOWLEDGE-BASED TOOL FOR USER INTERFACE EVALUATION AND ITS INTEGRATION IN A UIMS.....	132
7.2.1 <i>A Knowledge-Based Tool for Evaluation of User Interfaces, the KRI System</i> .....	132
7.2.2 <i>Integration with a UIMS</i> .....	134
7.2.3 <i>Conclusions</i> .....	136
7.3 STUDY 2: KNOWLEDGE-BASED EVALUATION AS DESIGN SUPPORT FOR GRAPHICAL USER INTERFACES .....	136
7.3.1 <i>The KRI/AG System</i> .....	137
7.3.2 <i>Conclusions</i> .....	141
7.4 STUDY 3: TUNE: A TOOL FOR USER INTERFACE EVALUATION .....	142
7.4.1 <i>TUNE</i> .....	143
7.4.2 <i>Conclusions</i> .....	147
7.5 STUDY 4: COMPUTER SUPPORT FOR USER REQUIREMENT EVALUATION IN SYSTEM DEVELOPMENT .....	148
7.5.1 <i>TURE</i> .....	148
7.5.2 <i>Conclusions</i> .....	151
7.6 CONCLUDING REMARKS.....	152

<b>8. FUTURE WORK .....</b>	<b>155</b>
8.1 EXECUTIVE SUMMARY OF WORK PERFORMED .....	156
8.2 CSUW IN INDUSTRIAL SYSTEM DEVELOPMENT.....	157
8.2.1 Introduction.....	157
8.2.2 Identification of Need.....	158
8.2.3 Identification of User Requirements ( <i>Requirements Definition</i> ) .....	160
8.2.4 Overall Design of the System.....	163
8.2.5 Identification of Software Requirements ( <i>Software Requirements Analysis</i> ).....	165
8.2.6 Software Design.....	166
8.2.7 Implementation and Unit Testing .....	169
8.2.8 Integration and Testing .....	170
8.2.9 Operation and Maintenance.....	170
8.3 SUMMARY .....	171
<b>REFERENCES.....</b>	<b>173</b>



## PREFACE

One of my advisors wrote in his thesis the following “Why do we go through graduate school and write dissertations?” He also gave an answer, “it is simply a pleasure.” Although I can agree with him, at least most of the time, my motive were more practical.

In my work as a usability consultant, customers are often focused on issues concerning user interface design. Most of the time they want advice on specific design proposals. They seldom ask for advice concerning the use of the system from a user, task or business perspective. Also, the interest in testing the usability of a developed or proposed system is minimal. This fact confused me, and I spent a lot of time wondering why the profession I represent is considered to be able to support industrial system development in such a restricted way. Therefore, I started investigate the literature and discuss with colleagues, to get an understanding of the process of industrial system development and to deepen my knowledge concerning possible contributions from the human-computer interaction discipline (HCI). The human-computer interaction discipline is defined as ‘the 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 Special Interest Group on Computer-Human Interaction Curriculum Development Group, cited in Hix, Finlay, Abowd, Beale, 1993, p. xi).

From this work, I realized that other professionals thought that much of the knowledge concerning user, task and business issues was to be found in other disciplines, not from usability experts and the HCI discipline. As I am convinced that HCI can contribute to industrial development of computer systems also with respect to these issues, especially if the computer systems



are going to support the work tasks performed by computer system users, I started to study how HCI could contribute.

One of the first objectives was to understand the industrial system development process. One reason for this was that I thought it was necessary to understand what to contribute to. Another reason was the necessity to understand when and how to contribute. Another goal was to investigate how HCI could contribute in a practical way. If not possible to show (and ideally proof) that HCI methods and knowledge can be of value, it is hard to convince managers and developers that it should be used. A third goal (from the beginning the main goal) was to develop some simple tools to illustrate the possibility to support industrial system development with HCI knowledge.

After I had started my work, I realized that my first and second goal was much more difficult to fulfill than I imagined from the beginning. The major reason for this was that industrial system development was more complex than just design, development and evaluation, as it is possible to conclude when reading some of the human-computer interaction literature. Another reason was that much of the HCI literature did not discuss the matter of how and when to contribute to the industrial system development process in enough detail.

These reasons resulted in that my work focused on the two first goals, and not as predicted on illustration of simple tools for bringing HCI knowledge into industrial system development. Another result easily noticed is that the thesis is not a monograph and not a collection of papers, it is something between.

Finally, the thesis is also written from a practitioners point of view. My interest is mainly in investigating how HCI methods and knowledge can contribute to industrial system development. Hopefully, this thesis also can remind us of the ideas of Dreyfus (1955, cited in Carroll & Rosson, 1985, p. 12-13).

## ACKNOWLEDGMENTS

Of course, there are many people and organizations who have contributed to this work. I will try to mention them all. Should I forgot someone, I give my apologies.

First of all, my advisors Kjell Ohlsson and Jonas Löwgren. Without their support and encouragement, I would never had finished, or started, this expedition into the academic world.

Leonard Adelman, my co-advisor, gave me valuable advice concerning both some of the articles presented, and many of the ideas expressed in the work.

My friend and colleague Per Asplund contributed to the work in many ways, especially in discussions of many of the ideas, and by implementing some of the tools mentioned. To be honest, some of the ideas presented here, are also his.

My colleagues Peter Ericsson and Johan Strand contributed both practically and theoretically to the work, for example, in implementing some of the tools and in discussions concerning ideas presented.

Staffan Löf has shared his ideas concerning development of computer systems with me, resulting in many interesting discussions.

Also my former colleagues at Enator Telub AB, Björn Bergström, Håkan Enqvist, Kaj Lethovaara contributed to this work.

All the people involved in some way and not mentioned earlier. Björn Peters, Sture Hägglund, Hans Marmolin, Göran Forslund, Ingemar Widegren, Ulrika Laurén, Karl-Erik Hedin, Lennart Ohlsson, Leif Larsson, Kent Lundberg, Kristian Sandahl, Joakim Karlsson, Nils-Erik Gustavsson, Stefan Cronholm.

Steve Andriole inspired much of my thinking by the ideas and experiences expressed in many of his books.

Jonathan Stubbs had the patience (and necessary knowledge) to read the thesis and to give me advice on how to improve my English.

The Swedish National Board for Industrial and Technical Development (NUTEK), Enator Telub AB, the Defence Material Administration (FMV), and my family provided necessary funding. Karl-Einar Sjödin, NUTEK, Per-Göran Nilsson, Enator Telub AB, Anders Mattson, FMV, and honorable wife, thank you. Also, National Defense Research Establishment (FOA) contributed funding to some of the studies, thank you Ingemar Widegren.

A special thanks to my family. My wife Marianne and our children Jenny and Johan supported me in many ways. Without this support, and their patience with my odd working hours, the work had never been done.

Finally, I want to thank my mother and father. Evidently, without them, this had never been possible.

*Värna, April 1997*

*Tommy Nordqvist*

## 1. INTRODUCTION

To develop computer systems is in many situations both difficult and time consuming, requiring expertise in many disciplines. The reason for this is the need to understand, for example, the work to be supported by a computer system, the technology or technologies to be utilized, and the process necessary to develop the computer system. These demands have resulted in a number of efforts directed to development of system development methods and supplementing activities. Different disciplines have also contributed, for example, system engineering, system analysis, software engineering and HCI. However, much work seem to have been performed purely within a discipline, with only minor interest in possible contributions from other disciplines. This is especially true if the work is studied from an industrial system development perspective. For example, work within HCI has mainly focused on user interface issues, different methods for identifying and analyzing user and task aspects with respect to human-computer interaction, different methods for making users more active participants in development, and alternative proposals concerning user centered system design. Also, a number of the efforts have related their work to oversimplified descriptions of the system development process. Resulting in that HCI expertise and methods are not utilized to its potential in industrial system development. This fact has resulted in that computer systems developed often do not fulfill user requirements, (see, for example, Christel & Kang, 1992; Lederer & Prasad, 1992; Raghavan, Zelesnik & Ford, 1994).

As a preliminary attempt to describe how methods and tools developed within HCI and other related disciplines can contribute, the present thesis exemplifies how different methods and tools can be integrated in industrial system development. Aiming at increased likelihood that the system developed being usable. Reason for this attempt is the issues mentioned



above, and also my experience that questions concerning the user and use of computer systems are not given priority, or are indeed forgotten in connection with development of computer systems, (see also Andriole, 1990; Näslund, 1994; Palmer, 1990).

Chapter 2 presents a superficial description of activities traditionally perceived as components in the industrial system development process. The basis for this description is different standards for system development, describing activities supposed, or required, to be performed in industrial system development. The chapter also presents some preliminary conclusions concerning handling of user requirements in connection with the system development process.

In Chapter 3, a definition of usability work is presented. From this definition, a small number of usability work methods are briefly described and analyzed. The methods presented are analyzed with respect to authors views on when they are supposed to be used, if any other methods has to be carried out as a supplement, practical experiences, and my own experiences from industrial system development and usability work.

In Chapter 4, my interpretation of when different methods for usability work shall be used is presented. Also, a preliminary model for integrating usability work and industrial system development is delineated. This model is followed by a brief example on how to integrate methods for usability work with different activities in the system development process, and how the results of the methods can be used in subsequent usability work and system development activities. The model is also elaborated further to illustrate how different methods for usability work can be integrated in the system development process. Finally, preliminary experiences from practical usability work in industrial system development are presented.

From these experiences, a simple analysis concerning the need for further integration of usability work in the system development process, is presented in Chapter 5. Here, need for additional methods, further development of methods presented, practical evaluation of methods, and need for computer support are discussed.

Chapter 6 gives a brief review of traditional computer support available in industrial system development, together with a simple analysis of its relevance for usability work. Mentioned in this chapter are CASE systems and User Interface tools.

Chapter 7 presents a summary of the four studies in the thesis, supplemented with a few concluding remarks. The studies focus on computer support for evaluation of user interfaces and user requirements fulfillment.

The thesis concludes with a discussion of possible need for computer supported usability work (CSUW), and presents some preliminary ideas concerning possible computer support in Chapter 8.

## 2. INDUSTRIAL SYSTEM DEVELOPMENT: AN OVERVIEW

### 2.1 Activities in Industrial System Development

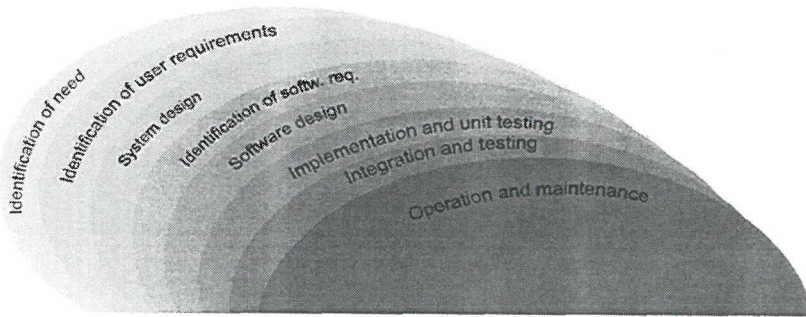
This section contains an overview of activities performed in industrial system development. From the system development standards IEEE P1233-1993 (1994), IEEE std 830-1993 (1994) and MILSTD 498 (1994), the following activities have been identified as parts of the system development process:

- identification of need,
- identification of user requirements (requirements definition),
- overall design of the system,
- identification of software requirements (software requirements analysis),
- software design,
- implementation and unit testing,
- integration and testing,
- operation and maintenance.

These activities can, to a greater or lesser extent, also be found in other system development literature (see, for example, Andersen, Kensing, Lundin, Mathiassen, Munk-Madsen, Rasbech & Sörgard, 1990; Andriole, 1990; Davis, 1990; 1993; Sage, 1992; Sage & Palmer, 1990; Sommerville, 1992; 1996; U.S. Department of Defense, 1985). System development standards are point of departure in describing industrial system development, as they are guidelines commonly used to describe necessary activities during system development.

The present description does not suppose any specific method of system development; it only identifies and briefly describes different activities performed in industrial system development. The role of project management

in successful system development is not considered in this description. In Figure 1 below, the system development process and system development activities are illustrated. It is important to note that in this model there is no rigid separation between different activities, often there are iterations within and between them (IEEE std 830-1993, 1994).



**Figure 1: Activities in industrial system development**

### *2.1.1 Identification of Need*

The system development process begins with a need or idea presented by a “user” (“user” is here used as a general term for all people contributing, for example, end-user, customer, business people) to a “developer” (“developer” or “system developer” is used here as a general term for all the people involved in development of a system. For a discussion of the different qualifications needed see, for example, Andriole, 1990). This can be an identified need for specific computer support or an idea for improvement of a business process. This idea or need is often expressed in general terms (see, for example, Sommerville, 1992). Usually it is necessary to help the user identify and specify actual needs or ideas from general ideas or needs.

### *2.1.2 Identification of User Requirements (Requirements Definition)*

Identification of need is followed by identification of user requirements (or requirements definition). The system developer, together with the user, identifies and defines the requirements for the future computer system. A detailed analysis is performed to establish exact needs. The goal of this activity is to identify all user requirements for the future computer system and to describe these requirements in a language understandable by both developers and users (IEEE P1233-1993, 1994).

According to IEEE P1233-1993 (1994), the identification of user requirements activity is iterative and consists of the following four sub-activities:

- “identify requirements from the customer, the environment, and the experience of the technical community,
  - build well-formed requirements,
  - organize the requirements into a SyRS (System Requirements Specification),
  - present the SyRS in various representations for different audiences, “
- (p. 17).

These sub-activities should not be seen as sequential. In most cases, there are iterations between them. Below, is a short description of the above mentioned sub-activities.

#### Identification of requirements.

With the needs or ideas identified in the identification of need activity as a basis, the requirements of the system to be developed are identified and defined. The purpose of this sub-activity is to identify every requirement,



check that each requirement is defined only once, and that no requirements are omitted.

According to IEEE P1233-1993 (1994), it is important that the process of identifying and defining requirements is managed to ensure the following:

- “the process is goal directed and aimed at the production of a set of requirements,
- the system boundaries are defined,
- all requirements are solicited, fairly evaluated, and documented,
- requirements are specified as capabilities and that qualifying conditions and bounding constraints are identified distinctly from capabilities,
- requirements are validated, or purged if invalid, from the requirements set,
- consideration is given to consistency when many individuals (‘authors’) may be contributing to the development of the requirements set,
- the developing requirements set is understood, at an appropriate level of detail, by all individuals participating in the process,” (p. 19).

There are a number of techniques for identification of requirements. Mentioned in IEEE P1233-1993 (1994) are:

- “structured workshops,
- brainstorming or problem-solving sessions,
- interviews,
- surveys/questionnaires,
- observation of work patterns (e.g., time and motion studies),
- observation of the system’s organizational and political environment (e.g., sociogram),

- technical documentation review,
- market analysis,
- competitive product assessment,
- reverse engineering,
- simulations,
- prototyping,
- benchmarking processes and products,” (p. 19).

#### Build well-formed requirements.

According to IEEE P1233-1993 (1994), this sub-activity is carried out by:

- “ensuring that each requirement is a necessary, short, definitive statement of need (capability, constraints),
  - defining the appropriate conditions (quantitative or qualitative measures) for each requirement. Avoid adjectives like resistant or industry wide,
  - avoiding the requirements pitfalls,
  - ensuring the readability of requirements. This entails:
    1. simple words/phrases/concepts,
    2. uniform arrangement and relationship,
    3. definition of unique words, symbols and notations,
    4. the use of language and symbology shall be grammatically correct,”
- (p. 20).

### Organization of requirements into a System Requirements Specification (SyRS).

In this sub-activity, the set of requirements is structured by relating the requirements to each other according to some method. According to IEEE P1233-1993 (1994), this activity is characterized by the following:

- “searching for patterns around which to group requirements,
- utilizing experience and judgment to account for appropriate technical approaches,
- utilizing creativity and intuition to generate alternative approaches and to prioritize requirements,
- defining the requirements properties,
- defining the requirements attributes,” (p. 21).

There are many strategies used to organize requirements into an orderly set. Most often utilized is gathering requirements into a service (capability) hierarchy, where general services are divided in subordinate requirements. Another method is to use network links (for example, hypertext), which show relations between requirements. According to IEEE P1233-1993 (1994), the following relations can be maintained in a system requirements specification:

- “hierarchical dependencies,
- events,
- information/ data,
- physical or abstract objects,
- functions,” (p. 21).

Presentation of requirements in the system requirement specification in different ways for different audiences.

In this sub-activity, the optimal way (alternatively, optimal ways) to communicate requirements to all individuals who need to understand, review, accept, or use the system requirement specification is identified. According to IEEE P1233-1993 (1994), one description is not enough in most instances, (see also Sommerville, 1992), because:

- “the customer and technical community usually have different cultures and languages; thus the same system requirements may have to be presented differently to the technical or customer communities,
- retrieval of specific information is difficult in some representations,
- representation of interactions can be difficult to do in some representations,
- relating information in one place to information in another place can be difficult in some representations,” (p. 21).

Therefore, it is important to present the system requirement specification in different ways, taking into consideration audience needs and background knowledge. For example, a general document including descriptive text and a selected set of high-level requirements, can be presented to customer staff responsible for project realization. For customer staff responsible for acceptance of the requirements, a more detailed document can be presented. For the design team, a document including low-level requirements can be presented.

Methods for describing requirements can, according to IEEE P1233-1993 (1994), be one or a combination of the following:

- “textual
  - paper,
  - electronic,
- model
  - physical,
  - symbolic,
  - graphical,
  - prototype,” (p. 22).

Definition of requirements usually continues after the system requirement specification is approved. In large and complex system development projects, it is likely that the first approved version of the system requirement specification has overlooked requirements, and/or misinterpreted needs or ideas. Knowledge concerning requirements also evolves in the process of developing the system. Therefore, it is necessary to iterate the process of requirements definition throughout the system development process. The aim being to correct deficiencies and/or supplement the system requirement specification with new requirements, and to enhance future computer system qualities (see, for example, Andriole, 1996; Sage, 1992; Workshop Proceedings, 1991, for a discussion of requirement definition and the development of requirements).



### *2.1.3 Overall Design of the System*

When the user requirements are defined and approved, system design follows. In overall system design, the focus is on issues relative to allocation of services (capabilities) to different parts of a system. Besides these general system design decisions, the system architecture is delineated (MILSTD 498, 1994; Sommerville, 1992). In other words, system parts, interfaces, and communication between parts, are identified and defined on a high level. Allocation of services to the computer system and to the user, may also be carried out (IEEE P1233-1993, 1994).

### *2.1.4 Identification of Software Requirements (Software Requirements Analysis)*

After system design, identification of software requirements (software requirements analysis) follows. User requirements are translated into a representation suitable for software development. This representation may be flow diagrams, object models, etc., (Andriole, 1990; Sommerville, 1992).

Main issues handled in identification of software requirements are, according to IEEE std 830-1993 (1994):

- “functionality. What is the software supposed to do?,
- external interfaces. How does the software interact with people, the system’s hardware, other hardware, and other software?,
- performance. What is the speed, availability, response time, recovery time of various software functions, etc.?,
- attributes. What are the portability, correctness, maintainability, security, etc., considerations?,

- design constraints imposed on an implementation. Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment(s), etc.?,” (p.4).

The outcome of this activity is a written Software Requirements Specification (SRS) used as the main reference when designing software.

### *2.1.5 Software Design*

In this activity, different functions are allocated to different software modules and software is structured in a convenient way (object-oriented design, functional design). According to MILSTD-498 (1994), focus is on definition and documentation of:

- general design decisions concerning modules,
- architectural design for each module (identification of software units in modules/components, interfaces and communication between units),
- detailed module/component design.

According to IEEE std 830-1993 (1994), the following is specified (see also Sommerville, 1992):

- partition of the software into modules,
- allocation of functions to modules,
- description of information or control flow between modules,
- design of data structures,
- design of algorithms.

### *2.1.6 Implementation and Unit Testing*

Software design is followed by implementation and unit testing. This activity consist mainly of implementation of different software units (modules), according to the software design, and testing of these units (MILSTD-498, 1994).

According to MILSTD-498 (1994), the following activities are performed during implementation and unit testing:

- software implementation (development and documentation of software in accordance with module design),
- preparation for unit testing, including development of test cases (input, expected results and evaluation criteria), test procedures and test data necessary to test the software in each software unit. Test cases shall encompass all aspects in module design,
- performance of unit tests (testing of each software module),
- revision and, if necessary, repeated testing (including necessary software revisions, retesting of the software and update of relevant software documentation),
- analysis and documentation of the results from unit testing (analysis of test results, documentation of tests and analysis results).

### *2.1.7 Integration and Testing*

Integration and testing (also called integration and system testing; see, for example, Sommerville, 1992), according to MILSTD-498 (1994), consist of software integration and testing of integrated software to check that integrated software works as specified. This process is iterated until all software is integrated and tested.

According to MILSTD-498 (1994), the following activities are performed in integration and testing:

- preparation for unit integration and testing (development of test cases, in terms of inputs, expected results and evaluation criteria, and development of test procedures and test data necessary for the integration and testing),
- performance of unit integration based on software design, and testing in accordance with test cases and test procedures,
- revision and, if necessary, repeated testing (including necessary software revisions, retesting the software, and updating relevant software documentation),
- analysis and documentation of unit integration and test results (analysis of unit integration, test results, documentation of tests and analysis results).

#### *2.1.8 Operation and Maintenance*

When the computer system is installed and acceptance tests are completed, the system is set in operation. Deficiencies recognized when using the computer system are resolved, and the computer system is further developed to meet business requirements. According to Sommerville (1992), this is often the most time-consuming activity. He divides maintenance into three types: perfective, adaptive and corrective maintenance. Perfective maintenance is maintenance necessary to develop the computer system further, without changing its functionality. Adaptive maintenance is maintenance necessary to adjust the computer system to changes in the environment. Corrective maintenance is maintenance necessary to correct deficiencies in the computer system that were not found during unit and system testing.

## 2.2 Conclusions

In the brief description of the process of system development presented above, user requirements are mentioned in the beginning of the process, in connection with identification of need, and identification of user requirements. They are not mentioned explicitly in subsequent activities. Accordingly, it is difficult to see how to guarantee that original user requirements are addressed throughout the system development process. As can be noticed in the description above, translations of user requirements are, for example, performed in conjunction with identification of software requirements and software design. This results in user requirements being expressed in a totally different (and more restricted) language when programmers are going to implement them. If original user requirements are not considered and presented in this situation, there is a risk that programmer interpretations of requirements are more influenced by their own experiences than by user requirements.

As can be noticed in this Chapter, the identification of need activity is described very superficially in the literature and little guidance is provided for the people that are to perform that activity. This activity seems to be superficially described also in the Software Engineering and Human Computer Interaction literature (see, for example, Dix, Finlay, Abowd & Beale, 1993; Nielsen 1993; Näslund 1994; Sommerville, 1992).

Also found in the literature surveyed, is that the original user requirements are divided in functionality and user interface (see sub-section 2.1.4). This separation is still present in software design, implementation and unit testing, and integration and testing. In the developed system then, functionality and the user interface is hopefully unified, rather than being iteratively integrated throughout all system development activities.



### 3. USABILITY WORK

#### 3.1 A definition

Usability work, as described here, comprises those activities that support development of usable computer systems. From the definition presented in Löwgren (1993), (other definitions can be found in Adler & Winograd, 1992; Card, Moran & Newell, 1983; Nielsen, 1993; Whiteside & Wixon, 1987; Woods & Roth, 1988):

“usability is a result of Relevance, Efficiency, Attitude and Learnability (REAL).

- the relevance of a system is how well it serves the users’ needs,
- the efficiency states how efficiently the users can carry out their tasks using the system,
- attitude is the users’ subjective feelings towards the system,
- the learnability of a system is how easy it is to learn for initial use and how well the users remember the skills over time," (p. 52),

usability work is defined as those activities that increase the likelihood that requirements related to relevance, efficiency, attitude and learnability are fulfilled in the computer system. Hence, usability work are those activities that contribute to:

- definition of the user requirements, in terms of services the system should deliver,
- definition of how, and to what extent, these services should support the user performance of work tasks,
- users perceiving the system to be good,
- easy learning and that knowledge is there for later access.

In this definition, development of the user interface and system services are not separated. Even if these activities can be separated (as is often the case in current system development practices), it is in my opinion necessary to treat them as supplementary perspectives both influencing computer system usability.

### **3.2 Examples of Methods/Techniques for Usability Work**

From this definition of usability work, there are many methods and techniques that may be referred to as parts of usability work (for the sake of simplicity, the term method is used to mean both methods and techniques). In the sub-sections of 3.2, examples of these methods are presented (The interested reader can find more methods and techniques in Dumas & Redish, 1994; Nielsen, 1993; Nielsen & Mack, 1994; Preece, Rogers, Sharp, Benyon, Holland & Carey, 1994, part 6). The basis for choosing the methods described, from the larger number of possible choices is that each fulfills one or more of following requirements:

- it should focus on some or more of the activities in the system development process described in Chapter 2,
- it should have been used in commercial system development,
- it should be possible to integrate (at least theoretically) with other methods,
- separate parts of the method should be possible to use individually and/or together with other methods,
- the methods should make continuous and iterative usability work practical throughout the system development process.

The reason for the first requirement is that the method should be possible to utilize in some of the system development activities. The motive for the

second requirement is the value of practical experience from utilizing the method. The third requirement originates from my experience that a single method alone is not enough to address all usability issues. It is necessary to see methods for usability work as tools in a tool box, to be used together in suitable combinations. For this to be possible, it is often necessary to integrate them in some way, to support each other. Requirement four has the same basis as requirement three. The reason it is presented as a separate requirement, is that for usability work to be efficiently accomplished, it is sometimes necessary to adapt methods to practical system development demands. This could mean performing less complex forms of usability work to be able to deliver timely results. The fifth requirement is perhaps the most important. If usability work is not performed continuously during the system development process, there is a risk that user and use perspectives “are lost” in one or more system development activities. From my experience (see also Andriole, 1990; Nielsen, 1993), it is evident that for successful system development to occur, these perspectives have to influence the entire system development process.

With these requirements as a basis for selection, one method for business analysis and one method for task analysis, together with the methods usability specification, heuristic evaluation, cognitive walkthrough, use of guidelines and styleguides, contextual design, prototyping and use testing (usability testing) are chosen. The reason why one method for business analysis (RASP), and one method for task analysis (KAT) have been chosen, is mainly that the purpose of this section is to illustrate how different usability work methods are performed and their outcome. It is enough to choose two out of all available methods to show applicability. However, this should not be seen as favoring these two methods over other methods for business and task analysis. The last statement is of course true also for other methods selected. Note, the methods selected here shall not be interpreted as discarding the methods mentioned in conjunction with description of

industrial system development (Chapter 2). Usability work methods can be considered as supplementary.

### *3.2.1 Business Analysis (RASP)*

RASP (Requirements Analysis and Specification methodology) is a method used for business analysis. (Other terms used for this kind of activity is Business Process Modeling, Hughes 1996; Business Process Reengineering, Hammer & Champy, 1993; Davenport, 1993). In RASP, people from business systematically map and describe present business, analyze possible needs for changes, and define future business (Telub AB & System Development Associates, 1990; Telub AB, 1995). (Other methods for business analysis that may alternatively be used in place of RASP can, for example, be found in Goldkuhl & Röstlinger, 1988; Willars, 1993a, b). Business analysis can result in and serve as information for:

- development of a computer system,
- development of organization,
- development of the staff in the organization.

The focus of RASP is mainly a functional perspective, where business is structured in a way that reflects the task oriented structure in the business studied. However, according to RASP, this perspective alone is not enough to describe and analyze a business. Therefore, RASP incorporates a human perspective (the business people) in parallel with the task oriented perspective. The functional perspective is considered at the model level and the human perspective is considered at the background level.

In RASP, modeling of business is important, therefore a generic business model has been developed. With the generic model as a framework, business is investigated, described, analyzed and developed.

The main components within the generic business model are:

- purpose,
- product/products,
- market/markets,
- resource/resources,
- supplier/suppliers,
- administrative instruments and reports,
- goal/goals,
- responsible people,
- business regulations.

Most important in the generic business model is the purpose of the business (the function). From this, concrete and abstract products are realized, which are aimed at one or more markets. The products can be a main product (the product realizing the purpose), by-products (one or more products not realizing the purpose, but useful for the business in some way), or other products (one or more products that must be handled by the business, but not useful for the business). To deliver the product or products, the business needs different resources. These resources are delivered by one or more suppliers. The resources can be resources attached to the product (resources that are part of the product), or resources attached to the process (resources used to develop the product), for example, resources in the form of knowledge, personnel, money or information. For management of business, different kinds of administrative instruments and reports are used. These



instruments and reports can be plans, policies, orders, messages, result reports etc. For the purpose of the business to be fulfilled, there must be business goals. Also people responsible for the business are important to identify. Business is also influenced by external business regulations, for example, instructions, guidelines and conventions, that the business has to follow.

A business analysis, according to RASP, consists mainly of the following activities:

- definition of the business analysis project,
- description of present business,
- need analysis,
- business development.

In the definition of project activity, the business analysis project is defined. Here, for example, project constraints, activities to focus on (description of present business, need analysis, business development), necessary members in the project, expected results, resources needed, and responsible for the project are defined.

In the description of present business activity, RASP experts use the RASP method to assist business staff in developing a description of the present business. First, the purpose of the business is identified and described. Then other aspects essential for the business, according to the RASP generic business model, are identified and described. If the business is complex, partitioning into sub-businesses, or sub-functions, is carried out. These different sub-businesses are then described according to the generic business model.

To describe present business, two supplementing modeling techniques are used. In RASP they are called functional modeling (or process modeling) and concept modeling (or object modeling). Using the functional modeling technique, present business, delivered products, resources needed, and the other components in the generic model are described. Using the concept modeling technique, different concepts used in the business are described.

In functional modeling, two supplementary ways to describe the business are used. The first is a written notation, supported by forms for description of functions. The second is a graphical notation, supported by function graphs. The written description is a complete description of the business studied. The graphical description is a summary description, focusing on the business relation to suppliers and markets. The graphical description is also used to identify sub-functions (sub-businesses) in the business. Sub-functions are described in writing and by function graphs. Sub-functions can be divided further, until an appropriate level of description has been reached. In concept modeling, a graphical object-oriented technique is used, where concepts are described through definition of their type instances and inherited parts. In Figure 2 and 3 below, is an example of a written and function graph business description.

Function: Company staff	
<b>Responsible:</b>	Chief of company
<b>Purpose:</b>	Managing the company
<b>Main product:</b>	Orders to platoons
<b>By-products:</b>	Information to other companies, information to commander
<b>Markets:</b>	The different platoons in the company, other companies, commander.
<b>Resources:</b>	Orders, information, staff, reports, communications.
<b>Suppliers:</b>	Commander, platoons in the company, other units.

Figure 2: A simple example of a written description of a function (business)

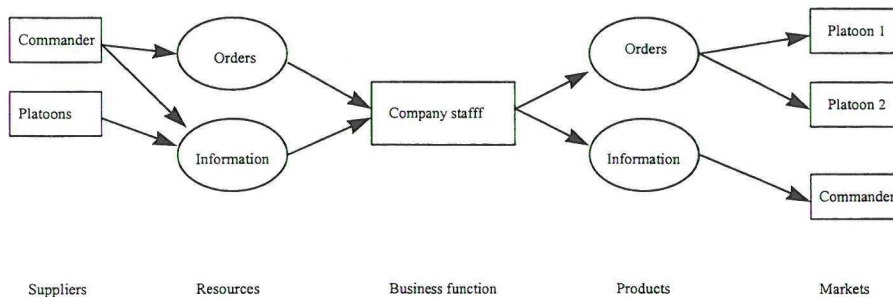


Figure 3: A simple example of a function graph

In need analysis, needs and wishes for business change are identified and described. Needs identified in conjunction with description of the present business are supplemented with a systematic analysis of possible change needs, carried out by business people and RASP experts. Basis for this analysis are models and documents generated during description of the

present business, plus ideas and suggestions voiced by business staff. The final task in need analysis is to prioritize change needs and to develop proposals on how to realize changes.

In business development, the future business is designed and described, (design alternatives may be included). Business development may take place on up to three levels; structuring of business, design of functions (business parts) and design of activities in business. Structuring of business is performed in order to change the structure of the present business to more efficiently fulfill the purpose of the business. For example, modification of the main function or identification of new resources or products. Design of functions is carried out when there is need to change the content in different sub-functions. Change of relations between sub-functions, deletion of sub-functions or creation of new sub-functions may be necessary. Design of activities is performed when it is necessary to define and describe how sub-functions are carried out in detail, and to describe the dynamics of this accomplishment. Design of activities can also include description of resources to be utilized, when they are to be utilized, who is to do what, and how different sub-functions will relate to each other.

To support the process of analyzing and describing present business, identify change needs, and develop proposals of future business, a computerized support system, MacRASP, may be used. This tool supports documentation, presentation, and consistency verification of function models and concept models.

The outcome from RASP is characterized by a detailed description of present business, a prioritization of change needs, and proposals for future business. This information can then be used in connection with development of a computer system and/or the organization and/or staff.

RASP is, according to the authors, especially suited for identification of possible needs for development of a computer system and in defining which part of a business will benefit from support. In this respect, RASP is of greatest benefit during the initial stage of system development, before decision about what to develop is made. The RASP analysis provides a basis upon which to base system development decisions.

### *3.2.2 Task Analysis, (KAT)*

Task analysis is a method for analyzing tasks (for example work tasks). The original purpose of task analysis was to support selection, training and education for different work tasks. Task analysis has been extended to include support of computer systems development, particularly user interface. The principal components of task analysis are the following three activities (Bodart, Hennebert, Leheureux, Provot & Vanderdonckt 1994; Diaper, 1989; Johnson, 1992; Johnson & Johnson, 1991):

- collection of information about task or tasks,
- analysis of the information,
- task modeling.

Johnson and Johnson (1990a, b; 1991) have developed a method for performing task analysis, KAT (Knowledge Analysis of Tasks). KAT is based on the TKS theory (Task Knowledge Structures, see, for example, Johnson, Johnson, Waddington & Shouls, 1988; Waddington & Johnson, 1989a, b for a description of TKS and its usage in task modeling) and focuses on identification and analysis of knowledge people possess about specific tasks. This knowledge is primarily utilized to support development of human-computer interaction.



According to KAT, task analysis consists of the following activities. Identification of the knowledge people possess about the task or tasks, analysis of this information, and task modeling.

Before task analysis can be carried out it is necessary to:

- define the purpose of the analysis,
- identify areas to be considered in the task analysis,
- identify tasks within each area,
- choose task or tasks to be analyzed,
- define information needed,
- identify where information can be gathered,
- decide how information shall be gathered,
- select which individuals to study.

According to KAT, task analysis is performed in the following way. First is identification of the knowledge people possess about the task or tasks. To identify this knowledge, goals and sub-goals necessary for task accomplishment are identified. Also, task procedures, objects used and actions taken during task performance are identified.

Suitable techniques for identifying this knowledge are, according to Johnson (1992):

- structured interviews and questionnaires,
- direct or indirect observation,
- concurrent or retrospective protocols,
- different experimental techniques, for example card sorting, rating scales, frequency counts.

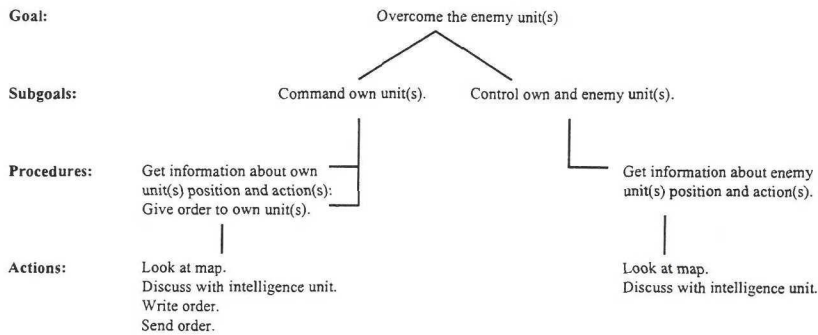
In Figure 4, below, a simple example of the results from above mentioned activities are depicted.

<b>Goal:</b>	Overcome the enemy unit(s).
<b>Subgoals:</b>	Command own unit(s). Control own and enemy unit(s).
<b>Procedures:</b>	Give order to own units. Get information about enemy unit(s) position and action(s). Get information about own unit(s) position and action(s).
<b>Actions:</b>	Look at map. Discuss with intelligence unit. Write order. Send order.

**Figure 4: A simple example of results from the identification of knowledge activity**

The information is next analyzed to identify representative, central and generic task components. The term “representative” meaning that some task parts are more representative or typical for the task or tasks. The term “central” meaning components necessary for task performance, without these components task completion is impossible. “Generic” task components are components common for a set of tasks within the task domain. Generic components are identified to minimize variation between tasks that are similar.

When task knowledge has been identified and analyzed, task modeling in TKS terms is performed. In Figure 5 below, a simplified example of a task model containing goals, sub-goals, procedures and actions is portrayed.



**Figure 5: A simplified example of a task model**

When modeling according to TKS, the model generated contains the following parts:

- a goal structure,
- a procedural sub-structure,
- a taxonomic sub-structure.

The goal structure is used to describe relationships between goal states. These relationships can be hierarchical relations or control relations. Hierarchical relationships describe how goals are divided in sub-goals. Control relationships describe how goals and sub-goals are related to each other in conjunction with task performance.

The procedure sub-structure is used to describe in detail, how a task is carried out. This sub-structure is directly related to the lowest level in the goal structure, and describes actions and objects, and the relations between them. Each procedure is defined by a pre-condition that defines the context that must exist before the procedure can be accomplished. The performance of a procedure results in an explicit outcome, a post-condition. These pre- and post-conditions are defined in the procedural sub-structure, and is a way to

describe the relationship between goal structure and procedural sub-structure.

The taxonomic sub-structure is used to describe hierarchical relations between objects in terms of their categorical affiliation. In addition, features and attributes of each object are described. Examples of features and attributes are object relationship to superordinate and subordinate categories, representativeness and centrality to task context.

According to TKS theory, it is possible to model existing tasks, as well as new or changed tasks in conjunction with their design.

According to Johnson and Johnson (1991, see also Johnson & Johnson, 1989), task analysis can support development of computer systems during the following activities:

“At the feasibility/initial planning stage:

- identifying and documenting any new functions/new tasks the computer may support,
- identify potential functionality of the system from user perspective,
- identify user population and characteristics of that population,
- identify characteristics of interface to be developed,
- allocation of function between user and system,
- assess scope/degree of larger-scale TA to be undertaken later in development lifecycle.

At the requirement/analysis stage:

- identify and document user/UI requirements comprising details about;
- hierarchical structure of tasks (goals and sub goals),
- how users achieve goals and sub goals,
- listing and ordering of undertaking task procedures,
- frequency with which particular procedures were carried out by users,
- reasons why and circumstances under which one procedure was used in preference to another,
- inputs and outputs from each procedure,
- events, data used, actions, objects,
- standard set of properties relating to objects and actions, e.g., frequency, time taken, etc.,
- expectations the user entertains about the system after user has carried out an operation,
- division between user and system.

At the Design stage/User Interface Development/Dialogue design:

- provide initial input to guide dialogue and screen design, comprising;
- details of what users expect to have available to them at any one time,
- the structure and sequence of their usage of system facilities,
- the names and form of representation to be given to screen-presented objects and events,
- information that should be available in given contexts, (i.e. design of screens),
- structure between contexts, (i.e., mapping between screens),



- how much to put on the screen at once with reference to number of commands,
- what information should go on screens and the grouping of that information,
- what commands are needed to support user operations and what those commands will be,
- user testing.

At the prototyping stage:

- guide initial format and presentation of prototype by indicating how the screens should look,
- identify data that has to be displayed,
- identify operations and sequencing of procedures,
- ensure dialogue specification is represented in a format that can be understood and verified with end users and to carry this out.

At the validation stage:

- user testing.

At the update and maintenance stage:

- identifying, documenting and cataloguing user problems," (pp. 14-15).

### 3.2.3 Usability Specification

Usability specifications are precise and testable performance measures of a user's planned performance of specific tasks using a computer system (Carroll & Rosson, 1985, see also Chapanis & Budurka, 1990). This method satisfies the need to specify usability goals that the system to be developed can be evaluated against. A usability specification has, according to Whiteside, Bennett, and Holtzblatt (1988), two purposes:

- clearly express how the usability of a system should be defined,
- function as a measure of how well, and to what extent, a computer system has fulfilled usability requirements.

The starting point in developing a usability specification for a computer system, is identification of functional goals and usability goals. Through an analysis of these goals, specific usability requirements are outlined that describe the user task to be supported by the computer system, in what respect the computer system shall support the task, and to what extent the computer system shall support the task.

According to Carroll and Rosson (1985), usability requirements can be partitioned into subskills, individual skills needed for successful performance of a certain task. For example, a subskill needed in creation of a document could be to understand commands. With these subskills as a basis, it is possible to evaluate components of the system under development for usability. For example, menus, dialog boxes and help texts. In Figure 6 below, a simple example of a usability specification is illustrated (based on Whiteside et al., 1988).

System attribute	Task measured	Measuring method	Minimal value	Planned value	Optimal value	Actual value	Comments
Map display	Find specific unit	Time (sec.)	3 sec.	2 sec.	1,5 sec.	5 sec.	
Map display	Look at info. about unit.	Time (sec.)	3 sec.	2 sec.	1 sec.	20 sec. - 3 min.	
Map display	Understand unit info.	Interviews	90% understood.	100%		85%	
Text area	Write order	Time (sec.)	3 min.	2 min.	1 min.	4 min.	

**Figure 6: Illustration of a simple usability specification example**

In the system attribute column, the system property in focus for the testing is listed. Task measured, is the specific task to be performed to evaluate the usability of the system attribute. Measuring method, is the method used to evaluate that the attribute fulfills the usability goal. Minimal value, is the minimal acceptable value for this specific attribute. This value indicates the usability baseline for the attribute. Important to note is that, according to Whiteside et al., (1988), the values of all attributes should reach at least this level for the total system to be regarded as usable. Planned value, is the level defined as desirable, and the level planned to be reached. Optimal value, is the level defined possible to reach for this attribute. This value can serve as a goal for future versions of the system. Actual value, is the value measured in connection with actual system use or manual work. The comments column can, for example, be used to reference other usability specifications, where attributes may be described in greater detail.

Usability specification is, according to the authors, used both as a way to create a common understanding among developers about usability requirements, and to define testable requirements against which a computer system may be evaluated. According to Carroll and Rosson (1985), the

usability specification should be developed in parallel, and integrated, with the functional specification (Software Requirements Specification).

Carroll and Rosson (1985) point to the importance of representative users and tasks when developing usability specifications. They also advocate the use of representative users and tasks when testing usability according to the usability specification.

A common way to evaluate if a computer system fulfills defined usability requirements is, according to Whiteside et al., (1988), to arrange an experimental situation, where users are requested to solve standardized tasks in a standardized situation. This approach makes aggregation of test data possible, for example, time used to solve a specific task for all users participating in the "experiment." Data can then be compared for different versions of the system. Carroll and Rosson (1985) advocate an informal and qualitative test situation, where focus is to elicit as much information as possible from each person involved with the computer system under development.

### *3.2.4 Heuristic Evaluation*

Heuristic evaluation is a systematic inspection method, developed by Nielsen and Molich (1990), to identify usability problems in a user interface. Development of this method is based on their and other's experiences that collections of guidelines for design of user interfaces (see, for example, Brown, 1988; Smith & Mosier, 1986) are difficult for developers to use (De Souza & Bevan, 1990; Tetzlaff & Schwartz, 1991; Thovtrup & Nielsen, 1991). With this kind of documents as a basis, they identified a number of general rules of thumb (usability principles) and a method for inspecting user interfaces using the rules.

The goal of heuristic evaluation is, according to Nielsen (1993), to discover usability problems early in the development process, making adjustment of the user interface in succeeding iterations possible. Heuristic evaluation shall be used during development of a user interface, and not as a method to review the usability of a already developed user interface.

Heuristic evaluation can, according to the authors, be used on early prototypes as well as implemented user interfaces. This causes the method to be suitable throughout the (software) development process, from early sketches to “completed” user interfaces.

According to Nielsen (1993; 1994), heuristic evaluation is carried out as follows. With the usability principles (see below) as a basis, the user interface is inspected by three to five experienced usability experts for possible usability problems. The usability experts perform inspection individually. When all inspections are accomplished, the evaluators analyze and compile generated comments collectively. In performing a heuristic evaluation an evaluator inspects a user interface several times. User interface elements are evaluated against usability principles and comments are recorded. If an evaluator discovers other possible usability problems, these are also documented, irrespective if there is a usability principle defined.

The evaluator decides how to perform the inspection, but the recommendation is to inspect the user interface at least twice. The first time to get an understanding of the user interface and the purpose with the system. The second time, focusing on every user interface element with the usability principles in mind.



Below is a brief description of the usability principles as defined by Nielsen and Molich (1990). A more thorough description of these principles is contained in the original document and also Nielsen (1993; 1994).

- Design the dialogue to be easy and natural. The dialogue should present only the information necessary for the user. All information should be presented in a way that is in accordance with the task to be performed by the user.
- Speak the users' language. The dialogue should be consistent with user knowledge and experience and should not be expressed in system oriented terms.
- Minimize user memory load. The user should not be required to remember information from one dialogue to another.
- Be consistent. Users should not be uncertain about meanings of terms, situations and actions.
- Give feedback. The computer system should always inform the user about what is occurring.
- Provide explicit exits. Give the user opportunity to exit states that are unwanted.
- Provide shortcuts. Give experienced users access to accelerators, to work more efficiently with the computer system.
- Design informative error messages. Error messages should be expressed in a way natural to the user. Give information about what is wrong and how the problem can be solved.
- Prevent error situations. Design the computer system so that potential error situations are avoided as much as possible.

- Help and documentation. Design help and documentation so that information is easy to find and understand. Information should also be presented in a way that reflects the task to be performed by the user.

The outcome from a heuristic evaluation is a list of potential user interface usability problems. Usability problems identified are closely related to usability principles utilized (Lewis & Rieman, 1993). This means there is a risk that some usability problems are missed. This risk can partly be taken care of by supplementing heuristic evaluation with some other evaluation method. (See, for example, Jeffries, Miller, Wharton, and Uydea, 1991, for a discussion of the advantages and disadvantages of different evaluation methods, with respect to usability problems identified).

### 3.2.5 *Cognitive Walkthrough*

Cognitive walkthrough (Polson, Lewis, Rieman, and Wharton, 1992; Lewis & Rieman, 1993; Wharton, Rieman, Lewis & Polson, 1994) is a method used to identify potential usability problems by imagining user intentions and actions the first time they use a computer system.

Cognitive walkthrough is carried out in the following manner. Use a prototype or a detailed design description of a user interface, plus knowledge of the user characteristics, as the basis of an evaluation. Choose one or more work tasks the future computer system will support. With these work tasks as a basis, attempt to tell a believable story about every action a user has to take to carry out a task. The story is believable if every user action is motivated by their knowledge, or prompts/feedback from the computer system. If it is not possible to tell a believable story about a user action, a probable usability problem has been recognized.

Deficiencies in user interface specifications can also be detected with cognitive walkthrough. Utilizing cognitive walkthrough, specifications can be inspected and potential deficiencies such as forgotten specification, be identified. A concrete example of what potential deficiencies can be identified is when relevant feedback is missing as when, "There is nothing specified, that informs a user the computer system is processing an input."

To perform cognitive walkthrough the following information is needed, according to Lewis and Rieman (1993):

- a user interface specification, a prototype or a completed user interface,
- a task description,
- a comprehensive description of actions required to perform a task when using the computer system,
- description of future users and of their knowledge regarding the task to be performed.

The main concept used in cognitive walkthrough is to attempt to tell a believable story about why a user chooses to execute each action. Then to critically review the story to ensure that it is believable. According to Lewis and Rieman (1993), the following four questions are important in story analysis:

- "will users be trying to produce whatever effect the action has?,
- will users see the control (button, menu, switch, etc.,) for the action?,
- once users find the control, will they recognize that it produces the effect they want?,
- after the action is taken, will users understand the feedback they get, so they can go to the next action with confidence?," (Chapter 4.1.3).

The first question relates to user intentions. Often, users have no intention to do what the developer thinks. The second question is related to the likelihood that users see controls at all. It is not unusual for controls to be hidden, so as to not damage a “beautiful” design. The third question is related to users’ possibility to identify the correct control. Even if users want to perform an action, and a control is possible to identify, there is no guarantee that they will understand it is the correct control. Note that identification and understanding are dependent upon each other. Users may not understand which action is correct, but a control that is easy to detect and understand helps them to determine what has to be done. The fourth question relates to feedback after an action is performed. Often, even the simplest action needs some form of feedback to inform users that the computer system has “understood” the action and that it has resulted in some form of processing.

According to Lewis and Rieman (1993), an evaluator can identify many different kinds of usability problems with the help of cognitive walkthrough:

- erroneous or defective assumptions about users’ intentions with the computer system,
- identification of controls (commands, switches etc.) that are obvious to the designer, but “hidden” for users,
- identification of possible difficulties to understand labels and prompts,
- identification of defective feedback, resulting in, for example, further user actions despite performance of a correct action the first time.

The potential problems found through a cognitive walkthrough are often simply resolved, since problems identified point to actions such as:

- change the user interface to be in accordance with user intentions,
- change the presentation of controls so they are easy to locate,

- change control design so that users understand their purpose,
- change label design so that users understand their meaning.

According to the authors, cognitive walkthrough can be used in the detailed design of a user interface, to evaluate a prototype, or an already developed user interface.

### *3.2.6 Use of Guidelines and Styleguides*

Guidelines and styleguides are recommendations and rules that are built on practical experience and research within the HCI (Human Computer Interaction) area. Guidelines are general recommendations and advice concerning design of user interfaces. Styleguides are specific rules (usually proprietary) for the appearance and, in some cases, also the behavior of a user interface for a specific implementation platform.

Examples of documents containing guidelines are:

- *Guidelines for Designing User Interface Software* (Smith & Mosier, 1986).
- *Principles and Guidelines in Software User Interface Design* (Mayhew, 1992).

The first document contains 944 guidelines, which deal with recommendations and advice concerning different ways of interaction, for example, menus, command language, forms, but also the design of help and feedback. The second document contains 288 guidelines concerning ways of interaction, such as menus, forms and direct manipulation.



Examples of styleguide documents are:

- *The Windows Interface: An Application Design Guide* (Microsoft, 1993).
- *OSF/Motif Styleguide* (Open Software Foundation, 1993).
- *Human Interface Guidelines: The Apple Desktop Interface* (Apple Computer, 1992).

There is also company and project specific guideline and styleguide collections. These often contain both guidelines and styleguides adapted to a specific company or project. Examples of this kind of collections can be found in Flygvapnet (1993), Defense Information Systems Agency (1994), Fernandez (1992) and Goddard Space Flight Center (1992).

To exemplify what guidelines are, two guidelines from *Guidelines for Designing User Interface Software* (Smith & Mosier, 1986) are presented:

- “Provide maps to display geographic data, i.e., direction and distance relations among physical locations,” (p. 163).
- “Allow users to select transactions; computer processing constraints should **not** dictate sequence control,” (p. 271).

To exemplify what styleguides are, a styleguide from *The Windows Interface: An Application Design Guide* (Microsoft, 1993) is presented:

- “The **Help** menu should contain components that provide user help facilities. The components in the **Help** menu usually bring up a DialogBox with the help information. Every application should have a **Help** menu. The **Help** menu should have a mnemonic of **H**,” (p. 9-70).

Guidelines and styleguides can, according to Smith and Mosier (1986), be utilized in connection with both design and evaluation of a user interface. Here, both situations are briefly described. When designing a user interface, guidelines and styleguides may be used in the following way:

1. prior to first design of a user interface, study those guidelines and styleguides that describe advantages and disadvantages of different styles of interaction,
2. when overall design of the user interface is decided, study relevant guideline and styleguide documents to identify guidelines and styleguides valid for the specific design,
3. use the guidelines and styleguides to review the design.

Many guidelines need to be concretized, if they are to be used in a specific design and implementation situation (Smith, 1988). An example of concretizing a guideline is:

guideline: Give feedback.

concretized guideline: Every user action that leads to a processing time longer than 5 seconds shall result in that feedback is presented. The feedback should indicate length of time necessary for processing and time elapsed.

Guidelines and styleguides are also useful in conjunction with identification of software requirements. Guidelines and styleguides may here be used in much the same way as in design. An advantage is incorporation of guidelines and styleguides into the requirement specification document (SRS).

In relation to evaluation, guidelines and styleguides can be used in following manner:

1. inspect user interface to be evaluated and identify user interface elements (menus, dialog boxes, and so on),
2. review relevant guideline and styleguide documents to identify valid guidelines and styleguides,
3. with identified guidelines and styleguides as a basis, evaluate the user interface.

The outcome from use of guidelines and styleguides is often a list of guidelines and styleguides with which the application does not conform. This list can then be used to inform further development of the user interface.

### *3.2.7 Prototyping*

Prototyping (sometimes also called modeling, see, for example, Andriole, 1989; ASTM, 1991; IEEE P1233-1993, 1994) is a method based on understanding the difficulty to define user requirements for the computer system to be developed. Prototypes concretize requirements, and users have the opportunity to validate requirements using the prototype, (Andriole, 1989; 1990; Andriole & Adelman, 1995; ASTM, 1991; Wood & Kang, 1992). Prototyping is also used in user interface design (see, for example, Nielsen, 1993).

Prototyping has many purposes. Here are mentioned some from the above references. The purpose of prototyping is to:

- facilitate communication between developers and users and between multiple developers,

- make it possible to concretize, in many cases, abstract user requirements,
- facilitate validation that requirements on the system to be developed are correct,
- facilitate identification of new requirements.

According to Sommerville (1992), there are the following activities in prototyping:

1. establishment of prototyping objectives,
2. selection of functions (requirements/services) to be included in a prototype,
3. development of a prototype,
4. evaluation of the prototype.

These four steps are supplemented with a fifth step (Andriole, 1990; Boar, 1984; Miller-Jacobs, 1991; Nielsen, 1993):

5. iteration of above steps until the prototyping work is considered finished.

Below, these activities are described in more detail.

Establishment of prototyping objectives. It is important to define the objectives of the prototype so that users and/or customers do not misunderstand the purpose of the prototyping work. This can result in erroneous expectations and frustration.

Selection of functions (requirements/services) to be included in the prototype. It is also important to define which functions (services) to implement in the prototype, and which not to implement. The reason for this is the necessity to

define what to include in the prototype, prior to prototyping. This is particularly important for the subsequent evaluation.

Development of the prototype. The prototype is developed using one of the techniques mentioned below.

Evaluation of the prototype. According to a number of authors, this is the most important activity in prototyping (see, for example, Andriole, 1990; Andriole & Adelman, 1995; Nielsen, 1993). In this activity, the prototype being developed is evaluated against defined goals (purposes) and services (functions). Misunderstandings, deficiencies, oversights and new requirements are taken care of in conjunction with next iteration.

Iteration of above steps until the prototyping work is considered as finished. It is crucial to recognize that prototyping is an iterative process, where a prototype is further developed until the requirements and/or the evaluation are judged to be reasonably comprehensive. Another aspect that influences the conclusion of prototyping, is of course that the purpose of prototype development is fulfilled.

There are a number of ways to create prototypes. Many authors differentiate between vertical and horizontal prototyping (see, for example, Nielsen, 1993), and between throw-away and evolutionary prototyping (see, for example, Dix et al., 1993). Vertical prototyping, is development of a prototype that is restricted to a specific application area, within this area the prototype is fully developed. Horizontal prototyping is development of a prototype to illustrate the complete system (user interface), where the prototype has restricted functionality (Nielsen, 1993). Throw-away prototyping (also called exploratory programming, Sommerville, 1992), is characterized by utilizing



prototyping only for definition of requirements upon the future computer system. When requirements are defined, the prototype is 'thrown away'. The knowledge acquired is, for example, used as input to a requirement specification. Evolutionary prototyping means that a system is gradually developed with the first prototype as a basis (Dix, et al., 1993).

There are many techniques for developing prototypes, depending on the demands for realism. Examples of techniques are:

- use cases (narratives, scenarios),
- paper copies of screen displays,
- storyboards,
- dynamic paper prototypes,
- limited functionality simulations,
- high-fidelity prototypes (high functionality simulations),
- selective fidelity prototypes.

Below, the different techniques are described briefly.

Use cases, (Andriole, 1989; also called Scenarios, Carroll, 1995; Nielsen, 1993) is a technique to simply describe what a system (or some part of it) should do, information needed, and result to be generated. A use case (scenario) is, according to Nielsen (1993), (see also Carroll, 1995), a written description of:

- "an individual user,
- using a specific set of computer capabilities,
- to achieve a specific outcome,
- under specified circumstances,
- over a certain time interval," (p. 100).

Use cases can be used in working with the user, to inspect and discuss different proposals relative to situations of use.

Paper copies of screen displays, (Andriole, 1989) are simple sketches concerning proposals for user interfaces, developed using some drawing program or drawn by hand. The sketches are used to illustrate for the user how the computer system (user interface) can be designed. The user can inspect proposals and make comments.

Storyboards, are, according to Dix et al., (1993), (see also Andriole, 1991), a graphical (and often animated) description of a proposal for a computer system user interface. A storyboard presents snapshots of the user interface in different interaction situations. With different kinds of computer programs, it is possible to give storyboards some dynamic features, letting the program 'play' a sequence of snapshots.

Dynamic paper prototypes, (Rettig, 1994) is a technique to illustrate and evaluate a user interface proposal with the help of paper and pencil. The technique is to prepare pictures describing different kinds of possible dialog states. In evaluating the prototype, someone is acting as a 'window manager' and presents the picture that is the result from a user 'button press,' 'menu choice,' and so on.

Limited functionality simulations, (Dix et al., 1993) are simple computer based prototypes, developed using a simple prototyping tool (for example, HyperCard, or Visual Basic). The purpose with these prototypes is to illustrate and evaluate interactive aspects of a future computer system. Another example of a technique for illustrating and evaluating interactive aspects is 'Wizard of Oz', where someone in the development team is acting

as an intermediate between the user and the prototype (Gould, Conti & Hovanyecz, 1983; Maulsby, Greenberg & Mander, 1993).

High-fidelity prototypes, (Löwgren, 1993, also called high functionality simulations, Dix, et al., 1993) are prototypes developed using techniques very similar to the technique to be used with the real computer system. This means that the prototype is going to be very realistic, giving opportunity for evaluations very similar to future use of the computer system.

Selective fidelity prototypes, (Voss, 1993, also called selective functional prototyping, Allusi, 1991) are prototypes based on identification of critical functions necessary to a user when accomplishing a task. The critical functions are developed in the prototype. Not so critical functions are developed using more simple prototyping techniques, for example, limited functionality simulations, or are omitted.

### *3.2.8 Contextual Design*

Contextual design is a method focused on customer driven development of computer systems (see, for example, Whiteside et al., 1988; Wixon, Holtzblatt & Knox, 1990; Beyer & Holtzblatt, 1993; Holtzblatt & Beyer, 1993; Holtzblatt & Jones, 1993). The authors state that this method moves the focus of system development to the customer and/or user and their work situation. Hence, giving the customer and/or user greater influence over system development. With this approach focus is on customers, continuous iterations, a common understanding of the user's work in the development team and continuous testing/evaluation of prototypes in the customer environment with real work tasks.

According to Holtzblatt and Beyer (1993), contextual design consists of the following five main activities:

- contextual inquiry,
- modeling of work,
- re-design of work,
- design of system (user environment),
- design of user interface.

Contextual inquiry, is an interview-based technique to study the customer/user during performance of work in the actual work environment. With contextual inquiry, the developer observes the work and continuously asks supplementing questions, to understand the work (see also Whiteside et al., 1988). During this study, investigation of issues concerning what is done; why it is done; and how it is done, is made. Observations made and answers to questions are written down. Usually, a number of studies are conducted with different customers/users within an organization, to obtain as complete description of work as possible.

When the majority of studies have been completed, the design team meets to compile the data and to discuss interpretations. To support this process, affinity diagrams are utilized. The diagrams are created by organizing the data in different groups on walls. Each grouping is given a descriptive label. After the grouping, each group is discussed and design ideas are created. The design ideas are written down in connection with the group discussed. (A more complete description of this process can be found in Holtzblatt & Jones, 1993).

Modeling of the work, is performed after diagrams are 'fully developed'. The models advocated by Holtzblatt and Jones (1993) are:

- context models,
- physical models,
- flow models,
- sequence models.

Context models illustrate, for example, how organizational, cultural and procedural factors constrain and create expectations on how people perform work, and what they produce. Physical models illustrate how the physical environment and the physical system influence work. Physical models also illustrate if work is distributed to different physical locations. Flow models illustrate different roles people take in their work. Each role represents a kind of customer for, or user of, a computer system. Flow models also illustrate what is needed and what is supplied to carry out a role. Sequence models illustrate the time aspect for accomplishment of activities in work. These models also illustrate specific tasks performed and specify in detail work to be supported by a computer system.

Re-design of work, is performed after modeling of work has been carried out. The purpose of re-design is to modify work to be performed, to cause maximum efficiency of use within the system to be developed. In this re-design, the same kind of models are used as in earlier modeling. The difference is that here, abstract models are developed. To develop these models, every existing model, of a specific kind and for a certain work task, are collected. From these models, a new model is developed. Specific details concerning individual workers are eliminated and the basic structure within the work is emphasized. Each model is validated against already obtained data about the work and further contextual inquiries in concert with new



customers/users. Already developed design ideas are also tested against the abstract model to evaluate how close the correspondence is with new models of work of the envisioned processes.

Design of system, (user environment) is performed with these new models as a basis. To avoid discussions concerning detailed user interface design, Holtzblatt and Beyer (1993) have developed what they call 'User Environment Design'. This technique is used to design the structure and function within a system. This is carried out by identifying focus areas, explicitly defined places within a system for performing a particular activity. For each focus area, functions and work objects necessary to carry out work are defined.

In design of user interface, an appropriate user interface is designed for each focus area. The design is then tested by evaluating paper prototypes in the work place. The customer/user is requested to carry out their work with the prototype, observations are made, and supplementing questions are asked. This process is iterated with more and more fully developed prototypes, to the point where a computer based prototype has been developed and tested. The final computer system is developed from this prototype.

### 3.2.9 Use Testing (*Usability Testing*)

Usability testing, or use testing as named here, is a systematic way of studying when users try to use a computer system (or a prototype) to carry out their work. Information about problems they encounter or experience is collected (Dumas & Redish, 1994; Lewis & Rieman, 1993). Use testing is also called empirical testing (Adelman, 1992) or empirical evaluation (Adelman & Donnell, 1986).

According to Dumas and Redish (1994), use testing is characterized by following qualities:

- the main goal with use testing is to improve usability of the computer system,
- the participants in use testing are real users,
- the participants in use testing try to carry out real work tasks with the help of the system,
- what participants are doing, and what they say, are registered,
- participant behavior and statements are analyzed to diagnose real usability problems, and to suggest proper actions.

Below, these characteristics are described in more detail.

Improvement of usability, is the purpose of use testing. The primary goal is to identify possible problems a user has when utilizing a computer system (or prototype). According to the authors mentioned above, use testing can be used to evaluate prototypes, early versions of a computer system, and already developed computer systems. (The above description is supplemented with the following; 'the main goal of use testing is to improve usability of the future computer system,' and 'the participants in use testing try to carry out real work tasks with the help of the system, or prototype,'). According to Lewis and Rieman (1993), it is important to remember it is the computer system, and its possible deficits, that is evaluated, not the user.

Real users, are of great importance in use testing. If participants in use testing do not represent users of a planned system, faulty conclusions concerning usability may easily be drawn. If not possible to find representative users, at least users that as much as possible are a representative sample of future users should participate.

Realistic work tasks, are necessary to make valid conclusions concerning usability of a computer system from use testing. If the computer system under development is complex, it may be necessary to select some out of all possible work tasks for actual use testing. Here, it is important to choose work tasks on the basis of use testing goals. According to Lewis and Rieman (1993), it is important that tasks selected are not too fragmented. If work tasks consist of several sub-tasks, it is important to incorporate all these sub-tasks into use testing.

Registration of user behavior and statements, is carried out to make subsequent aggregations and analysis of data possible. According to Lewis and Rieman (1993), there are two basic approaches for collecting data concerning user interaction with a computer system:

- collection of data regarding what users are doing and how they carry out tasks,
- collection of data concerning how efficiently users carry out a task or tasks.

To obtain data about what users are doing, and how they are carrying out their tasks, it is, for example, possible to observe users during their work with the computer system (or prototype). Every problem recognized in conjunction with task performance is noted, and then discussed with users. These observations can be supplemented with video recordings of user utilization of the computer system being studied.

It is also possible to utilize a technique called 'think aloud' (Lewis, 1982) in order to understand what a user is thinking of during use of a computer

system. 'Think aloud' is carried out by asking users to report verbally what they are thinking of when performing a task, for example:

- what they are trying to do,
- questions emerging in connection with task performance,
- what they are looking for on the screen,
- what different messages mean.

To make 'think aloud' more efficient, it is usually necessary for an evaluator to give users some help by asking questions about what they are thinking. This is especially important if a user is quiet for a long time. This can mean preoccupation with thinking of a solution to some issue, and thus forgetting to 'think aloud.' It is very important that an evaluator is neutral in asking, and avoids giving hints to users about what to answer or do.

To get data about user performance in conjunction with accomplishment of a task or tasks it is also possible to register, for example:

- time used by a user in carrying out a task or tasks,
- number of erroneous actions in connection with task performance,
- number of times the help function is utilized (if any),
- if the task was possible to carry out at all.

If usability requirements have been specified in advance (see sub-section 3.2.3), there are natural measures against which to evaluate actual user performance.

Analysis, diagnosis and change, is perhaps the most important aspect of use testing. Collected data must be analyzed and used to diagnose what real

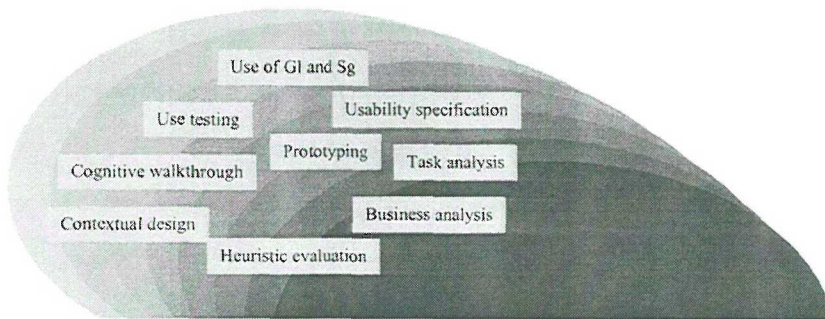


usability problems exist. This diagnosis must then be used to further develop a computer system, otherwise use testing is of minimal value.

Use testing can be performed on prototypes, early versions of a computer system and already developed computer systems. Often, valuable information regarding development of a new computer system can be obtained by use testing the old computer system. It is also possible to use test different parts of a computer system, for example, evaluation of installation and/or maintenance of hardware and software.

### 3.3 Conclusions

If we consider usability work in industrial system development as different kinds of activities that can be performed in conjunction with system development, it is possible to illustrate this in Figure 7, below.



**Figure 7: Usability work in industrial system development**

In industrial system development, usability work can be carried out at a number of places in the overall process. Below, this issue is reviewed and



discussed. With the industrial system development process (as described in Chapter 2) as a reference, methods described earlier in this Chapter are analyzed from the following perspective:

- authors of the various methods views about when in system development their methods shall be used,
- how method authors intend the methods to be used,
- why, according to various authors, the methods should be used (outcome, benefits),
- authors' views concerning need for carrying out supplementary methods,
- experiences from practical use of the method,
- my own experiences in system development and usability work.

### *3.3.1 Business Analysis (RASP)*

When shall business analysis (RASP) be used:

RASP, as with other methods for business analysis (see, for example, the SIM method, Goldkuhl & Röstlinger, 1988), is especially suited to identify possible need for computer system development, and which business part(s) will benefit from computer support. Business analysis provides greatest benefit initially in the system development process before any decision about what to develop.

### Utilizing business analysis (RASP):

Activities performed in business analysis are, according to RASP, primarily the following:

- description of business,
- need analysis,
- business development.

In description of business, present business is analyzed and described using functional modeling and concept modeling. This activity considers a business as hierarchical functions consisting of sub-functions. Main functions are broken down into sub-functions, to a level where business people, method experts, and customer of business analysis agree that descriptions accurately portray present business.

With these descriptions as a basis, possible change needs are identified and prioritized. When change needs have been identified and prioritized, one or more possible future business are designed, using functional modeling and concept modeling. When business design is carried out at the activity level, techniques for flow modeling are used (flow charts, Petri-diagrams, Gant-diagrams).

### Result of business analysis (RASP):

The result generated by RASP is a detailed description of present business, a prioritization of change needs, and one or more proposals concerning design of new business. The design of new business can include overall changes on the business level, as well as detailed changes of specific activities. According to the authors (Telub AB & System Development Associates, 1990; Telub AB, 1995), these results can be used as a basis for development of a computer

system, and/or development of organization and/or development of business staff.

#### Need for other methods:

In descriptions of RASP, no other methods are mentioned as a necessary pre-condition or as a natural continuation. In the case where business analysis (RASP) is followed by development of a computer system, authors seem to conclude that activity models resulting from flow charts etc., are sufficient.

#### Practical experiences:

Practical experiences from RASP indicate that the strength of this type of method is mainly in creating a basis for decisions about any of the following actions; development of organization, development of computer system, personnel development, or a combination of these actions (Enqvist & Lethovaara, 1996).

#### My own conclusions:

RASP (as most of the other methods for business analysis) has been primarily used to analyze and model businesses that are administrative in character. This suggests that possible benefits and disadvantages in connection with development of more complex computer systems, for example, command and control systems and process control systems, is not clear. Business analysis is also rather abstract, since most of the analysis is performed in meetings, where business is discussed in a rather theoretical way.

It is also uncertain if the activity models mentioned above are sufficient for development of computer systems, since these models provide abstract descriptions on how activities shall be accomplished. In addition,

performance of business analysis is rather resource demanding. Active engagement and commitment is needed from a number of business people.

Despite these conclusions, it is often important to perform business analysis in connection with possible computer system development. The reasons for this are:

- business analysis offers a better basis for decisions concerning possible need for development of a computer system. Many times computer system development is accomplished without any analysis of possible need,
- it is difficult, and sometimes impossible, to develop a computer system that supports a business (at least if the computer system is to support complex businesses) without detailed knowledge of the business (see also Andriole, 1990; 1996).

### 3.3.2 Task Analysis (KAT)

When shall task analysis (KAT) be used:

According to the authors, the KAT method is particularly effective for generation of ideas about services needed in a future computer system, and in evaluation of a developed computer system. The authors describe task analysis as supporting the following system development activities (with focus on user interface):

- feasibility study/initial planning,
- requirement definition/analysis,
- design,
- prototyping,

- validation,
- update and maintenance.

#### Utilizing Task analysis (KAT):

The activities performed in task analysis, according to the KAT method, are:

- identification of knowledge people possess about a task or tasks,
- analysis of task knowledge,
- modeling of present and/or future task or tasks.

In identification of task knowledge, goal and sub-goals that motivate task performance are identified. Then, procedures used in task performance are identified. The procedures are used as a basis to determine objects used and actions taken during task performance. This information is collected through interviews, observations and similar techniques.

In the subsequent analysis, goals, sub-goals, procedures, objects and actions that are; task typical; necessary; and common, are identified. This analysis meets the need for prioritizing and aggregating different task aspects.

Modeling of present/future tasks is carried out by creating a goal structure, a procedure sub-structure and a taxonomic sub-structure. In the goal structure, relationships between goals and sub-goals are described. The procedure sub-structure describes how tasks are/or shall be performed. In the taxonomic sub-structure relations between objects and actions, included in tasks, are described.



### Result of Task Analysis (KAT):

The result of task analysis, according to KAT, is:

- a description of task knowledge people possess. In other words, a description of goals, sub-goals, how tasks are performed, objects used and actions taken,
- a prioritization of task knowledge gained from descriptions,
- a model describing how to achieve a more efficient task design. Here, detailed information about task goals, sub-goals, how the task/tasks can be performed, objects needed and actions to be taken are included.

### Need for other methods:

The authors do not describe need for any supplementing methods. Neither is the need for results from any other method mentioned, nor that results from task analysis shall be used in any other method.

### Practical experiences:

Practical experiences from the utilization of task analysis in connection with system development are not extensive, (see, for example, Johnson, 1992; Johnson, Johnson & Wilson, 1995). Therefore, it is difficult to make any definitive conclusions about applicability in industrial system development.

### My own conclusions:

According to my assessment, task analysis in line with KAT is very resource demanding, as task analysis is quite detailed. This implies that it is difficult to motivate accomplishment of task analysis in an industrial system development setting where, for example, demands on delivery time are high. A possible solution to this problem can be to accomplish part of task analysis early in the system development process (identification of user requirements).

The main purpose being to create goal and procedure structures. The development of the taxonomic structure can be accomplished later, for example, in conjunction with software design. Partitioning of task analysis is also discussed in Johnson and Johnson (1991).

Performance of task analysis is especially important when developing systems characterized by high interactivity and critical task situations. It is important to have detailed knowledge about tasks to be accomplished with the system, if the system shall support users in complex situations with high demands on user actions. For example, systems for command and control, and process control.

### *3.3.3 Usability Specification*

#### When shall usability specification be used:

According to Carroll and Rosson (1985) and Whiteside et al., (1988), usability specification is used to create an understanding in developers about the usability goals, and to define measurable requirements against which a computer system can be evaluated. Usability specification is most beneficial when used in connection with specification of a computer system, and in testing of a developed computer system. Carroll and Rosson (1985) advocate that usability specification shall be used in connection with development of the functional specification (SRS). They add that the usability specification (possibly decomposed into subskills) shall be used continuously throughout the system development process (development of the user interface) to support continuous usability (use) testing.

### Utilizing usability specification:

The activities performed in usability specification are, according to Carroll and Rosson (1985), the following:

- identification of functional goals and usability goals (if not already identified),
- definition of usability requirements,
- identification of necessary sub-skills,
- testing system parts (user interface elements) utilizing these sub-skills,
- testing system usability with the usability requirements as the test criteria.

In identification of functional goals and usability goals, services the computer system shall provide are defined, along with how users want/have to work with the services. In definition of usability requirements, the task or tasks the system shall support, how this task or tasks shall be supported, and to what extent the tasks shall be supported, are defined. For identification of sub-skills the individual proficiencies needed for task completion are identified. Using sub-skills as a basis, separate parts of a computer system are usability tested. For example, comprehension of menu items. When the system is more fully developed, more complete parts are usability tested against previously defined usability requirements.

### Result of usability specification:

The outcome from utilizing this method can, according to Carroll and Rosson (1985) and Whiteside et al., (1988), be summarized as follows:

- increased knowledge about what is needed to make a computer system usable, and explicit definition of what is required of the system to be deemed usable,

- possibility to usability test a computer system during its development.

#### Need for other methods:

No author mentions need for other methods as a prerequisite for usability specification. The authors indirectly suggest task analysis, as they describe a need for knowledge about tasks to be performed as data needed for development of a usability specification. They also suggest use testing (in some form) as this method is intimately connected to usability specification.

#### Practical experiences:

The authors reports few explicit experiences from use of usability specification in practical system development. In Carroll and Rosson (1985), there is a description of an example of usability specification in development of a word processing system. However, from the description, it is difficult to conclude if it is a theoretical example or an actual system development project. In Whiteside et al., (1988) there are reports of the authors' experiences that usability specification is useful. However, a problem is that usability specification tend to express developers' usability requirements, and not those of users'. Other authors (see, for example, Carlshamre, 1994; Wiklund, 1994) report that usability specification is useful.

#### My own conclusions:

Although there are reports concerning the effectiveness of usability specification, it is difficult to make any definitive claims about applicability to industrial system development. One reason is that usability specification has not been used to supplement functional specifications (software requirement specifications, SRS). A second reason is that although Carlshamre (1994) report that they use usability specifications in evaluating a prototype, nothing

is mentioned about its influence on usability of the final computer system. However, it is probable that:

- usability specification can be useful,
- usability specification is very resource demanding.

Usability specification can be beneficial in industrial system development, as it is rather easy (at least theoretically) to integrate with a software requirement specification (see also Carroll & Rosson, 1985). Particularly because of ease of integration, usability specification can be a valuable supplement to the functional perspective. This is especially true if also considering the possibility of using usability specifications to identify and present performance measures from a use perspective.

Usability specification requires too much time and effort, especially if also the sub-skills are to be identified, for easy integration into the industrial system development process. Although testing is advocated in connection with implementation and integration, usability specification at the level advocated by Carroll and Rosson (1985) imply too much influence on the process, as much time and effort is needed for this detailed testing.

### *3.3.4 Heuristic Evaluation*

#### When shall heuristic evaluation be used:

According to Nielsen (1993; 1994), heuristic evaluation can be utilized on early prototypes, as well as user interfaces under development. He does not advocate that heuristic evaluation be used in evaluation of already developed user interfaces.



### Utilizing heuristic evaluation:

The activities performed in heuristic evaluation are, according to Nielsen (1994):

- definition of the rules of thumb (usability principles) to be used in the evaluation,
- evaluation of the user interface by three to five experienced usability experts,
- compilation and prioritization of results from the evaluation.

In definition of the rules of thumb to be used, recommendation is to use the rules of thumbs mentioned in section 3.4 as a basis. These can be supplemented with domain specific rules (Nielsen, 1994, p. 29). After rule definition, an evaluation may be performed. Each evaluator performs evaluation separately and inspects the user interface at least twice. The first time to be familiar with the user interface, second to inspect the user interface from the perspective of rules of thumb defined. After inspection, evaluators meet and aggregate their comments. The results are analyzed to determine degree of importance of usability problems.

### Result of heuristic evaluation:

The result of heuristic evaluation is a compiled and prioritized list of possible usability problems with respect to the rules of thumb used. This list can then be used to decide about changes in the user interface.

### Need for other methods:

According to Nielsen (1993; 1994), it is sometimes necessary to perform task analysis prior to accomplishment of heuristic evaluation. According to Nielsen (1994), task analysis may be necessary before starting a heuristic

evaluation of a computer system intended for specific users. Often, a use scenario from a task analysis is required for usability experts to conduct heuristic evaluation in an efficient way.

#### Practical experiences:

Nielsen (1992; 1993; 1994) has utilized heuristic evaluation in system development projects, demonstrating the value of the method. Studies by Jeffries et al., (1991), Desurvire (1994) and Karat (1994) also underline the value of heuristic evaluation. However, this is based on the premise that people performing heuristic evaluation are usability experts (Desurvire, Kondziela & Atwood, 1992; Karat, Campbell & Fiegel, 1992; Nielsen, 1994).

#### My own conclusions:

While, a number of authors have shown through their work that heuristic evaluation is of value in detecting possible usability problems, there are a few important issues to review. The first concerns number of usability problems detected with a heuristic evaluation. According to Desurvire (1994), proportionally fewer usability problems are detected with a heuristic evaluation (and other inspection methods) when compared to usability testing. Of the problems detected, very few are related to tasks performed with the computer system. The second issue relates to the need for experienced usability experts. There are very few usability experts involved in industrial system development (at least in Sweden). This can result in difficulty to utilize heuristic evaluation in industrial system development. However, heuristic evaluation by persons that are not usability experts is to prefer if usability experts are not found (see, Nielsen, 1994, 1995, for a discussion of discount usability engineering, where he advocates performing usability engineering even when resources are scarce).

### 3.3.5 Cognitive Walkthrough

#### When shall cognitive walkthrough be used:

According to Wharton et al., (1994), cognitive walkthrough can be used in conjunction with detailed design of the user interface. It can also be used to evaluate how easy a simple or more advanced (user interface) prototype is to learn and use.

#### Utilizing cognitive walkthrough:

The activities performed in cognitive walkthrough are:

- definition of necessary data for the cognitive walkthrough,
- performance of the cognitive walkthrough,
- development of suggestions to change actions/changed design.

The process of definition of necessary data, involves identification and description of tasks to be used, actions required to perform the tasks, possible users and their knowledge. With this information as a basis, cognitive walkthrough is performed. Tasks and actions are used in inspection of the user interface (a written description, a prototype, or a system) to judge if a user wants to perform the actions, if a user can perform them, and if a user is able to determine that necessary actions have been performed. From the result of the inspection, possible changes to the user interface are recommended.

#### Result of cognitive walkthrough:

The result of a cognitive walkthrough is a detailed description of possible usability problems, with respect to user need of the functions (services) in the user interface, the possibility to perform the task or tasks, and understanding of actions performed.

### Need for other methods:

The authors do not mention need for supplementing methods. However, they point to the need for detailed task knowledge and understanding of user characteristics as a necessary precondition to a cognitive walkthrough. This requirement suggests that there is a need for at least some form of task analysis.

### Practical experiences:

Cognitive walkthrough does not seem to have been used extensively in practical system development. Practical experiences in using the method appear to be based mainly on the authors' own testing in connection with further development of the method. This testing suggests that a cognitive walkthrough will identify possible usability problems at a detailed level, as every user action is analyzed. The results from using cognitive walkthrough indicate that the method is very time consuming. To avoid this problem, Rowley and Rhoades (1992) further developed the cognitive walkthrough method, using video recording equipment along with a more informal and interactive evaluation session. This 'cognitive jogthrough' method requires less time to perform than the conventional cognitive walkthrough. Although, this method was used to evaluate an application, there is little information about its pros and cons in industrial system development.

### My own conclusions:

Practical experiences from utilizing cognitive walkthrough (and cognitive jogthrough) is not comprehensive. Experiences are largely acquired in connection with further development of the method and comparative studies of the method (see, for example, Desurvire, 1994; Jeffries, et al., 1991; Rowley & Rhoades, 1992). In these cases, cognitive walkthrough (and cognitive jogthrough) was used in conjunction with development of rather simple

computer systems. Therefore, it is difficult to make any definitive conclusions about its advantages and disadvantages in industrial system development.

Despite this, cognitive walkthrough (and cognitive jogthrough) is potentially valuable in the development of more complex computer systems. This is particularly true in the development of computer systems where user understanding, and management of the computer system is critical. As for example with command and control systems or process control systems. Cognitive walkthrough (and especially cognitive jogthrough) can be used in these settings to check that tasks can be performed at all. In other words, a simplified cognitive walkthrough can possibly be performed to study, on a general level, the issues mentioned in sub-section 3.2.5.

### *3.3.6 Use of Guidelines and Styleguides*

#### When shall guidelines and styleguides be used:

Guidelines and styleguides can, according to Smith and Mosier (1986) and Flygvapnet (1993), be utilized in conjunction with specification, design and evaluation of user interfaces.

#### Utilizing guidelines and styleguides:

The activities performed through the use of guidelines and styleguides are:

- identification of applicable guidelines and styleguides,
- inspection of design proposal or developed user interface,
- documentation of deviations from guidelines and styleguides.



In identification of applicable guidelines and styleguides, a design proposal or developed user interface is usually a starting point. From this, guidelines and styleguides judged as relevant are chosen. Guidelines and styleguides are then used when inspecting the design proposal or the developed user interface. During the inspection, possible deviations are documented.

#### Result of using guidelines and styleguides:

The result of using guidelines and styleguides is rather concrete, possible deviations are documented and then used to decide on user interface changes. Result are at a low level; deviations identified mainly concern deficiencies in menus, dialog boxes, etc.

#### Need for other methods:

No need for other methods is mentioned by the authors.

#### Practical experiences:

The practical experiences from using guidelines and styleguides are rather comprehensive (see, for example, De Souza & Bevan, 1990; Mosier & Smith, 1986; Tetzlaff & Schwartz, 1991; Thovtrup & Nielsen, 1991). These experiences indicate that guidelines and styleguides are difficult to use in a practical system development situation.

#### My own conclusions:

From the experiences of this author (see, for example, Löwgren & Nordqvist, 1990; 1992; Nordqvist, 1995), guidelines and styleguides are very difficult to use in industrial system development for the following reasons. First, guidelines and styleguides documents are very comprehensive and difficult to understand by developers. This discourages developer usage. Second, use

of guidelines and styleguides is very demanding on system development resources (time and personnel) when compared to obtained results. Use of guidelines and styleguides, as a consequence, is not given priority. A majority of the time they are never used in an evaluation setting. Nevertheless, it is important to use guidelines and styleguides for user interfaces to be consistent.

### *3.3.7 Prototyping*

#### When shall prototyping be used:

Prototyping can, according to the authors, be used early in the system development process to identify, concretize and evaluate the user requirements on the computer system to be developed. Prototyping can also be used in conjunction with user interface design.

#### Utilizing prototyping:

In prototyping the following activities are performed:

- identification of requirements (services) to be implemented in the prototype,
- selection of suitable prototyping technique,
- prototype development,
- prototype evaluation,
- possible further development of the prototype.

The point of departure in identification of requirements to be implemented in the prototype is usually an idea (or need) concerning a computer system, or the set of requirements identified together with customer/user. From this,

requirements (services) to be implemented are chosen, and an approach for implementation is determined (for example, horizontal or vertical prototyping). Depending on requirements and how extensive implementation will be, a prototyping technique is selected (for example, use cases, storyboards, high-fidelity prototypes). The selected technique is used to develop the prototype, which in turn is evaluated against, for example, user requirements identified/usability requirements. Using the evaluation as a basis, possible further development of the prototype is carried out. When essential user requirements are identified and validated, the prototype can function as a description of the set of requirements, or requirements can be documented in a requirement specification. As mentioned earlier, the prototype can also be evolutionary developed into the final system.

#### Result of prototyping:

Prototyping results, in conjunction with identification of user requirements, in that identified requirements are concretized, validated and further requirements are identified. In conjunction with user interface design the outcome is a user interface design proposal.

#### Need for other methods:

The authors mention need for use testing (see, for example, Andriole & Adelman, 1995) in evaluation of a prototype. The result from prototyping seem to be regarded as directly usable in subsequent system development work.

#### Practical experiences:

The practical experiences of prototyping are numerous. These suggest that prototyping is a powerful technique for definition of user requirements on the system to be developed. According to Andriole (1990; 1994), it is the most

successful method for defining user requirements (see also Miller-Jacobs, 1991; Gordon & Bieman, 1994, for discussions about the value of prototyping in system development). However, in industrial system development, prototyping has been difficult to fully introduce, as it is hard to plan and manage prototyping. Therefore, project managers have been reluctant to advocate prototyping as a tool for identification of user requirements (Sommerville, 1992). In connection with software design, prototyping has been used to support user interface design.

#### My own conclusions:

Based on my own experiences, following observations are made:

- prototyping is seldom used in connection with definition of user requirements. Instead, prototyping is most often utilized in connection with identification of software requirements and/or in user interface design,
- in cases where prototyping is performed, prototype evaluation is usually performed randomly. For example, a number of users try to use the prototype in an unstructured way, or is only requested to express their opinions.

The above experiences can result in developers not knowing what to develop early in the system development process (see also Andriole, 1990). The probability that a computer system is developed that fulfills user requirements and is usable is then minimal.

Further observations are:

- prototyping is very powerful as a technique to minimize differences in interpretation of user requirements. Prototyping avoids problems that

occur in the interpretation of complex and/or abstract requirements using only ordinary text documents (see also Sommerville, 1996).

- if simple prototyping techniques (paper copies of screen displays, storyboards) are combined with the 'Wizard of Oz' technique the communicative effect of these techniques are augmented.

### *3.3.8 Contextual Design*

#### When shall contextual design be used:

According to the authors, contextual design may be used throughout the system development process. It can be regarded as an alternative to more traditional system development methods (examples of traditional development methods/ models can be found in Boehm, 1988; Royce, 1970).

#### Utilizing contextual design:

The activities performed in contextual design are:

- contextual inquiry,
- modeling of work,
- (possible) re-design of work,
- design of computer system (user environment),
- design of user interface.

In contextual inquiry, users are observed and interviewed as they are performing their work, to clarify what is done, why it is done, and how. The results from these observations and interviews are then combined into affinity diagrams. When this grouping is 'fully developed,' organizational aspects, physical environment, roles performed by users and specific tasks performed,



are modeled. When necessary, these models are used to create abstract models of possible work changes. With these models as a basis, focus areas are used to design a computer system. Focus areas are also used to define functions and objects necessary to perform work, and to design a user interface. The proposed design is realized in a simple prototype and evaluated by users in the real work environment. The prototype is then further developed into the final system.

#### Result of contextual design:

Contextual design includes a description, in the form of notes and models, of work performed by users, as well as suggestions for work changes, system design and user interface design. According to the authors, the final result is a developed computer system.

#### Need for other methods:

The authors do not express any need for other methods before, or after contextual design. They advocate contextual design as a replacement for other approaches to system development. However, note that (evolutionary) prototyping is part of contextual design.

#### Practical experiences:

The practical experiences mentioned by the authors seem to emanate from projects whose main purpose is testing and further developing contextual design.

#### My own conclusions:

Practical experiences described by the authors are few, and it is difficult to make any definitive conclusions about contextual design's possible

advantages or disadvantages. From an industrial system development perspective, the advantages with contextual design is the focus on the user (customer), and what is done in a work situation. A disadvantage may be that contextual design appears to be designed for system development where there are no limitations on work situation study and prototype evaluation in a work setting. In industrial system development, study of work situations can be limited, and users are often unavailable (see also Nielsen, 1994). It is also uncertain if contextual design can replace more traditional system development methods, as it seems to neglect the need for more formal software requirement analysis and software design. In development of more complex computer systems, it is advisable to use formal analysis and design techniques that offer necessary structure. Contextual design can possibly supplement industrial system development. This method can deliver models concerning, for example, the physical environment, that in turn can be useful during system design.

### *3.3.9 Use Testing (Usability Testing):*

#### When shall use testing be used:

Use testing shall, according to the authors, be used to evaluate prototypes, early versions of computer systems, as well as fully developed computer systems.

#### Utilizing use testing:

The activities performed in use testing are:

- definition of purpose with the use testing,
- identification of work tasks and users,
- performance of use testing,

- analysis of use testing results.

In definition of purpose for the use testing, aspects of prototype or developed system to be studied, and what registrations to do, are determined. With the purpose of the use testing as a basis, representative work tasks and users are identified. Work tasks selected are then used during user evaluation of the prototype, or the developed computer system. The user tries to accomplish tasks using a prototype, or a developed computer system. In conjunction with this task performance, for example, what users are doing, what problems they have, how tasks are performed and how fast, are registered. These registrations are analyzed to identify possible usability problems and possible corrective actions.

#### Result of use testing:

The results of use testing are influenced by what is decided to be registered. Generally, the results of use testing are explicit indicators of possible usability problems that will occur when the computer system is operating in a real work situation.

#### Need for other methods:

The authors do not express need for other methods. However, they point to the need for realistic and representative work tasks. They also point to the need for real users during use testing. This implies need for task analysis and usability specification before use testing can be carried out. It is also necessary to have a 'system' to use test; either a prototype or an already developed computer system. Therefore prototyping is sometimes necessary.

### Practical experiences:

Use testing is, according to many authors, the evaluation method that best identifies possible usability problems (see, for example, Desurvire, 1994; Karat, 1994). Use testing has also been used in many system development projects and has proved its value (see, for example, Dumas & Redish; 1994; Karat, 1992).

### My own conclusions:

While use testing is the evaluation method that identifies most usability problems, and best corresponds to real use situations, there are some disadvantages with the method. In an industrial system development situation, there is usually insufficient time and money to perform use testing to the extent necessary. Especially if use testing is performed in the experimental fashion as discussed by Lewis and Rieman (1993). Another possible disadvantage is that use testing can be too artificial and arranged (to collect objective data) to realistically reflect real use.

Despite these possible disadvantages, use testing is of value and should be performed more often than is the case. In case resources for use testing are scarce, it is possible to perform use testing as described in discount usability engineering (see, for example, Nielsen, 1995) where a small number of users informally use test an application. From my experience it is also possible to use test simple prototypes (use-cases, paper copies of screen layout, storyboards) as well as other models early in computer system development. Although the empirical data obtained are not precise enough for definitive conclusions concerning usability issues, it is often of value to use test as practical evaluation often result in important information. Another reason for carrying out use testing also on simple prototypes is the necessity to confirm that the prototype (model) correctly describes the phenomenon under study.



## 4. USABILITY WORK AND INDUSTRIAL SYSTEM DEVELOPMENT

### 4.1 Introduction

Using the industrial system development process described in Chapter 2 as a basis, Figure 8 shows this author's interpretation of when methods presented should be used. In this interpretation, a method is related to one, or at the most two, system development activities; although some authors quite reasonably say their method support more activities. This is interpreted to mean that the result from a method may be used in more subsequent system development activities. A method is related to a system development activity as defined here, even when authors associate their method to an activity of a different name. For example, Nielsen (1993) states that heuristic evaluation should be used in connection with design and development of user interfaces. System development, as presented here, differentiates between system design and software design, therefore Nielsen is interpreted to advocate use of the method in conjunction with software design.

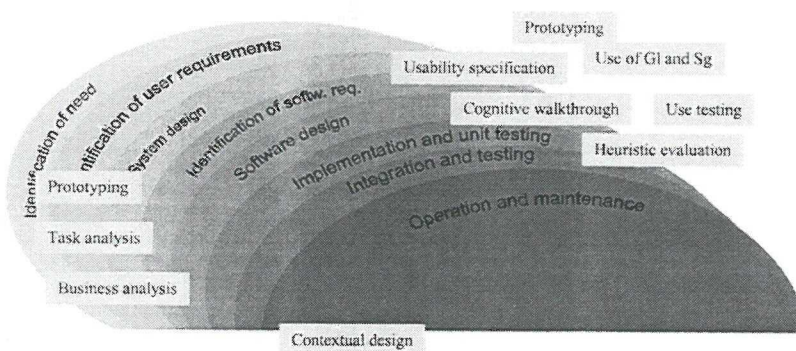


Figure 8: Methods for usability work and their use in industrial system development



As depicted in Figure 8, authors are interpreted to mean that methods for business analysis, task analysis, and prototyping are to be used during early system development activities (Identification of need, Identification of user requirements). Use of guidelines and styleguides, heuristic evaluation, cognitive walkthrough (jogthrough), usability specification, prototyping, and use testing, according to the authors, are used during later system development activities (Identification of software requirements, Software design, Implementation and unit testing, Integration and testing, Operation and maintenance). Contextual design is used in all system development activities. It is an alternative to the system development activities presented.

This interpretation of method use indicates that usability work is not carried out in some system development activities (not even theoretically). It also indicates that only minor usability work is performed in many system development activities, and that there is no continuity in usability work in the continuum of system development work. The consequence of this may be that user requirements identified in early system development activities are 'lost,' and may or may not be found again in later activities when use testing and other kinds of usability work are performed. Another consideration illustrated in Figure 8 is that many methods focus on few system development activities. This means that the potential benefits of usability work are not fully utilized. As an example, use testing seems to be carried out only in connection with implementation and unit testing, and integration and testing, despite the fact that this method could be beneficial in other system development activities, such as identification of user requirements.

As a preliminary attempt to increase the integration between usability work and industrial system development, and to increase the utilization of methods for usability work, a simple model together with examples of such an integration is outlined in the following sections.

## 4.2 Integration of Usability Work and Industrial System Development: A Preliminary Model

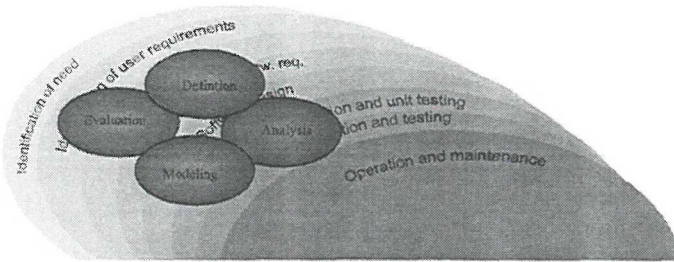


Figure 9: A model for integration of usability work and industrial system development

In principle, usability work can be seen as encompassing four activities; definition, analysis, modeling and evaluation. These activities should be seen as continuous iterations, where the result from definition is analyzed, modeled and then evaluated, to be followed by definition again if necessary. Definition, analysis, modeling and evaluation should, according to this model, be carried out as part of all system development activities. Figure 9 above, portrays how definition, analysis, modeling and evaluation overlap. The continuity in usability work throughout the entire system development process, is also illustrated in Figure 9. The methods presented in Chapter 3 can be summarized as a combination of definition, analysis, modeling and evaluation to a higher or lower degree. For example, in prototyping the services (functions) to be implemented are first defined and analyzed, then prototyped and finally, evaluated.

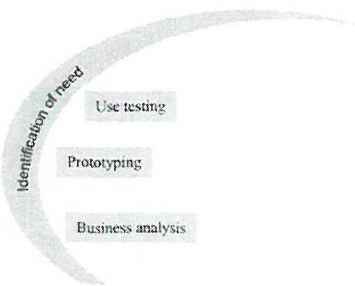
No individual method is enough to ensure that the computer system under development fulfills user requirements and becomes usable. Thus, there is reason to find a common denominator in these methods and to view them as a tool box rather than as separate methods. Another reason for this approach

is the increased likelihood that the methods are considered as able to support and complement each other. That earlier acquired results facilitate and make subsequent usability work more efficient.

### 4.3 An Example of Integration of Usability Work and Industrial System Development

This section gives an example on how to integrate usability work into industrial system development. System development activities are discussed from the perspective of usability work and examples of suitable methods and techniques are presented.

#### 4.3.1 Identification of Need



**Figure 10a: Integration of usability work and industrial system development**

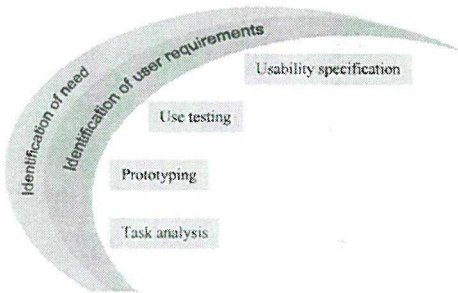
In accordance with the authors, (TELUB AB & System Development Associates, 1990; TELUB AB, 1995), the use of business analysis (RASP) in connection with identification of need is recommended. In the model advocated, see Figure 10a, business analysis is supplemented with

prototyping and use testing. The reason for this is that it is:

- difficult for users to understand and assimilate models (function graphs, object models), that are the outcome from business analysis according to RASP. (This is also true for other business analysis methods),
- difficult for users to decide if proposed design of future business, in the form of function graphs and object models, will result in a more effective business, unless these proposals are supplemented with prototypes that can be use tested.

It is not necessary that advanced prototypes are developed, and experimentally use tested. Rather, use of simple prototyping techniques, such as use cases, paper copies of screen displays ("screen layouts" means here suggestions on services to be delivered by the computer system, not pure screen layouts), and storyboards are recommended tools. Simple use testing is performed to increase understanding in users and developers of proposed business design and its effects. Use testing can, in this case, be use tests of simple prototypes where real users perform a small number of work tasks, and where the goal is to identify use problems and user opinions as to how proposed design will function during actual task performance. Another important aspect possible to check with use testing is that the prototype (or other model) is a correct description of the problem at hand. This validation of prototypes and other models is of course also necessary in subsequent system development activities.

#### 4.3.2 Identification of User Requirements (Requirements Definition)



**Figure 10b: Integration of usability work and industrial system development**

In identification of user requirements, see Figure 10b, task analysis (KAT), prototyping, and usability specification will provide a significant quantity of needed descriptors. By supplementing these three methods with use testing, a more complete picture of user requirements is possible. The reasons for advocating task analysis, prototyping, usability specification and use testing are that:

- task analysis in most cases is a precondition for identifying what to prototype (and develop),
- users have difficulty deciding if requirements are correct and if their expressed needs are being correctly interpreted by developers. When requirements and expressed needs are concretized with the help of prototypes which can be use tested clear communication with users is more likely,
- prototyping is an aid for defining reasonable usability requirements.

This implies that the first activity in connection with identification of user requirements is a task analysis. While some task analysis is necessary, it is not



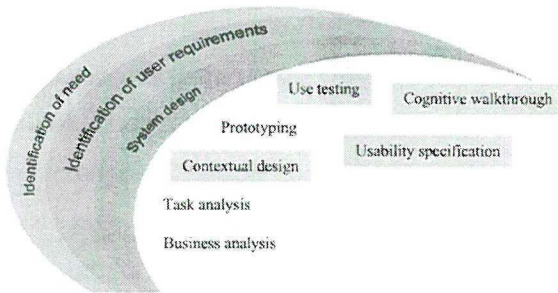
always necessary to perform a complete task analysis according to KAT. To create conditions for identification of user requirements it is initially enough to identify the work (business), and then do a preliminary selection of tasks to be supported by the computer system. Tasks to be supported can then be analyzed to identify goals, sub-goals, and procedures. With this analysis as a basis, it is possible to create simple prototypes (use cases, paper copies of screen displays or storyboards). The materials developed during identification of need may be modified and reused. Through this practice user involvement in development is fostered and iteration occurs. (This is true also in subsequent system development activities). Using advanced techniques and tools to create prototypes in this early stage of system development may cause users, as well as the people who support the users in identifying requirements, to inflexibly and too early decide on specific solutions. It is also easier (and cheaper) to create alternative solutions when using simple prototyping techniques.

Prototypes should be use tested to validate that requirements are appropriate and to identify further requirements. Use testing is best restricted to only investigating that necessary services are delivered by the prototype and that work tasks are possible to carry out at all. The reason for advocating use testing so early in system development is the importance of identifying user requirements on the system to be developed, before development starts (see also Andriole, 1990; 1996). If user needs are not known, a computer system that efficiently supports their needs is unlikely to be developed.

As the user requirements are analyzed, prototyped and evaluated, the usability specification can be specified. Even if those requirements in this early stage of system development risk being described too generally to be of direct use in later system development activities, they can serve as means for creating an understanding concerning the usability requirements for those

involved in the system development effort (see, for example, Whiteside et al., 1988).

#### 4.3.3 Overall Design of the System



**Figure 10c: Integration of usability work and industrial system development**

According to Johnson and Johnson (1990; 1991), the results from task analysis (KAT) can be used in connection with system design. Results from business analysis (RASP), contextual design and prototypes developed are also important input for system design, see Figure 10c. The result from business analysis is important because it gives an overview of which services the computer system should deliver, what other functions (services) are, and the relation between the computer system and other functions (the business). The result from task analysis serves, for example, as basis for defining general allocation of tasks between users and the computer system (Johnson & Johnson, 1991). Models of the physical environment created and focus areas identified in conjunction with contextual design also offer valuable results useful in system design. Prototypes can be used together with function graphs and physical environment models to create an overview model of the total system (user-task-organization-technology), which then can be use tested. Here, use testing is an informal test to determine if the computer system

(prototype) design fulfills defined user requirements and if defined work tasks can be performed. Also a simplified cognitive walkthrough (cognitive jogthrough) can be carried out on this overall model. Further development of the usability specification, for example refinement of performance measures are also advocated.

#### 4.3.4 Identification of Software Requirements (Software Requirements Analysis)

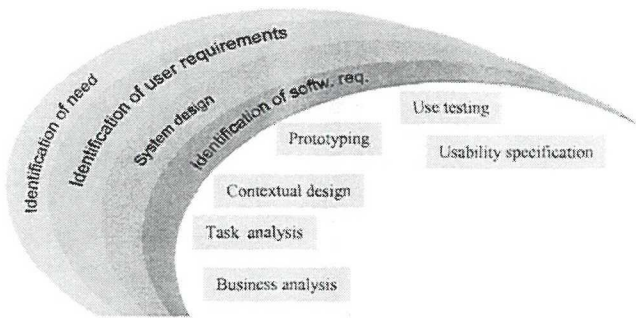


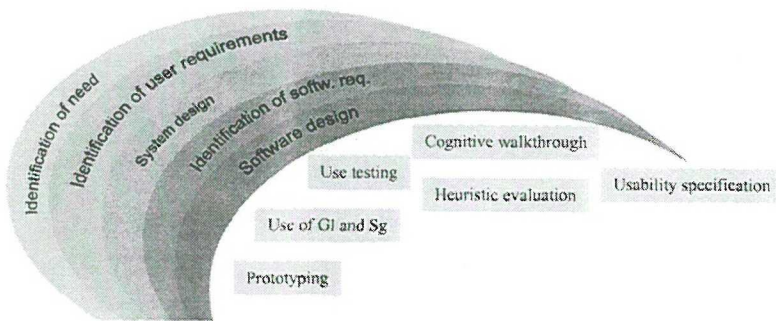
Figure 10d: Integration of usability work and industrial system development

As identification of software requirements focuses on translating user requirements from a non-specialized representation (natural language, graphs and prototypes) to a representation better suited for software development, usability work does not have an explicit role. However, it is important to ensure that user requirements are correctly “translated”. Therefore, it is valuable to continue the analysis of user requirements and to further develop models and prototypes in parallel with identification of software requirements. It is also advisable to use test these models and prototypes and to further develop the usability specification.

Further prototype development may be development of a completely new prototype. This is often necessary in conjunction with development of more

complex computer systems. In prototyping to identify user requirements, early ideas and interpretations of requirements, are frequently defective or even erroneous. Prototyping is used to modify and give a better understanding of user requirements on the system to be developed.

#### 4.3.5 Software Design



**Figure 10e: Integration of usability work and industrial system development**

Part of software design, is user interface design. Usability work is here focused on user interface prototyping in combination with inspection and testing of proposed design, see Figure 10e. Guidelines and styleguides that may be useful during design are identified. Here, it is necessary to identify two kinds of guidelines and styleguides. First, guidelines and styleguides concerning appropriate interaction techniques (see, for example Smith & Mosier, 1986). Second, guidelines and styleguides concerning design of specific user interface elements. (These guidelines and styleguides are also used during implementation & unit testing, and integration & testing).

User interface design proposals are implemented in one or more prototypes, (here, more advanced prototyping techniques such as dynamic prototypes,



limited functionality simulations, high functionality prototypes or selective fidelity prototypes will generate the more complex data needed at this stage). Prototypes should be evaluated using identified guidelines and styleguides, heuristic evaluation, cognitive walkthrough (cognitive jogthrough) and use testing in some combination. Cognitive walkthrough (Cognitive jogthrough) is important for inspection of user interface design, as this method explicitly has the tasks to be performed with the computer system as a basis. Use testing with the help of real users and real work tasks should be used later on in the software design, when a user interface has reached its "final" design. Before use testing, other techniques should be used to recognize simpler deficits in design, so that users are not used up (see also Nielsen, 1993). The usability specification should also be further developed, for use in implementation & unit testing, and integration & testing.

#### 4.3.6 Implementation and Unit Testing

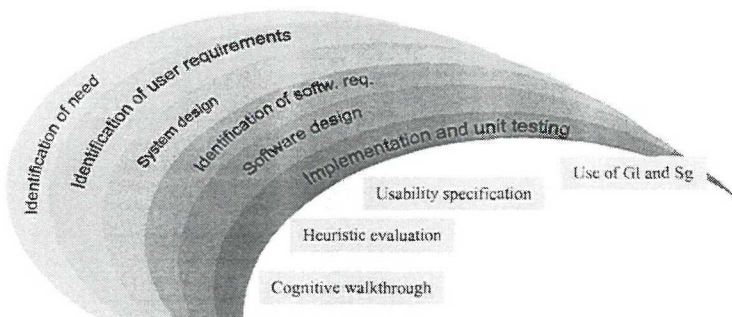


Figure 10f: Integration of usability work and industrial system development

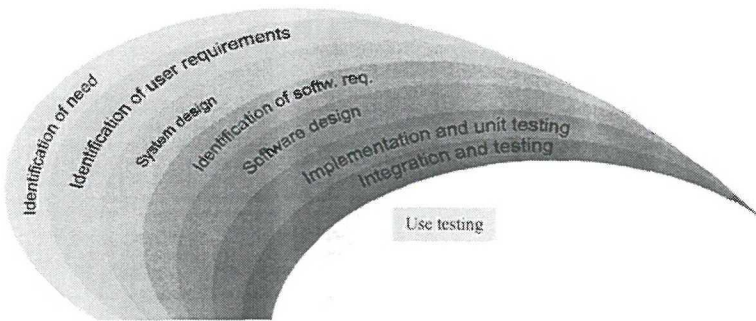
When software design, particularly user interface design, has the input from usability work, simpler usability evaluations such as cognitive walkthrough (cognitive jogthrough), heuristic evaluation, etc., are sufficient for the



usability work part of implementation and unit testing, see Figure 10f. It is for example, possible to depend on usability specifications for evaluation of user interface elements as they are being developed, (understanding of menus, dialog boxes etc.). However, only the most important user interface elements should be evaluated using this resource demanding method. In parallel with this evaluation, heuristic evaluation and/or use of guidelines and styleguides to continuously evaluate the user interface during implementation, are also advisable. Use of both guidelines/ styleguides and heuristic evaluation are sometimes recommended. In the requirement specification there may be explicit requirements to adhere to some specific guideline and/or styleguide. In this case, it is not enough to perform only heuristic evaluation.

Simple usability evaluations are sufficient if identification of need, identification of user requirements, system design, identification of software requirements, and software design have been performed with the help of usability work (especially prototyping and use testing). It is not always necessary to perform prototyping and use testing, which require major resource investment, when implementing and testing software.

#### 4.3.7 Integration and Testing



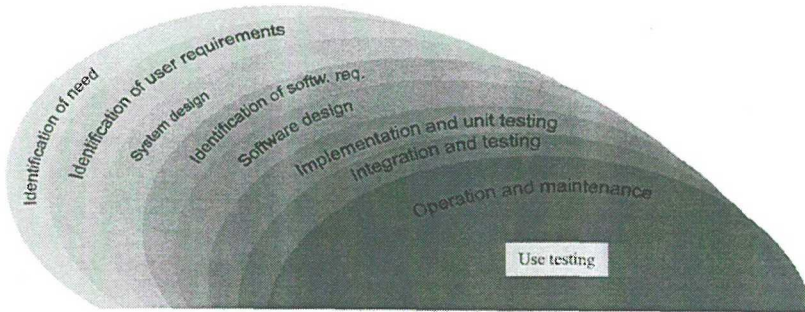
**Figure 10g: Integration of usability work and industrial system development**

When integrating and testing software it is valuable to again carry out use testing, see Figure 10g, because of following reasons:

- the importance of testing that the user requirements are fulfilled,
- the importance of testing for system usability,
- here is the first time when it is possible to evaluate the whole system from a usability perspective,
- use testing with real users and real work tasks create test conditions that most closely approximate actual use.

In this use testing the scenarios (use cases) and the usability specifications developed can be valuable input.

#### 4.3.8 Operation and Maintenance



**Figure 10h: Integration of usability work and industrial system development**

In connection with operation and maintenance it is also important to focus on use testing, see Figure 10h. This use testing is a supplement to delivery and acceptance testing. The developed system should be evaluated in its real environment, with real users and real work tasks. Scenarios (use cases) and usability specifications developed can also here be used as input to the use testing performed.

## 4.4 Conclusions

### 4.4.1 The Model

As illustrated in Figures 10a-h above, use of methods for usability work in all system development activities is recommended. Emphasis on usability work during all activities (especially early ones) in system development will hopefully minimize the risk of having to make costly and extensive changes late in the development process and after beginning actual system use.

The potential benefits of including usability work continuously in system development are:

- the computer system is being more usable as the user and use of the computer system is emphasized continuously during the system development process,
- results from earlier usability work can be used in later (and parallel) usability work, making it more efficient. For example, the result from task analysis can be used in and facilitate heuristic evaluation as well as cognitive walkthrough (jogthrough). Both of these methods require work task knowledge,
- there is considerable reduction in risk that user requirements are forgotten, or wrongly interpreted, during identification of software requirements, software design, implementation & unit testing, and integration & testing.

One reason for focusing on usability work during early system development activities is that it emphasizes user requirements and likely results in the system under development being more effective in the actual work situation (see also Andriole, 1990; Andriole & Monsanto, 1995). Another reason is that deficiencies and errors in requirements detected early in the system development process, are up to 200 times cheaper to correct than if detected during operation and maintenance (Davis, 1990; 1993). A third reason is that time delay and budget overruns in many cases are due to deficiencies in what is here called usability work, as omission early on requires larger later effort (Lederer & Prasad, 1992).

#### *4.4.2 Experiences*

The model has not been formally evaluated. What has been done is an informal evaluation in the way that business analysis, task analysis,

prototyping and use testing have been used in connection with identification of need and identification of user requirements. Also, prototyping, heuristic evaluation and use of guidelines and styleguides have been utilized in connection with software design (especially user interface design) and implementation and unit testing. In both cases the methods were used in real system development projects and/or system developers who have used above mentioned methods in real system development projects were interviewed. Examples of system development projects included are: a system for training of air force command and control, a system for military airfield command and control, a system for presenting and managing geographical information, and a system for presenting attacking enemy air forces. An indication of the magnitude of the projects is that they represent from 1 to 100 man months invested in the activities of identification of need, identification of user requirements and system design. Preliminary experiences from this evaluation are as follows.

In identification of need:

- business analysis imply a better understanding in users, as well as those responsible for development, of which change requirements could (and should) be fulfilled by a computer system (see also Andriole, 1996),
- business analysis requires a major investment in time and staff; as many people from business need to participate in the development process and business is frequently complex,
- it is sometimes difficult for users to understand results of business analysis (graphs, flow charts, object models),
- prototyping results in a better understanding of which computer system should be developed, which services the computer system should deliver (see also Andriole, 1996; Sommerville, 1996),
- it is easier for users to understand the effects of proposed business design, if models and prototypes are use tested.



In identification of user requirements:

- task analysis (in some form) is a necessary supplement to other methods, to identify work tasks to be supported and to obtain information about how to support work tasks,
- task analysis requires excessive resources (time and man power) in relation to the perceived utility, if carried out in exact accordance with all steps in the method,
- prototyping results in a better understanding of what computer system is necessary to develop (what services it should deliver),
- prototyping is often necessary for users to be able to actively contribute in the process of identifying requirements, concretize requirements and to help them understand their own requirements,
- prototyping sometimes causes users to "wish for everything,"
- prototyping sometimes results in users focusing on user interface issues, instead of issues of relevance to what services the system should deliver,
- prototyping sometimes results in too early fixation on an idea, or requirement set, then used as a basis for development of the final computer system. In the case where prototyping is used to identify requirements for a complex computer system, early ideas or requirement sets may miss some requirements and be defective. An early idea or requirement set used to develop a final system (evolutionary prototyping), risks developing a system that fails to efficiently support user work. In many situations it is necessary to discard an early prototype, and use the acquired experience for development of a new prototype,
- use testing helps users to understand the effects of their requirements and encourages them to identify other requirements than the most obvious,
- developers (and in some cases users) sometimes think that theoretical discussions about user requirements can replace use testing,

- use testing is sometimes perceived as requiring too much time to perform,
- when task analysis, prototyping and use testing have not been performed, user requirements tend to focus on what information is to be handled in the computer system, and what information is to be sent to and from the system. User requirements that focus on what the user should be able to do with the system are not given priority.

#### In software design:

- user interface prototyping is efficient to communicate user interface design proposals,
- heuristic evaluation is a fast and efficient way to inspect a design proposal (see also Nielsen, 1993),
- heuristic evaluation is sometimes perceived as inadequate as it primarily identifies “low-level” problems,
- use of guidelines and styleguides require too much resources compared to perceived contribution to usability,
- use of guidelines and styleguides are usually “forgotten” in software design, even if there is an explicit requirement to follow some guideline and/or styleguide.

#### In implementation and unit testing:

- same experiences as in software design.

#### Summary:

Preliminary experiences in carrying out usability work in industrial system development can be summarized as follows:

- usability work contributes to a better understanding in users, with respect to the system to be developed and the effects of the system,

- usability work leads to more comprehensive user influence upon what is being developed,
- usability work places greater demand on users because they, more explicitly, are responsible for what is being developed,
- usability work leads to greater demands on developers because they must understand what to develop in terms of user language,
- methods for usability work often "disturb" the system development process. Much is required of those who are going to use the methods and much resources in time and participating people are required (usability experts, users, developers, etc.),
- it is easy to get an understanding of the necessity to carry out usability work, but it is difficult to receive an adequate level of funding,
- it is difficult for outsiders to develop an understanding of the need to carry out usability work during early system development activities,
- usability work result in (require) user participation in system development.

## 5. THE NEED FOR EXTENDED INTEGRATION OF USABILITY WORK AND INDUSTRIAL SYSTEM DEVELOPMENT

### 5.1 Introduction

The preliminary model presented here represents only a first step in integration of usability work and industrial system development. Necessary future work to deepen integration is characterized by following:

- the need for closer study of how different usability work methods can be integrated to optimize use in an industrial system development setting,
- further identification and definition of appropriate usability work methods,
- the need for research and development that supplements and develops existing methods. For example, to simplify some methods for more effective application in industrial system development,
- further practical evaluation of the model and the “tool box” of methods advocated here,
- development of different kinds of computer support. For example, software that supports use of guidelines and styleguides, task analysis, business analysis, and use testing,
- more complete integration of usability work and industrial system development, so that usability work is perceived as a natural part of system development,
- education of those participating in system development to deepen understanding of usability work possibilities and deficiencies.

Below, a brief discussion of some of the issues above is presented. The focus is on discussing each issue from the perspective of concretizing and exemplifying what has to be done.

## **5.2 Further Integration of Usability Work**

The model depicted in Chapter 4, describes usability work methods to be carried out during system development to foster usability of a final product. The model also describes how to apply these usability work methods during system development activities. However, the model does not consider, in enough detail, how results from usability work in one system development activity relate to usability work in subsequent system development activities. It is also beneficial for understanding of usability work advantages and disadvantages to describe how usability work contributes, in a concrete way, to different system development activities. Another issue not handled in the model, is how usability work may effect the variability (development) of the requirement set in system development (see, for example, Andriole, 1996; Davis, 1993; Sommerville, 1992).

Most important is to elaborate the description of how results from one method for usability work can be used by other methods. The utilization of results was described in section 4.1, here it is concretized further. Below, a brief description of results obtainable from different usability work methods are presented, together with mention of how results can be used to support other methods.

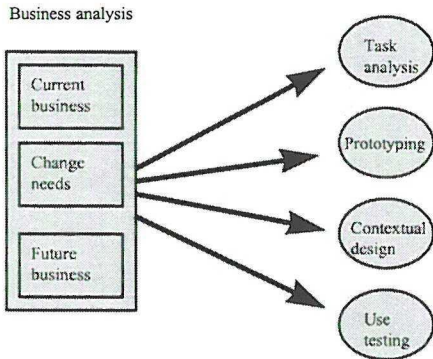


<b>Method:</b>	<b>Result:</b>
Business analysis.	Description of present business, prioritized change needs, model of proposed future business.
Task analysis.	Description of present work task knowledge, model of proposed task design.
Usability specification.	Knowledge of requirements for a system to be deemed usable, basis for use testing.
Heuristic evaluation.	Description of possible usability problems in user interface based on general usability principles.
Cognitive walkthrough, (cognitive jogthrough).	Description of possible usability problems based on user needs when carrying out one or more tasks.
Use of guidelines and styleguides.	A list of deviations from defined guidelines and styleguides.
Prototyping.	Concretized and validated user requirements, identification of additional user requirements. Identification and definition of user interface requirements and user interface design.

Contextual design.	Description of work performed by a user, suggestions for change of work, design of physical environment, computer system, and user interface.
Use testing.	Indications of possible usability problems during user performance of work tasks, using the computer system (or prototype).

#### *5.2.1 Role of Business Analysis*

Results from business analysis is important for task analysis, prototyping, contextual design, and use testing. For task analysis, a model of future business facilitates identification, analysis, and modeling of tasks. In prototyping, results from business analysis serve as a frame of reference in discussion of what services a computer system should deliver. Results from business analysis can also assist when studying the relationship between these services and those services not to be supported by a planned computer system. For contextual design, information about present business and possible design of future business, provide context when discussing how work should be designed. With use testing, a model of future business contributes to the evaluation of computer system usability from a business perspective. In summary, the greatest value of business analysis in relation to other methods is for background information, as related to business information. The result from business analysis serves as a foundation for the other methods. In Figure 11, below, the use of results from business analysis is illustrated.



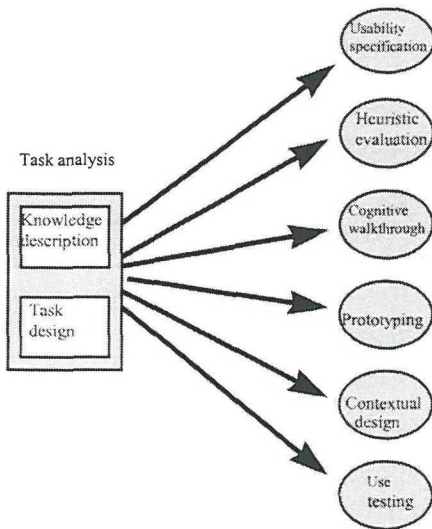
**Figure 11: Illustration of use of results from business analysis**

### *5.2.2 Role of Task Analysis*

Task analysis provides information of value in usability specification, heuristic evaluation, cognitive walkthrough (cognitive jogthrough), prototyping, contextual design and use testing. With usability specification, information about present and/or future work tasks is used to define usability goals and to choose tasks for inclusion in the usability specification. Heuristic evaluation incorporates information about tasks in its analysis (see, for example, Nielsen, 1993; 1994), therefore, task analysis is of value. The same is true for cognitive walkthrough (cognitive jogthrough), where task descriptions form the basis of the walkthrough to be carried out. Prototyping, uses for example information about tasks to evaluate relevance of “unstructured wishes”, that fail to relate to tasks to be supported by the computer system being developed. Task analysis also provides information concerning services to be included in prototyping. Information from task analysis is probably also of value in contextual design for increased understanding of tasks to be supported by a computer system.

Knowledge from task analysis can also help to ensure that the physical environment is designed in accordance with tasks allocated between a

computer system and user (business people). With use testing, a task description is of value when selecting tasks to be included in the use test. A task description can also be useful in deciding criteria for measurement. (If a usability specification has been developed, need for results from task analysis in conjunction with use testing is not as important, see below). In Figure 12, below, use of results from task analysis is illustrated.



**Figure 12: Illustration of use of results from task analysis**

### 5.2.3 Role of Usability Specification

Results from usability specification are closely related to use testing. Usability specification contains information about what to measure, how to measure, and expected results. Results from usability specification also contribute information during heuristic evaluation and cognitive walkthrough. In carrying out these methods, information from a usability specification can be used to formulate criteria useful for identifying possible usability problems.

#### *5.2.4 Roles of Heuristic Evaluation, Cognitive Walkthrough (Jogthrough), Use of Guidelines and Styleguides*

The result from heuristic evaluation, cognitive walkthrough (jogthrough) and use of guidelines and styleguides, do not directly relate to other methods. Results from these methods are not incorporated into analysis used by other methods. However, heuristic evaluation, cognitive walkthrough (jogthrough) and use of guidelines and styleguides can indirectly assist other methods. From these methods, it is possible to more clearly identify and understand critical parts of a computer system (especially the user interface). The methods are also important for discovery of unanswered questions needing further study, through, for example, prototyping and use testing. Heuristic evaluation, cognitive walkthrough (jogthrough) and use of guidelines and styleguides identify other kinds of usability problems, and can be regarded as supplementary to use testing (see, for example, Jeffries et al., 1991). However, experienced usability experts can identify need for prototyping and use testing, during, for example, heuristic evaluation. Probability of identifying need for further usability work increases when usability experts understand business and tasks to be supported. The value, importance and role of the methods will of course vary according to the level of experience of those working on development. The results from these methods can also unburden use testing. Less critical usability issues (e.g., low level problems in the user interface) can be studied before use testing is carried out. These methods can of course also be used to inspect prototypes. Thus results from these inspections are of immediate application in prototyping.

#### *5.2.5 Role of Prototyping*

Results from prototyping have direct implication in a number of other methods. With business analysis, simple prototypes can deliver valuable information about advantages and disadvantages of proposed business



design. In task analysis, prototyping can help to make effects of different task designs and task organization explicit. In contextual design, prototyping has been described as an integral part. Use testing, cognitive walkthrough (jogthrough), heuristic evaluation and use of guidelines and styleguides require prototypes if no system exists to be evaluated.

#### *5.2.6 Role of Contextual Design*

Results from contextual design can be of value during use testing. When more than the computer system (or prototype) is studied, information about the physical environment provides detail needed for design decisions.

#### *5.2.7 Role of Use Testing*

Results from use testing may be viewed as part of the prototyping process. Use testing is needed for conclusions concerning prototype performance. Use testing can also be part of business analysis. Use testing (in some form) of business models can provide important information on how a proposed business design influences users and/or business people possibility to carry out work. The same is true for task analysis.

### **5.3 Additional Methods Needed**

As is evident in the descriptions of results derived from usability work methods, the outcome is focused on business, work tasks, user requirements, user interface and use of computer system (or prototype). The user is an important member in, and supplier of information to, many of the methods.

However, user input and study is mainly from the perspective of business and task in which they participate. Little is mentioned about the user, and his or her characteristics as a human being. None of the methods explicitly mention user physical and psychological needs/abilities (although these needs and abilities are the foundation for cognitive walkthrough, heuristic evaluation etc.). Contextual design explicitly focus on user (customer) in connection with computer system development. However, the method seems to assume that user focus alone is enough to identify and consider user needs/abilities.

In my opinion, it is necessary to supplement the usability work "tool box" presented here with methods for identifying and describing the computer system user needs/abilities. Approaches in this direction is for example, user profiling (Andriole, 1989; 1996), cognitive (systems) engineering, (Andriole & Adelman, 1991; 1995; Hollnagel, Mancini & Woods, 1988; Woods, 1988; Woods & Roth, 1988), user modeling (Kelly & Colgan, 1992; Wilson, Johnson, Kelly, Cunningham & Markopoulos, 1993), and GOMS (Card et al. 1983; John, 1995; John & Kieras, 1996).

#### **5.4 Further Development of Methods for Usability Work**

As mentioned earlier, for the methods to be more fully adapted to the industrial system development situation, it is necessary to simplify and/or further develop some of them. One reason for this is that many of the methods (for example, business analysis, task analysis) require excessive resource investment (see, for example, the experiences mentioned in section 4.4). This may cause methods not to be used when time and funding for development and delivery of a computer system are limited. Experiences from system development also suggest that little time is dedicated to early system development activities (see, for example, Andriole, 1990; Davis, 1993).

This fact can cause an unwillingness on the part of project management to dedicate resources to usability work. Particularly if resource demands are high compared to perceived value for the project.

Work necessary to simplify and/or further develop different methods, is beyond the scope of the present work. To simplify and/or further develop a method it is necessary to consider the theory that is the basis for the method (see, for example, Johnson, Drake & Wilson, 1990, for a discussion of the value of a theoretical frame of reference when developing a method). If a method is further developed or simplified without considering theory, outcome from method use may be defective. Another reason is the necessity to practically evaluate a further developed or simplified method to verify that use will still result in the right outcome.

Briefly presented here are some actions needed to make methods described more natural parts of the system development process. The first action recommended is to simplify business analysis, task analysis and use testing. To adjust these methods so that they may be carried out in a "mini way", to support initial and more general attempts. This modification does not advocate always utilizing a "mini method." It is necessary to remember that simplified methods have limitations compared to original methods. Although some literature does indicate that simplified methods have value in system development (Nielsen, 1995; Rowley & Rhoades, 1992).

Second, it is important to illustrate the dynamics in business models (graphs, flow charts, etc.,) developed in business analysis. Many businesses are so complex that a static description is insufficient for users and/or business people to understand a business process (see, for example, Hughes, 1996).

Third, it is valuable to supplement use testing with environmental simulation (in some way), especially if a computer system is being developed to support complicated and extensive work situations. Sometimes, it is necessary to simulate how other business parts, users, and the external environment act and react, when users evaluate a computer system by carrying out work tasks. This simulation does not have to be performed by computers; use of 'Wizard of Oz' techniques (Maulsby et al., 1993) can in many cases be enough.

## 5.5 Practical Evaluation

The preliminary evaluation of the model and the methods described in section 4.4 is not sufficient. A more thorough and deep practical evaluation, to test and study advantages and disadvantages of the model and the methods, and to identify needs for new methods, is required. The most important actions to perform in such an evaluation are to:

- use the model and the methods throughout the system development process, from identification of need to operation and maintenance. The purpose being to study advantages and disadvantages in every system development activity, and to study how the methods work together. To study the process of transferring results from usability work in one system development activity to other system development activities will also provide useful information,
- study possible effects of usability work on a requirement set that is being developed. According to Andriole (1994), Brown, Earl and McDermid (1992), for example, the most serious deficiencies in present system development concern deficiencies in the requirement set (requirement specification). Important issues to study are for example; Does focus on usability work continuously during the system development process result in requirements changes being regarded as natural?; Can these changes be integrated in a natural way when



prototypes are developed?; Does a focus on usability work early in the system development process result in a more complete requirement set, earlier in the system development process?,

- identify reactions toward usability work by business people, users, system developers, and customers. If these groups of people do not consider usability work as important, usability work will probably not be a natural part of system development,
- study developers during practical work to acquire in-depth knowledge of their needs and demands,
- study use of usability work methods, to identify further development possibilities and needs.

## **5.6 The need for Computer Support in Usability Work**

The need for computer support, for example different kinds of CASE (Computer Aided Software Engineering) systems, in system development has been recognized for a long time (Brown, et al., 1992; Sommerville, 1996). In user interface development, adoption of UIMSs (User Interface Management Systems) or User Interface Tools has been advocated (Löwgren 1991; Myers, 1995). In usability work computer support can also be useful. According to Andriole (1996), tools (computer support) to implement methods, processes and principles into the system development process are necessary aids that reduce resource demands. Without tools, the methods, processes and principles tend to increase workload excessively. In my view, the same is applicable to usability work. Another reason for computer support is to support the difficult work of ensuring that user requirements identified early, are taken care of later in the system development process (Wilson, et al., 1993).



It is possible to divide computer support for usability work into two groups:

- computer support for a particular method,
- computer support to support the process of usability work.

The extent to which computer support can aid a particular method and/or the process of usability work can vary widely. It is possible to identify a range of support levels. For example, computer support that assists:

1. in documentation of the result derived from the application of a particular method,
2. in utilization of a particular method,
3. in carrying out usability work in a particular system development activity,
4. in transfer of results from usability work in one system development activity to usability work in subsequent system development activities,
5. integrated usability work during the entire system development process.

Greatest benefit occurs when there is computer support on the most comprehensive level (level 5), where usability work is supported in an integrated way during system development. Almost the same ideas can be found in I-CASE (Integrated CASE, see, for example, Cronholm, 1994). Computer support on lower levels is of course also useful. At present, only computer support on levels 1, and 2, seem to be available to assist usability work. A number of tools that support documentation of usability work exist, for example, word processors and drawing programs. There are also different kinds of tools (programs) appropriate for prototyping, for example, HyperCard, Visual Basic, Visual C++, etc., (see also the description of CAUSE tools in Nielsen, 1993). However, above mentioned computer support only assists in documentation and accomplishment in the sense that they are possible to use in documentation and performance. They do not support

documentation and performance through assisting tool users in, what to document, how to document, or what to prototype, how to prototype and how to evaluate the prototype.

There are also different kinds of tools developed, that support the system development process in a more traditional way, with no direct reference to usability work. In Chapter 6, these tools and a discussion of in what respect they support usability work are briefly presented.

## 6. TRADITIONAL COMPUTER SUPPORT IN SYSTEM DEVELOPMENT

### 6.1 Introduction

This chapter provides short descriptions of different kinds of computer support, or tools, developed to support the system development process. Described here are CASE systems and User Interface Tools. The description is followed by a brief discussion of CASE systems and User Interface Tools, from the perspective of usability work. More detailed descriptions of CASE systems and User Interface Tools may be found in McClure (1989), Sommerville (1996) and Myers (1993; 1995), respectively.

### 6.2 CASE Systems

Computer Aided Software Engineering (CASE) systems (in accordance with Sommerville, 1996, "systems" is used as a general term for all kinds of CASE technology: tool, workbench or environment) is a common term for tools aimed at supporting and bringing improved efficiency to the system development process. (Another common term is Computer Aided System Engineering tools, Eisner, 1987; Goldkuhl, 1991). According to Bubenko (1989), Cronholm (1994), Goldkuhl (1992) and Sommerville (1996), CASE systems often consist of the following building blocks:

- a textual and/or graphical editor to create models (for example, flow charts and/or object models),
- a repository, to store results from modeling,
- a function for verification and consistency checking of models,
- a function for transformation of descriptions from one level to another,
- a function for report generation,

- a function for import and export of data.

CASE systems can, according to Sommerville (1996), be divided in three different levels:

- CASE systems that support the production process,
- CASE systems that support project management,
- CASE systems that support development of CASE systems, META-CASE systems.

Only the two first levels are discussed here. Within these levels, it is possible to divide CASE systems into the main groups; Tools, Workbenches and Software Engineering Environments (Brown, et al., 1992; Sommerville, 1996). Software Engineering Environments are also called Integrated Project-Support Environments, IPSEs (Sharon & Bell, 1995), or I-CASE (Andriole, 1996; Cronholm, 1994).

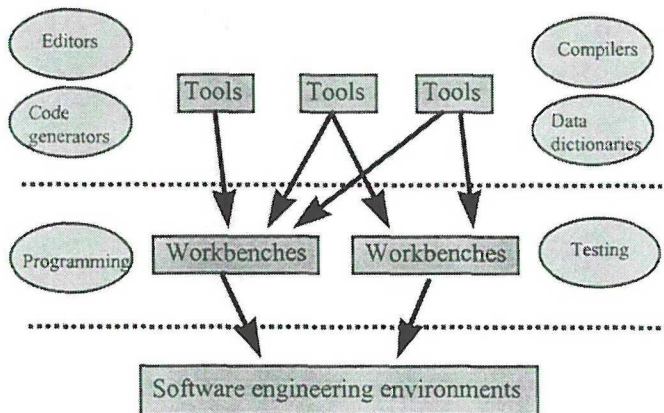
Tools support individual sub-activities within the system development process. Example of tools are, data dictionary tools, diagram drawing tools, prototyping tools, interactive debugging tools, documentation tools and tools for analyzing software code.

Workbenches can either support activities within the system development process (for example, software design, implementation and testing), or actions necessary during the entire system development process (for example, configuration management). Examples of workbenches are programming workbenches, analysis and design workbenches, testing workbenches, cross-development workbenches, configuration management workbenches, documentation workbenches and project management workbenches. Workbenches consist of a set of tools that are integrated at some level. (For a

detailed discussion of different types of integration, see, for example, Sharon & Bell, 1995; Sommerville, 1996; Wasserman, 1990).

The purpose of software engineering environments is to support the entire, or main part of, a system development process. Software engineering environments usually consist of a number of tools and workbenches, that are integrated to support the system development process. A typical combination of tools and workbenches might support software requirements analysis, software design and implementation & unit testing (in Sommerville, 1996, these activities are named analysis and design, and programming).

A simple description of the division of CASE systems, is presented in Figure 13, below.



**Figure 13: Different types of CASE systems and their relation**

CASE systems can also be divided in Upper-CASE and Lower-CASE. Upper-CASE systems are systems that support early system development activities, such as requirements definition (Cronholm, 1994), or software requirements



analysis (Sommerville, 1996). Lower-CASE systems support such later activities as software design and implementation & unit testing (Cronholm, 1994; Sommerville, 1996). Most Upper-CASE are tools, while Lower-CASE can be tools, workbenches or software engineering environments.

There are CASE systems for almost every system development activity. However, the support delivered are at different levels. For the system development activities software design, and implementation & unit testing, there is a range of available support systems. These systems deliver a high degree of support in areas such as data modeling, software design and implementation. For workbenches, the support delivered may also include transformation of design descriptions to code. For system development activities early and late in the process, for example, identification of need and operation & maintenance, there is not so much support in CASE systems. Support is mainly in documentation management and text editing systems. Exceptions are Upper-CASE systems mentioned in relation to description of business analysis (sub-section 3.2.1, also discussed in Goldkuhl, 1991). Upper-CASE systems support, for example, documentation of business analysis results and consistency checking of developed models.

According to Fisher (1988), the advantages with CASE systems are mainly that they contribute to:

- development of complete requirements specifications,
- development of complete design specifications,
- development of timely design specifications,
- decreased time needed for implementation,
- more effective development and maintenance of code.

Following are short descriptions of some of the workbenches mentioned above.

### *6.2.1 Analysis and Design Workbenches*

According to Sommerville (1996), analysis and design workbenches are collections of tools that support analysis and design activities in a system development process. These workbenches are normally used to support development and analysis of models, such as data flow models, ER-models, or object models. Workbenches for analysis and design usually support a specific analysis and design method, like structured analysis and design, or object-oriented analysis and design. However, there are also workbenches that support a number of methods.

Analysis and design workbenches can, according to Sommerville (1996), consist of following components:

- a diagram editor to develop data flow diagrams, structured diagrams, ER-graphs, object models and other illustrations of software structure,
- tools for analyzing a developed design to identify errors, deficiencies and inconsistencies,
- a repository to store results from modeling,
- a query language a developer can use to search for already developed designs, and connected information, in the repository,
- a data dictionary containing information about entities used in a design,
- a tool for development and generation of design documentation,
- a tool for import and export of design information to, and from, other development tools,
- code generators that automatically generate code and code skeletons from the design.

Analysis and design workbenches are developed for a number of analysis and design methods, for example, HOOD (Robinson, 1992), Objectory (Jacobson,

Christenson, Jonsson & Övergaard 1992), Structured Systems Analysis and Design Methodology, SSADM (Ashworth & Goodland, 1990), Jackson System Development, JSD (Jackson, 1983).

### *6.2.2 Programming Workbenches*

A programming workbench is, according to Sommerville (1996), a collection of tools aimed to support the software development process. This type of workbench was the first type of application for CASE systems. Support consisted of software development tools, such as compilers, editors and debuggers.

A programming workbench can, according to Sommerville (1996), for example, include the following tools:

- a compiler to transform a source program to object code, to create an abstract syntax tree (AST) and a symbol table,
- a structure editor to edit syntactic representations of the program (AST),
- a linker to link object code program and already compiled components,
- a loader to load executable programs into memory before execution,
- a cross-reference function to produce a cross-reference list that lists where all program names are declared and used,
- a printer function to print source program,
- a static analyzer to be used in analyzing source code to identify deficiencies,
- a dynamic analyzer that, for example, gathers information concerning execution of different parts of source code, and statistics of processor usage,
- an interactive debugger for developers to check order of execution and to supervise the condition of a program during execution.

One example of a programming workbench is, according to Brown et al. (1992), UNIX PWB (Programmer's WorkBench). According to Sommerville (1996), many language compilers (C++, Pascal, Lisp, Smalltalk) including additional tool support, are also examples of programming workbenches.

### 6.2.3 *Testing Workbenches*

Sommerville (1996), defines testing workbenches as a collection of tools to test and detect errors in software. Most testing workbenches are built by purchasing a testing workbench and adapting it to organizational demands. Because of this internal development (usually not performed when analysis and design, or programming workbenches are purchased), it is difficult to formulate a common description of testing workbenches. According to Sommerville (1996), a testing workbench may consist of following tools:

- a test manager to manage and supervise performance of tests,
- a test data generator to develop test data,
- an oracle to develop predictions of expected test results. (Results can be results from earlier software versions, or theoretically generated results),
- a function that compares obtained and expected test results, and reports differences,
- a report generator to define and develop test reports,
- a dynamic analyzer similar to that described in programming workbenches,
- simulators to simulate, for example, target system where software is to be executed on, the user interface, input and output to software.

Examples of testing workbenches are WinRunner and XRunner (Mercury Interactive Corporation, 1993a, b, c).



#### *6.2.4 Conclusions*

Use of CASE systems has not resulted in the productivity improvement in system development predicted. According to Sommerville (1996), this is because CASE systems:

- do not take care of the main problems in system development, complexities in the product and in the development process,
- are not integrated at the level required; there are no CASE systems that support an entire system development process. It is also impossible to integrate present CASE systems to support an entire process,
- are complicated and difficult to learn. It is also difficult to adjust CASE systems to another system development method than the one for which the system was originally developed.

Another factor possibly influencing the absence of productivity improvement is that few CASE systems support early system development activities. Exceptions are the kind of computer support mentioned in conjunction with business analysis. When Upper-CASE systems are defined to also include this kind of tools (see, for example, Cronholm, 1994; Goldkuhl 1991), the situation is somewhat more positive. However, problems with integration are still present, as these tools are not integrated with CASE systems for analysis and design (Sommerville, 1996).

### **6.3 User Interface Tools**

User Interface Tools is a collective term for tools developed to support one or more of the design, implementation and evaluation of user interface activities (Myers, 1995). Within this term are User Interface Management Systems (UIMSs), Toolkits, User Interface Development Environments, Interface Builders, Interface Development Tools and Application Frameworks. User



Interface Tools can also be seen as a sub-entity of CASE systems (see, for example, Fisher, 1988; Sommerville, 1996).

According to Löwgren (1991), Myers (1989; 1995) and Shneiderman (1992), possible advantages in using User Interface Tools during user interface development are, for example, the following:

- the user interface is being more usable, because:
  1. user interface prototypes can be developed and evaluated prior to the development of an underlying application,
  2. user interface is more consistent because tools used decrease possible design solutions,
  3. it is easier to adjust user interface to different user groups, since user interface can be changed without influencing the underlying application,
  4. other professionals than programmers can actively participate in development. For example, human factors specialists and graphic designers,
- the user interface is easier to develop and maintain, because:
  1. the user interface code is separated from the application code,
  2. it is easier to reuse user interface elements,
  3. less user interface code has to be developed, since much of the basic code for the user interface elements is delivered by the tools.

Below, is a brief description of some of the User Interface Tools mentioned above. For a more detailed description, see, for example, Myers (1993; 1995).

### *6.3.1 Toolkits*

A toolkit is a tool box, or a library of widgets, containing for example menus, buttons, scroll-lists and dialog boxes. To implement a user interface using a toolkit, usually programming expertise is required. An advantage in using a toolkit is that the user interface developed will have a similar look and feel, as other user interfaces developed using the same toolkit. Disadvantages with toolkits are the restricted support for the design situation. Within the constraints provided by a toolkit's pre-defined widget set, it is possible to develop any user interface. Examples of toolkits are Motif toolkit, Microsoft Windows toolkit, and Macintosh toolkit.

### *6.3.2 Interface Builders*

Interface builders are tools that support a user interface developer in creating for example, menus, dialog boxes and buttons. User interface elements are selected from a library or a tool box containing widgets, and then placed at appropriate places in the user interface under development. The behavior of user interface elements is defined in attribute forms (Myers, 1995). Interface builders make development of traditional user interfaces easier and faster. However, design possibilities are limited to toolkit options in the interface builder. Interface builders do not guide the design of user interface, as developers can freely decide where to place user interface elements in the user interface to be developed. Examples of interface builders are, according to Myers (1995), Windowmaker for Microsoft, and UIMX for X Windows and Motif.

### 6.3.3 User Interface Management Systems

User Interface Management Systems (UIMSs) are tools that support development (design and implementation) and execution (dialog management) of user interfaces (Löwgren, 1991; Myers, 1995). UIMSs usually consists of a development module and a run-time module. The development module normally contains a graphical editor, and/or a specially developed language, that can be used by a developer to create user interface elements, define their behavior, and to define dialog between user and application. The run-time module usually contains a toolkit utilized to present user interface elements, and a dialog manager. The dialog manager administer user input and also ensures that right actions are performed by an application. Very few UIMSs have an evaluation module to support evaluation of user interfaces.

In my opinion, UIMSs can be divided into a number of groups, depending on the approach selected to support development and management of user interface (alternative groupings can be found in Bergsten, Bern, Kool & Wingstedt, 1993; Myers, 1995). The groups identified are; Traditional UIMSs; Direct Manipulation UIMSs; Automatic generation UIMSs; and Application Frameworks (the last group is described in a separate sub-section, below). These groupings need not be seen as absolute; some UIMSs can be assigned to more than one group. The primary reason for this grouping is to discuss in what ways different approaches influence user interface development.

Traditional UIMSs, are those supporting development and management of user interfaces, by providing developers access to a high-level language. Developers specify user interface elements using the high-level language. Example of languages are: state transition networks; event languages; and declarative languages. The program created is interpreted by a management module in the UIMS that define the result of actions from user and operations

from application software. Examples of UIMSs of this kind are, according to Myers (1995), VAPS from Virtual Prototypes Inc., and Open Dialogue from Hewlett Packard.

Direct manipulation UIMSs, are UIMSs that support development and management of user interfaces, by giving the developer access to a graphical specification language. With this language, a user interface is created by “drawing” a user interface using widgets and/or other user interface elements. Examples of this kind of UIMSs are, according to Myers (1995), HyperCard by Apple, and Toolbook from Asymetrix Corp.

Automatic generation UIMSs, are those that support development and management of user interfaces, by automatically generating a user interface from a high-level description. The high-level description is created by a user interface developer and can, for example, consist of a model of included interaction objects, what they should do, and how these objects relates to each other and to the underlying application. A model of this kind is sometimes called a description of the user interface on the semantic level (Löwgren, 1991). Example of this kind of UIMS is UIDE (Sukaviriya, Foley & Griffith, 1993).

According to Löwgren (1991), UIMSs have not been as successful in development of user interfaces as predicted (see also Myers, 1995). UIMSs are also not used as much as predicted in conjunction with industrial system development. Reasons for this are, according to Löwgren (1991), Myers (1995) and Reiterer (1994), primarily that UIMSs:

- usually are difficult to integrate into a system development environment,
- are not adapted to what developers need.



Reasons why UIMSs are difficult to integrate include factors such as software language incompatibility and limited usability. Many of the UIMSs (at least the ones developed from a research perspective) utilize development environments such as LISP, which make them difficult to integrate with other development environments. Other UIMSs are especially developed for certain platforms, for example UNIX, and are difficult to integrate with other software like Microsoft Windows.

That UIMSs are not adapted to what developers need is, according to Löwgren (1991), primarily due to the fact that development of UIMSs has not been influenced by developer needs. Influence has rather been from a technical point of view. According to Löwgren (1991), an understanding of the process of developing a user interface is required to influence the development of UIMSs.

Other influential factors are that many UIMSs (at least the ones called traditional UIMSs) require profound programming expertise, as well as deep knowledge of the specific tool. They are difficult, if not impossible, for use by human factors specialists and graphic designers who lack such knowledge. Probably, they are not interested in, and have not enough time for, learning a complex language and an advanced UIMS environment. Automatic generation UIMSs are at present time not mature enough (see, for example, Myers, 1995), to be an alternative in industrial system development.

#### *6.3.4 Application Frameworks*

Application frameworks are especially developed UIMSs for development of user interfaces for a specific type of platform or a specific kind of application. Examples of applications frameworks are, according to Myers (1995), MacApp



(described in Schmucker, 1986) and Unidraw (described in Vlissides & Linton, 1990). In my opinion, also UIMSs for development of data bases, for example 4GL tools, can be called application frameworks.

Another type of application frameworks are the ones, developed out of the concept of small software modules. These can be put together to create special application programs such as document management programs (Paley, Hansen, Kazar, Sherman, Wadlow, Neuendorffer, Stern, Bader & Peters, 1988, referenced in Myers, 1995). In these application frameworks, the cut and paste technique is further developed. In a specific document it is possible to incorporate tools such as a drawing program and/or a calculation program. Using this approach it is possible to use functionality from these programs without opening the separate programs. Examples on this approach can be found in OLE from Microsoft and OpenDoc from Apple.

#### *6.3.5 Conclusions*

User interface tools primarily support implementation and functioning (management) of user interfaces. If we look at user interface tools from the perspective of the activities in the system development process, as described earlier in the thesis, it is possible to make following conclusions. User interface tools support:

- part of software design,
- part of implementation and unit testing,
- part of operation and maintenance.

User interface tools do not support:

- identification of need, identification of user requirements (requirements definition), overall system design, identification of software requirements, and integration and testing.

User interface tools do not comprehensively support software design. They do support design of specific user interface elements. Not supported is design of user interface as a whole. With respect to implementation and unit testing, user interface tools support implementation, and in exceptional cases testing. The MIKE system (Olsen & Halversen, 1988), the Framer system (Fischer & Lemke, 1988; Fischer & Lemke, 1989; Fischer, Lemke, Mastaglio & Morch, 1990), the KRI system (Löwgren & Nordqvist, 1990; 1992) and TUNE (Nordqvist 1996), illustrate how to support evaluation of user interfaces. Only the MIKE system can be called a user interface tool. All others are modules that can possibly be used together with a user interface tool. In operation and maintenance, many user interface tools can support further development of a user interface carried out in connection with the different kinds of maintenance mentioned by Sommerville (1992). This is especially true for user interface tools making it possible to change a user interface without need to change the application code.

User interface tools do not support early and late activities in the system development process, as none has been developed for this purpose. (That user interface tools are sometimes used as prototyping tools during early activities in system development does not necessarily mean that support is provided). User interface tools have been developed to support design, implementation and evaluation of user interfaces.

Development of user interfaces, and utilization of user interface tools are often regarded as a stand-alone activity. This has caused minimal effort to

integrate those tools with other available tools, for example different kinds of CASE systems. Although efforts have been made (see, for example, Johnson et al., 1990; Löwgren 1991; Myers 1995, for a description of these approaches), much work remains before user interface tools are integrated with other system development tools.

Another factor that influences the value of user interface tools in system development, particularly user interface development, is that they are not adapted to user interface developer needs. According to Löwgren (1991) UIMSs are not adapted to the requirements of developers. For user interface tools in general, Myers and Rosson (1992); Myers (1995); Rosson, Maass, and Kellogg (1988), also mention deficiencies in this respect. For example, Myers and Rosson (1992) consider the task to investigate user needs, and user requirements on the user interface to be developed, as the main difficulties in conjunction with development of user interfaces. Myers (1995) point to the need for tools that support task analysis. Rosson et al., (1988) advocate that user interface tools support creation of task scenarios and discussion of different design ideas. Existing user interface tools do not support these activities to the needed level of detail.

#### **6.4 Traditional Computer Support in System Development and its Relevance for Usability Work**

Above mentioned computer support tools for the system development process has, despite the deficiencies described, contributed to more efficient industrial system development (Myers, 1995; Sommerville, 1996). However, the question remains as to the extent that these tools support a system development process that focuses on users and use of the system. In other words, do these tools support usability work, and if so, how? Below, an initial analysis discusses the main points of this question.

In looking at the conclusions made for CASE systems and user interface tools, from the perspective of usability work in industrial system development, it is apparent that traditional computer support for system development seldom supports usability work. Following is the basis for this conclusion. The point of departure is the simple "model" of support levels presented in section 5.6. From this, CASE systems and user interface tools:

- do not support documentation of results of most methods for usability work,
- partially support utilization of a single method,
- do not support accomplishment of all aspects of usability work in a specific system development activity,
- partially support transfer of results from usability work in one system development activity to subsequent system development activities,
- do not support integrated usability work throughout the entire system development process.

When comparing these conclusions, with the model and the methods advocated in earlier chapters, it is possible to concretize them further.

Traditional computer support for system development supports documentation of results from business analysis (see, Upper-CASE, Cronholm, 1994), and prototyping (user interface tools). Also supported is the utilization of some methods (business analysis, prototyping) in that some CASE systems (Upper-CASE) and user interface tools can be used in these activities. Not supported is accomplishment of all aspects of usability work in a specific system development activity. The tools do not give any answers to issues as, what methods to use, how to combine them, and how results from one method can contribute to another method. Traditional computer support for system development partially supports transfer of results from usability work in one system development activity to subsequent activities, as user



interface tools can be used to further develop the prototypes. They do not support integrated usability work, as basic integration between different tools is missing (Sommerville, 1996).

Using the usability work model, advocated in this thesis, it is also possible to conclude the following. Traditional computer support for system development does not support the evaluative aspects of usability work to any great extent. CASE systems or user interface tools do not contain significant functionality for evaluation of user interface designs (see, for example, Myers, 1995; Löwgren, 1991) or other usability aspects.

To contribute to the elimination of these deficiencies in computer support for usability work, a number of studies are presented in Chapter 7. These studies present different simple tools that support evaluation. Three of the studies focus on the issue of evaluating user interfaces in conjunction with design and implementation. The fourth study provides a simple illustration of how to support evaluation that user requirements are in a computer system being developed.



## **7. SUMMARY OF THE STUDIES**

### **7.1 Introduction**

The studies in this thesis have the general and common goal of studying the possibility to support development of computer systems, that fulfills user requirements and are usable, with different types of basic computer support. While the studies focus on low-level problems (for example, consistency in user interfaces, design of user interface elements), computer support for these can hopefully contribute to increased overall usability in computer systems.

### **7.2 Study 1: A Knowledge-Based Tool for User Interface Evaluation and its integration in a UIMS**

This study had three objectives. The first was to develop a prototype tool for knowledge-based evaluation of user interfaces (the KRI system). Design goals for this prototype called for it to have capacity to illustrate possibility to use knowledge-based techniques (specifically the critiquing approach). In that way making it possible to utilize expert knowledge to support evaluation of user interfaces. Secondly, this study also discussed how a tool of this kind could be integrated in a User Interface Management System (UIMS). Thirdly, the study also tried to illustrate the possibility to support evaluation on more than the presentation level.

Justification for the study was the identified need to supplement UIMSs with the capacity to evaluate user interfaces (Myers, 1989; Olsen, Green, Lantz, Schulert & Sibert, 1987).

### *7.2.1 A Knowledge-Based Tool for Evaluation of User Interfaces, the KRI System*

The following are the KRI system's primary components:

- a knowledge-base containing evaluation knowledge,
- an inference mechanism,
- a database containing guidelines concerning user interface design,
- a taxonomy of user interface aspects (elements).

The knowledge-base consists of a set of rules. These were drawn up after studying user interface experts evaluating user interfaces and interpreting applicable guidelines. The inference mechanism is based on forward chaining where the rules are used to identify and report design flaws. The database contains guidelines from Smith and Mosier (1986), used as a reference for possible comments concerning deficiencies in user interface design. The knowledge-base and the database emphasize knowledge relative to presentation and syntax levels of the user interface.

The taxonomy had two functions. First, to give users of the KRI system the choices for the evaluation of user interface elements. The taxonomy is presented as a graph in the user interface of KRI to facilitate evaluation choices. Secondly, the taxonomy serves as a foundation for structuring the knowledge-base.

Evaluation of a user interface using the KRI system was largely performed in the following manner:

1. the developer developed a design proposal (user interface),
2. the representation (presentation and syntax) of the design proposal was loaded into the KRI system,

3. using the graph, the developer selected in an interactive way what user interface elements to evaluate,
4. with these selected elements as criteria, the KRI system inspected the user interface representation and executed the applicable rules,
5. when a user interface element did not follow a rule in the knowledge-base, it was noted as a comment in the result file,
6. when the KRI system had completed the evaluation, the developer studied the result file (comments), referenced guidelines and change suggestions.

#### Practical use of KRI:

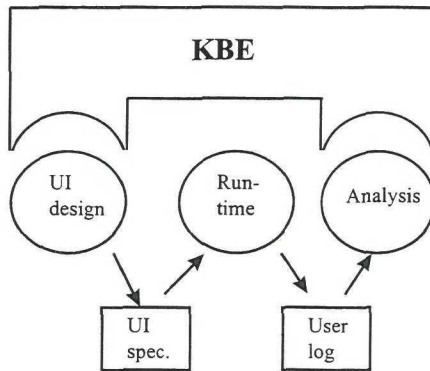
The KRI-tool was used to evaluate an application consisting of three separate tables for entering data and six pull-down menus. The necessary activation of tables before data entry could be performed through use of specially dedicated functions keys.

In evaluation of the function keys a comment concerning the mismatch between presentation order of the tables and the implicit order of the functions keys was generated. The reason for this comment was that the tables were activated with function key 3 for the top table, 1 for the middle table and 2 for the bottom table.

Practical use of the KRI system demonstrated that knowledge-based technique can be used to support evaluation of user interfaces. However, when user interface experts studied the comments generated, a number of comments were judged as trivial, or that they failed to address semantic (meaning), user, or task (pragmatic) issues. The reason for this limitation in the KRI system was that the user interface representation used did not handle these issues.

### 7.2.2 Integration with a UIMS

Systems such as the KRI system must be integrated with a UIMS to better support evaluation during user interface design. Using the Seeheim model (Tanner & Buxton, 1985) as a basis, an extended model supplemented with a knowledge-based evaluation module (KBE) was developed, see Figure 14 below.



**Figure 14: An extension of the Seeheim model**

This KBE module is intended to support both design of a user interface (ui) and evaluation (analysis) of registrations from use of a user interface. During design of a user interface, the KBE module can be used to evaluate design proposals. This is achieved by integrating the evaluation functionality in the KRI system with a commercially available UIMS, TeleUSE (Telesoft, 1989). Based on the representation (X widgets) in TeleUSE it is possible to evaluate the design proposal with respect to presentation aspects.

By utilizing the time stamped protocol (interaction log) generated by TeleUSE during use of a user interface, the knowledge in the KBE module can be used

to evaluate user interaction with the system. Although this log only delivers information about the lexical (presentation) level, it is possible to have information about, how often menu alternatives etc., are chosen, usual sequences, errors in user interface handling and help requests, for example. In this way it is possible to supplement the low-level evaluation performed at design-time with information related to user and task issues. From this run-time evaluation it is possible to obtain evaluation information usually only attainable from high-level representations.

### *7.2.3 Conclusions*

The results of the study indicated the possibility to utilize knowledge-based techniques (and in that way expert knowledge) to support design and evaluation of user interfaces. The study also principally discussed, how the evaluation functionality in the KRI system could be integrated with a UIMS. With this integration, developers have access to evaluation support for the presentation aspects of user interface elements in conjunction with design. Through use of the knowledge-base to analyze the interaction log it is also possible to obtain information on deficiencies in user interface design related to user interaction with the computer system.

## **7.3 Study 2: Knowledge-Based Evaluation as Design Support for Graphical User Interfaces**

The aim of this study was to further investigate ways to make human factors knowledge available when a User Interface Management System (UIMS) is used to develop user interfaces. This study focused on three issues mentioned in study 1. First, to investigate the possibility to use traditional knowledge sources such as guidelines and styleguides in computer support for user



interface evaluation. Second, to augment a UIMS with this kind of knowledge-base. Third, to further explore performance of run-time evaluations during development of user interfaces and in that way support evaluation of user interface aspects related to user and task.

In the study, a prototype knowledge-based tool (KRI/AG) containing knowledge from guideline and styleguide collections was developed. This tool was integrated with a UIMS, to extend the design and implementation environment with an evaluation module. The knowledge-based tool can be used at the convenience of a developer to evaluate design proposals developed in the UIMS. With this approach the developer can have information concerning further development of the design as needed. In other words, formative evaluation of user interface is supported.

### 7.3.1 The KRI/AG System

The KRI/AG system consists primarily of a parser and a knowledge-base, see Figure 15 below.

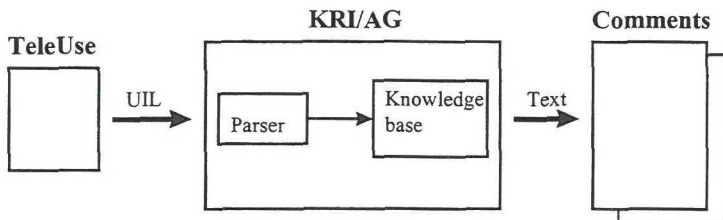


Figure 15: Overall architecture of KRI/AG

The parser is used to translate the representation (UIL) of the user interface, created in the UIMS tool (TeleUSE) to a representation understood by

KRI/AG. The representation contains information concerning static parts of the user interface, for example, buttons, menus and forms.

The knowledge-base is used to evaluate the user interface representation and to generate possible comments about design deficiencies. The knowledge-base contain rules generated from guideline and styleguide documents (for example, Brown, 1988; Open Software Foundation, 1988; Smith & Mosier, 1986). In KRI/AG, there are rules concerning graphic design, menu design, menu dialogue and other dialogue.

#### Practical use of KRI/AG:

KRI/AG was used to evaluate a user interface consisting of a map window, a number of tools for manipulation of overlay symbols (military units, etc.,) in the map, a number of option menus for inspection or change of symbol attributes and two pull-down menus containing global commands. The evaluation resulted in a number of comments. An example of a generated comment is: "There is no **Help** menu in the menu bar. Every application should have a **Help** menu. The recommended standard menus in the menu bar are File, Edit, View, Options and Help, in that order. (Motif Style Guide p. 7-42)."

Practical use of KRI/AG to evaluate a user interface showed that guidelines and styleguides (in this case, Motif) can be used to automatically evaluate a UIMS-developed user interface. However, analysis of these evaluations revealed that most of the design knowledge was useful only when the actual use situation was considered.

### Support issues:

Two important issues related to development of support systems for user interface design were identified during development and evaluation of KRI/AG. The first related to how developers of user interfaces shall be supported. The second concerned the level of evaluation appropriate for a design support tool.

Concerning the first issue, a number of authors (see, for example, Tetzlaff & Schwartz, 1991; Lemke & Fischer, 1990) have advocated the use of toolkits, good design examples and user interface skeletons rather than guideline and styleguide documents. However, practice has shown that despite this kind of support it is still possible to develop user interfaces that violate guidelines as well as styleguides. Possible deviations from guidelines and styleguides are deficiencies in consistency within and between applications. From the preliminary results obtained in this study, it is possible to believe that the KRI/AG system can support designers working on design issues related to consistency in user interface.

Concerning level of evaluation at which a design support tool is most effective, this study, and study 1, showed that evaluation at the task level provides the best data. Evaluation at the task level can be performed in two ways. One is to use a design representation where information about user tasks as well as domain knowledge are represented (see, for example, Foley, Kim, Kovacevic & Murray, 1989). Another is to collect and analyze logs (registrations) from real use of a user interface. In the study the second way was discussed, here called run time evaluation (RTE). The reason for this was that; this method was regarded as more compatible with commercially available design tools; there was no additional complexity in the design situation (rich and complex representations are harder to create and

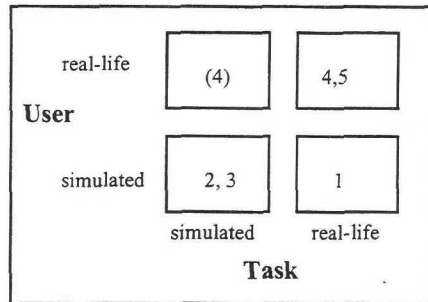
interpret); and it did not depend on a priori assumptions about the use situation as does the first method.

An RTE function to collect and analyze logs can be integrated with KRI/AG and TeleUSE. With this addition to KRI/AG, it is possible to evaluate logs generated by TeleUSE in conjunction with practical use of a user interface. Since TeleUSE utilizes X events and a specially developed language and event mechanism to manage events (called D language), it is only possible to collect logs on low-level aspects. Examples of data that may be collected are: keyboard input, button presses, mouse position. Utilizing a combination of knowledge-based and algorithmic techniques, logs may be analyzed and comments generated on deficiencies in a user interface not possible to assess in design time evaluation. Some of the user interface properties that might be evaluated with the RTE functionality are:

1. long sequences for common operations,
2. change of interaction technique within a task,
3. inconsistency in manipulation syntax (object-command or command-object),
4. detection of errors in handling of user interface, and use of help,
5. identification of accelerators for most common user actions.

Also a conceptual framework to relate the idea of runtime evaluation (RTE) to software development was developed, see Figure 16 below. According to this framework a task can be simulated or real-life. Simulated tasks can be defined from requirements specifications or they can consist of general interaction with the user interface. Simulated tasks can be tested in the development environment. Real-life tasks must be the real tasks the system is intended to support and they have to be carried out in the real environment. Users can also be simulated or real-life. A simulated user can be someone in the development team or a person chosen at random willing to pretend being the

intended user of the system. A real-life user is one of the users for whom the system is intended.



**Figure 16: The space of RTE and properties possible to evaluate**

In Figure 16, the properties possible to evaluate using RTE mentioned above are inserted in the evaluation space (number in parentheses means the property can be evaluated to some extent). According to the framework, it is possible to evaluate, for example, properties 2 and 3 with simulated users and simulated tasks, while properties 4 and 5 requires real-life users and tasks. From this it was concluded that simulated users and/or simulated tasks can be used to evaluate some user interface properties. Thus making it possible to carry out evaluation early in the system development process and to minor costs.

### 7.3.2 Conclusions

The study showed how human factors knowledge, in the form of guidelines and styleguides, can be included into computer support to evaluate a UIMS-produced user interface design. This kind of tool can probably reduce the number of design flaws related to user interface consistency.



The study also indicated that to address deficiencies in user interface associated with user and task the use situation must also be considered. As a possible solution to this problem a function for run time evaluation (RTE) was suggested, together with examples of user interface aspects possible to evaluate utilizing the RTE function.

Finally, a framework for relating evaluation of user interface properties to different phases in the system development process was presented. This framework illustrated that some user interface aspects could be evaluated using simulated users and tasks, and in that way be included early in the system development process.

#### **7.4 Study 3: TUNE: A Tool for User Interface Evaluation**

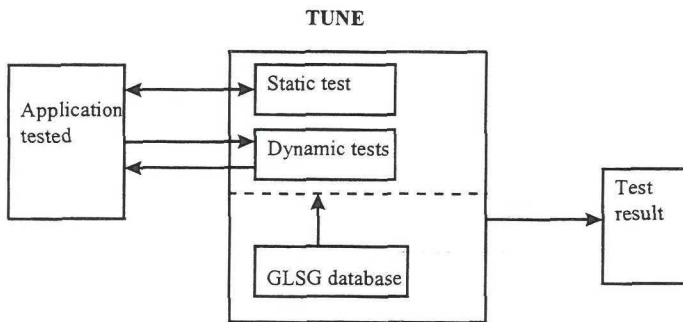
The purpose of this study was to develop a prototype tool (TUNE), that facilitated inclusion of human factors knowledge, in the form of guidelines and styleguides (GLSG), in development of user interfaces. The reason for this was the increased awareness of the difficulty to use present GLSG documents in conjunction with development of user interfaces (de Souza & Bevan, 1990; Mosier & Smith, 1986; Tetzlaff & Schwartz, 1991). This difficulty has resulted in that developers are reluctant to use this type of documents (Smith & Mosier, 1984; Thovtrup & Nielsen, 1991). A number of methods and computer support tools have been developed to overcome this barrier (Study 1 and 2 in this thesis; Nielsen & Molich, 1990; Perlman, 1989a, b).

Due to approaches often used in industrial system development, use of these methods and tools may not be practical. For example, in many system development projects UIMSs are not used and so the UIMS solutions in Study 1 and 2 cannot be adapted. Therefore, a prototype for a simpler computer

support tool (TUNE) was developed. TUNE was designed to illustrate how to facilitate the usually time consuming and laborious process of evaluating whether a user interface complies with GLSGs (specifically, *The Windows Interface: An Application Design Guide*, Microsoft, 1993). TUNE was tested by evaluating selected applications.

#### 7.4.1 TUNE

TUNE mainly consists of test programs for static and dynamic tests (implemented in C++) and a GLSG database, see Figure 17 below.



**Figure 17: Overall architecture of TUNE**

Static tests are utilized to evaluate the existence and appearance user interface elements. Examples of static tests are:

- existence of menus and menu items,
- appearance of mnemonics and short-cuts.

Dynamic tests are utilized to evaluate if user interface elements behave as specified in the GLSG being used. Examples of dynamic tests are:

- function of menu items, mnemonics, short-cuts and buttons,

- presentation of dialog boxes when menu items followed by three dots are selected.

The rules for accomplishing the tests are in the GLSG database or the test programs. In the GLSG database are rules that can be chosen, depending on the specific user interface elements in the application being evaluated. In the test programs are rules considered as generic for all applications. Utilizing the rules, TUNE evaluates a user interface by inspecting the information about user interface elements in an application and checking that static properties are as specified in GLSGs. For a GLSG for dynamic behavior, TUNE activates the user interface element and checks that the behavior is as specified in GLSGs. Deviations from GLSGs are written to a result file that may be inspected after an evaluation.

#### Practical use of TUNE:

TUNE were used to evaluate three applications, and the results are summarized as follows. A number of deviations from GLSGs were identified. Most of the deviations were related to the fact that the developer had forgotten some GLSGs. A number of design flaws were also repeated in the user interface. Developers had objections to evaluation results on very few occasions, in those cases they preferred other labels for menu items than the ones defined in GLSGs.

TUNE was also evaluated against a number of pre-defined goals. These goals were; reduction of time needed for evaluation of GLSG compliance; support the task of evaluating all user interface elements; enhance consistency in user interfaces; and support iterative design of user interfaces. The study to investigate if these goals were realized was performed by comparing TUNE testing of the three applications and manual evaluation by three usability

experts. The results from the evaluation are described briefly below. (Note, results from manual evaluation are presented as mean values).

Reduction of time needed for evaluation of GLSG compliance:

In Figure 18, below, time used (min) for manual and TUNE evaluation is depicted.

	A	B	C
Manual	115	21	135
TUNE	20	4	27

**Figure 18: Time used (min) for manual and TUNE evaluation for three applications**

As illustrated in Figure 18, time used in TUNE evaluation is about 20% of time used in manual evaluation. From this it is possible to conclude that TUNE reduces time needed for evaluation of GLSG compliance.

Support the task of evaluating all user interface elements:

Figure 19, below, illustrate number of evaluated user interface (ui) elements for TUNE and manual evaluation compared to total number of ui elements for three applications.

	A	B	C
Manual	231	38	290
TUNE	252	36	305
Total number	255	39	308

**Figure 19: Number of evaluated ui elements together with total number of ui elements**

The results in Figure 19 indicates that TUNE evaluation result in that more ui elements are evaluated compared to manual evaluation (at least for application A and B). The reason for the discrepancy between TUNE evaluation and total number is that TUNE did not evaluate the Exit menu item (including mnemonic and shortcut). Figure 19 also illustrate that the difference between TUNE and manual evaluation is greater when the application is more complex (more ui elements).

Enhance consistency in user interfaces:

Figure 20, below, illustrates number of recognized deviations for manual and TUNE evaluation for three applications. The basis for using number of deviations as an indication of enhanced consistency was our hypothesis that the more deviations recognized (and corrected) the more consistent user interface.

	A	B	C
Manual	69 (222)	33 (73)	58 (97)
TUNE	85	42	83

**Figure 20: Number of recognized deviations in manual and TUNE evaluation for the three applications**



From Figure 20 it is possible to conclude that TUNE evaluation result in that more deviations are recognized. However, this is only true if only implemented GLSGs are considered. If all GLSGs and the usability experts expertise is also included, manual evaluation result in detection of more deviations (numbers in parentheses).

#### Support iterative design of user interfaces:

Since TUNE can be used to evaluate user interface designs in progress it is possible to use TUNE in iterative design. Also interviews with designers who have used TUNE indicated that they experience TUNE as a support tool in iterative design.

#### *7.4.2 Conclusions*

Preliminary conclusions from utilizing TUNE can be summarized as follows:

- evaluation using TUNE is less time consuming than manual evaluation,
- evaluation using TUNE causes more user interface elements to be evaluated, especially for complex applications,
- TUNE identifies more deviations compared to manual evaluation, within its scope of GLSG coverage,
- TUNE supports iterative design, as a developer can use it for personal support when evaluating a user interface under development.

It is important to observe two basic limitations in TUNE. TUNE only evaluates simple user interface elements such as menus, menu items, dialog boxes and buttons. TUNE only evaluates presentation and behavior of user interface elements. Evaluation with respect to task and user is not supported in TUNE. Despite these constraints, the study illustrated that TUNE can

support a developer in the laborious work of inspecting user interface compliance with defined GLSGs.

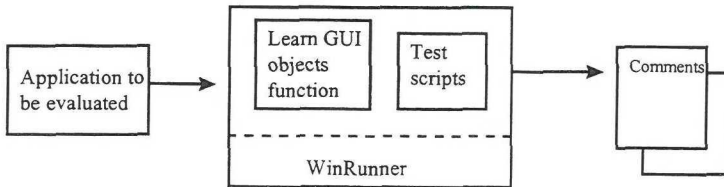
### **7.5 Study 4: Computer Support for User Requirement Evaluation in System Development**

The purpose of this study was to illustrate how to support evaluation of user requirements fulfillment in an application being developed. To do this, a prototype for evaluating if defined user requirements are in a computer system was developed (TURE, Tool for User Requirement Evaluation). The reason for the study was the need to find a means of supporting the comprehensive and laborious process of validating that user requirements are covered in the developed computer system. This process is currently performed manually. TURE was used in a system development project to investigate advantages and disadvantages of this approach in comparison to manual evaluation of user requirement compliance.

#### *7.5.1 TURE*

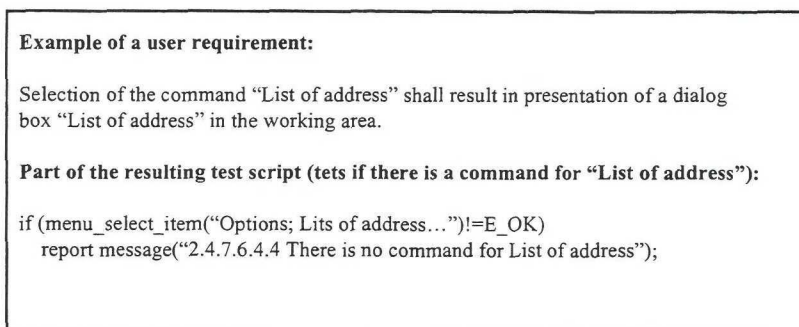
As a platform for the development of TURE, WinRunner (Mercury Interactive Corporation, 1993, a, b, c) was used. WinRunner is a tool for development of automatic software tests of applications that use Microsoft Windows. TURE consists of the following components: The Learn GUI Objects function in WinRunner and a number of specially developed test scripts. The Learn GUI Objects function is used to create a representation of the user interface in the application being evaluated. This representation includes, for example, logical names (labels on buttons, menus etc.,) and physical descriptions (window, dialog box, menu, etc.,) of user interface elements. The test scripts consist of functions in a C-like programming language, containing defined user

requirements. These functions are then used to check that defined user requirements are in the application. The overall architecture of TURE is illustrated in Figure 21 below.



**Figure 21: Overall architecture of TURE**

Creation of test scripts necessitates some kind of requirements documentation (a system requirements specification, for example). With the requirements documentation as a basis, user requirements are transformed into a test script. The level of detail in the requirements documentation influences creation of test scripts. When requirements lack sufficient detail, it is necessary to ask the user to be more specific about their requirements. If this is impossible, details in test scripts must usually wait until a more detailed documentation (for example, a software requirements specification) is developed. In Figure 22, below, an example of a user requirement and the resulting test-script is illustrated.



**Figure 22: An example of user requirement and resulting test-script**

When a prototype, or a version of the application, has been developed, the Learn GUI Objects function is used to create a representation of the user interface. In the case the requirements documentation is very detailed, and all user interface elements are defined (and implemented in the test-script), an evaluation can be performed almost immediately. Otherwise, it is necessary to further develop the test script using the generated representation as a basis.

#### TURE used in practice: A case study:

The study to investigate possible advantages and disadvantages using TURE in practical system development, was carried out by comparing manual and TURE evaluation in evaluating a developed computer system. Performance measures compared were time to create validation specification (a validation specification is a detailed check list for every requirement specified, used in manual evaluation of an application) and test scripts respectively, and time used in manual and TURE evaluation of the application. The results from this case study is depicted in Figure 23 below.

	Manual	TURE
Time used in conducting the evaluation.	8 h	1 h
Time used in creating validation spec./test-scripts	8 h	8 h

**Figure 23: Comparison of manual and TURE evaluation**

As illustrated in Figure 23, the time used for creating validation specification and test-scripts were the same (Note, the number given in the figure should not be seen as absolute, measurement detail was restricted to 15-min intervals). The reason for this was mainly the fact that both validation

specification and the test-scripts were created from scratch and that both requires roughly the same amount of work.

Concerning time used for manual and TURE evaluation the results indicated that use of TURE was much faster.

### *7.5.2 Conclusions*

The result from the study showed that almost equal amounts of time were needed in creation of test scripts and validation specification. The case study also showed that less time was needed to accomplish evaluation using TURE, compared to manual evaluation. However, these results should not be generalized to evaluation of other applications. Further studies are necessary to make any conclusions about possible advantages in wider use of TURE. The study only demonstrated that it is possible to use computer support when evaluating if user requirements are implemented in an application. Also, it is important to note that TURE only inspects user requirements reflected in a user interface. Much work remains before TURE can be considered an efficient tool for use during system development.

Further results from practical use of TURE indicated that:

- it was possible to identify a number of deviations from defined user requirements,
- some of the deficiencies were due to defects in the test-script (some requirement were implemented using other user interface elements and TURE searched for originally defined elements),
- tracing identified deviations back to user requirements was simplified by identification of original requirements in the test report (comments),



- the possibility to replay the evaluation session facilitated communication with developers,
- additional activities are necessary when evaluating an application, for example, validation of user requirements, control if user interface elements have been replaced or changed names, incremental development of test-scripts during system development.

## 7.6 Concluding Remarks

The studies accomplished within this thesis have illustrated and exemplified the following:

- it is possible to support development of user interfaces with tools that inspect user interface design proposals,
- this support is on a low level and focuses on appearance and behavior of individual user interface elements,
- it is possible to utilize knowledge from usability experts, as well as guideline and styleguide collections with this type of tool,
- it is possible to further develop this type of tools and to, at least partially, evaluate user interfaces with respect to user and task issues,
- tools for evaluation of the appearance and behavior of individual user interface elements can be integrated with UIMS tools,
- it is possible to develop support systems to, at least partially, inspect what user requirements are implemented in the computer system developed,
- the tools illustrated in this study are simple prototypes, therefore no definitive conclusions can be drawn,
- further work is necessary.

Below, a short discussion of above conclusions are presented.

Studies 1, 2 and 3 illustrated the feasibility of supporting development of user interfaces with evaluation tools. These tools can be stand-alone modules, or integrated modules in a UIMS.

Studies 1, 2 and 3 demonstrated that these types of tools support design at a low level. Developers participating in the evaluation of the tools felt that identified design flaws were trivial and/or not related to user and task. However, even if the identified design flaws are trivial, it is of value to identify them. Many complex computer systems can include hundreds of user interface elements. If the number of simple design flaws is large, there will be dramatic impact on computer system usability. Another reason for the importance of identifying simple design flaws is that for some applications consistency in and between systems can be of great importance. Command and control systems and process control systems are examples of two types of systems where simple design flaws could have dramatic impact.

Studies on user interface design and evaluation used knowledge from user interface experts as well as knowledge in guideline and styleguide documents. This is a valuable new feature. The potential to implement guidelines and styleguides in computer support for evaluation of user interfaces suggests that this kind of knowledge can be used more often in industrial system development.

Studies 1 and 2 also discussed the possibility to further develop the evaluation functionality to include registration and analysis of user interaction with a computer system. While this technique does not directly address design flaws related to user and task, it is possible to draw some indirect conclusions about design flaws relative to these factors. For example, frequent use of the help function or long sequences for accomplishing usual actions indicates deficiencies in adaptation of user interface to user and task.

Studies 1 and 2 illustrated how to integrate the evaluation functionality with a UIMS tool. Regrettably, this integration has not been developed further. Preliminary work indicates need for development of a function for analysis and compilation of the great number of "events" generated by the UIMS during user interaction with a computer system.

Study 4 demonstrated that it is possible to develop computer support for evaluation that user requirements (some of the user requirements related to the user interface) are implemented in an application. This study should be considered as a pilot study and no definitive conclusions should therefore be drawn. The study provides only a preliminary indication that it is possible to support evaluation of whether defined user requirements are implemented. While the study goal was not about handling user and task aspects along with user interface evaluation, some comments are possible. Further development of the tool presented in this study may extend its function to include evaluation of user interfaces from a user and task perspective. User requirements exemplified in the study suggest that it is possible to also use this kind of scripts as a basis for evaluation of user interface design issues. Supplementing the RTE functionality described in study 2 with this type of script can make a more extensive evaluation of user interfaces from the perspective of user and task practical. This idea of course needs further study to be more fully investigated.

All tools presented in the studies are prototypes. They should be regarded as simple illustrations of what is possible. However, a number of the tools can probably be of value in development of user interfaces, particularly if further developed.

Further research is necessary to investigate how the above mentioned tools can support the process of developing user interfaces and usability work.

There are a number of issues that need further study. Here, a few of them, already mentioned in the studies, are discussed.

First, it is very important to continue the study of how to make it possible to handle user and task issues in design, development and evaluation of user interfaces. The studies did not provide a clear answer, although possible solutions are indicated.

Second, from my experience in usability work in industrial system development, it is important to further develop the kinds of tools described to make them possible to use (and evaluate) in a number of real system development projects. This is necessary to avoid the mistakes presented in Löwgren (1991) that occurred during development of UIMSs.

Third, it is important to further study how to integrate, in an efficient way, tools containing user interface design knowledge with UIMSs and other kinds of User Interface Tools.

Fourth, it is necessary to further study how the process of developing user interfaces is related to, and influenced by other system development activities. To develop support for user interface development, it is necessary to understand how other activities in the system development are accomplished. Although some preliminary studies of the system development process and user interface development have been made (see, for example, Bellotti, 1988; Rosson et al., 1988) extended studies within this area are necessary.

Finally, it is necessary to continue the study of how the tools exemplified above can be integrated in industrial system development, with methods for usability work, and with other tools. This issue is elaborated in Chapter 8.



## 8. FUTURE WORK

### 8.1 Executive Summary of Work Performed

The material in the previous chapters can be summarized into three parts. The first part, in Chapters 1 to 4, uses a traditional description of the system development process to exemplify integration of usability work into system development activities. A definition of usability work and a number of methods for usability work are used to describe this integration. The use and outcome from the application of these methods is described for different system development activities. The second part, Chapters 5 to 6, uses the foundation of the first part to describe the need for further and more comprehensive integration of usability work into industrial system development. Also traditional computer support in system development and its relevance for usability work is discussed. Part 3, Chapter 7, reviews studies on simple computer-based tools to support usability work, primarily user interface development.

The work of integrating usability work and industrial system development is of course not completed. The work in this thesis can be considered as a point of departure. In the remainder of this Chapter ideas concerning possible future attempts are briefly described and reviewed.

Usability work is extensive and requires performance of many activities to be effective; in turn, the need for a range of supporting sub-activities arises. Therefore, preliminary proposals for computer supported usability work (CSUW) in system development, are presented. Other ideas with respect to future work, are described in the end of each chapter, presented earlier.



## 8.2 CSUW in Industrial System Development

### 8.2.1 Introduction

To facilitate identification and discussion of possible CSUW in different system development activities, each system development activity is briefly reviewed. Following is a short description of usability work advocated for each system development activity. Finally, ideas concerning possible computer support are presented. Where computer support for the different methods exists (to my knowledge), they are also mentioned. CSUW proposals should be considered as preliminary. The goal is to present and discuss simple ideas on how to support usability work.

The justification for advocating CSUW are summarized as follows:

- the need to document usability work results, to facilitate use in subsequent and parallel usability work and system development activities,
- the need to continuously develop obtained results further (for example, different kinds of models and prototypes) as additional knowledge is acquired,
- the necessity to make usability work easier and faster to perform. A number of the methods exemplified in this document require excessive resources (time, people and money). All efforts resulting in usability work delivered faster and cheaper, will encourage its integration into the industrial system development process,
- the need for communicating usability work results so that all participants benefit. Computer support that facilitates development of prototypes is important in this communication process (see, for example, Andriole, 1990, 1995; Wood & Kang, 1992),

- in the case where method support is included in the computer support (see, for example, some CASE systems) use of the method by other than experts is facilitated.

See also section 5.6, for further motives for CSUW.

### *8.2.2 Identification of Need*

#### Short summary of system development activity:

Identification of need is the first activity in the system development process. Here, usually a need, or idea, concerning specific computer support or improvement of business, is presented by a customer or user. The need or idea is often expressed in general terms, resulting in the necessity to identify and concretize specific needs or ideas.

#### Short summary of usability work:

To support identification of need, inclusion of business analysis, prototyping and use testing is advocated as a means of providing more complete data. Prototyping is here accomplished using simple prototyping techniques. Use testing consists of simple use tests where users perform a small number of work tasks and use problems are identified. Also the accuracy of the model is use tested.

#### Possible computer support:

In connection with identification of need, computer support for business analysis, prototyping and use testing will speed up work and reduce investment of staff time.

In business analysis (basically consisting of identification and description of current business, identification and prioritization of change needs and modeling of new business) the following computer support will contribute to usability work. First, computer support to describe current business in a way both resulting in an overall description of the complete business and containing detailed information. This would make it possible for business people and developers to fully understand the business. Second, computer support to assist in the prioritization of change needs. Third, computer support for modeling business change. Important here is that the models present a general view of the business and can be understood by all involved in development. Computer support is also useful in illustrating dynamics in the changed business, to increase probability that participants in the business analysis understand potential effects of change proposals.

The method for business analysis described here uses computer support, MacRASP (other business analysis methods also use computer support, for example, the TRIAD method, Willars, 1993a, b). This computer support mainly assists in description of current business, modeling of changed business and consistency checking of descriptions and models. Support for prioritization of change needs or for illustration of business dynamics is minimal.

Hughes (1996) discuss tools for illustrating business process dynamics (based on system dynamics). Also tools like Ithink (High Performance Systems, Inc., 1994) and ReThink (Gensym Corporation, 1995) can be valuable.

In conjunction with identification of need, prototyping consists mainly of development of use cases, screen layouts (describing services) and storyboards. Simple computer support is sufficient to assist in documentation of results from prototyping (for example, word processing and drawing

programs). More advanced functions that illustrate dynamics in prototypes are usually not necessary. Prototypes supplements business models to increase probability that business people/users understand proposed business design.

Use testing can also be supported using simple documentation support to document the work tasks forming the basis in use testing, use problems and user opinions about prototypes and other models.

The concretized needs or ideas from this system development activity need to be documented using natural language. Therefore, some form of documentation support is also needed here. Probably some CASE-system supporting documentation can be of value. An interesting support alternative, in the form of computer-based templates, is presented in Andriole (1996). These templates can, according to Andriole, 1996, be used (and further developed) in the entire system development process.

### *8.2.3 Identification of User Requirements (Requirements Definition)*

#### Short summary of system development activity:

In identification of user requirements, the developer and user identifies and defines user requirements on the computer system to be developed. In this process, detailed analysis of identified needs (ideas) is performed. The purpose being to identify all user requirements on the future computer system, and to describe these requirements in a way understood by developers as well as users.

### Short summary of usability work:

In identification of user requirements, task analysis, prototyping, usability specification and use testing were advocated. Prototyping is here also accomplished with simple prototyping techniques. Use testing is carried out with real users and work tasks. The aim of use testing being to verify that necessary services are going to be delivered by the computer system (to be developed) and that tasks can actually be performed.

### Possible computer support:

Computer support for task analysis, prototyping, usability specification and use testing during identification of user requirements will facilitate usability work through, for example, reduction of time for completion of these activities.

Task analysis includes collection of task information, analysis of the information and modeling of tasks. Computer support for these activities offers structured documentation for easier analysis. Documentation support to better organize collected information is needed. This documentation support will support task analysis in a more profound way if also functionality to directly structure information into groupings of goal, sub-goals, procedures, objects and actions are included. It is also useful to include support for analysis of the information. For example, to identify representative, central and generic components in tasks. Modeling of work tasks will also benefit through support for building of goal structures, procedure structures and taxonomic structures in accordance with TKS (Task Knowledge Structure). A prototype tool, ADEPT, has been developed, that probably can support development of task models (Johnson, Wilson, Markopoulos & Pycock, 1993; Johnson, et al., 1995; Wilson, et al., 1993; Wilson & Johnson, 1995). While this tool is focused on user interface design, it can probably be useful also in the identification of user requirements activity.



Tools like Top Down are, according to Andriole, 1996, also useful for creating task models. Tools also of value to support task analysis are, according to Andriole, 1989, DecisionMap and Expert Choice, at least if there is a need to prioritize tasks to be included in the computer system to be developed.

As prototyping mainly uses the same kind of simple prototyping techniques as in identification of need, the need for support is similar. An interesting prototyping technique for developing simple prototypes is presented in Landay and Myers (1995). They have developed a tool for Sketching Interfaces Like Krazy (SILK), allowing developers to sketch user interfaces using an electronic stylus.

Usability specification does not need extensive computer support. Word processing and drawing programs, supplemented by functionality to divide general usability specifications into more detailed usability specifications, and to check for consistency, will support development. Without this kind of functionality it can be difficult to manage the set of usability specifications needed in development of large and complex computer systems.

Use testing can in conjunction with identification of user requirements be supported by the same kind of computer support as in identification of need. Of value to document is:

- need for other services,
- necessary change of services to better suit work tasks,
- in what respect tasks are possible to perform,
- critical comments concerning correctness of the prototype (model),
- critical comments concerning work task performance.

Also some kind of computer support to document identified user requirements is necessary. This kind of computer support was mentioned in sub-section 8.2.2.

As a supplement to the above, support for the process of determining if all defined services (user requirements) are in the prototype will provide more complete usability data. A simple prototype tool was described in Study 4 (Computer Support for User Requirement Evaluation in System Development) that perhaps can be used to support this process (if further developed, of course).

#### *8.2.4 Overall Design of the System*

##### Short summary of system development activity:

In overall design of the system, focus is on issues concerning distribution of functions to different parts of the system. Questions concerning what functions to be performed by the computer system and by the user can (shall) also be addressed.

##### Short summary of usability work:

Usability work advocated in connection with this activity was business analysis, task analysis, contextual design (part of) and prototyping. The results of business analysis, task analysis and prototyping were obtained previously and the models and descriptions are immediately usable. Contextual design (part of) is carried out during system design. Models, prototypes and descriptions can also be integrated to create a model of the "complete system," which can be use tested to check that general user requirements are fulfilled. The results from User Profiling, Cognitive (Systems) Engineering and GOMS are probably also of value. These methods

provide, for example, information important for the allocation of functions between computer system and user. Also, cognitive walkthrough (jogthrough) and further development of the usability specification were advocated.

#### Possible computer support:

In overall design of the system, use of results (models, descriptions, prototypes, etc.,) from earlier usability work provides a foundation upon which to base the design. For these results to be usable, computer support that finds different models, descriptions and prototypes, compares them, checks for consistency and presents results from evaluation of the models and prototypes is recommended. The possibility to inspect models and prototypes with early usability specifications as a basis also aids design. Integration of models and prototypes to create a general model of the complete system, and to use test it, is advisable. Therefore, computer support to illustrate and integrate models and prototypes are useful.

Cognitive walkthrough (jogthrough) will benefit from support focused at documentation of work tasks, task actions, users and their knowledge. To further develop the usability specification the support tool described in 8.2.4 can be used.

### *8.2.5 Identification of Software Requirements (Software Requirements Analysis)*

#### Short summary of system development activity:

In analysis of software requirements, user requirements are transformed into a description appropriate for software development. For example, flow charts, object models and so on.

#### Short summary of usability work:

During identification of software requirements, performance of prototyping by continuously developing prototypes further was advocated. Here, further development could also mean development of a completely new prototype. Also, further development of other models (for example, business models and task models) and the usability specification was advocated. Use testing of prototypes and other models were also considered necessary.

#### Possible computer support:

Further development of prototypes and other models, and continuous evaluation that software requirements in a proper way reflect the user requirements are key issues. Therefore, it should be useful to have the same kind of computer support as in the preceding system development activities. In the case where further development of a prototype means development of a completely new prototype, it can be of value to utilize some of the tools that support more advanced prototyping techniques. For example, one of the User Interface Tools described earlier.

To verify that defined software requirements reflect user requirements, computer support will simplify the process. In my opinion, this need may be solved in at least two ways. First, to provide computer support for manual

handling and comparison of documented user requirements and software requirements. This can maybe be done with some kind of hyper-text tool. Second, to develop algorithms that automatically compare documented user and software requirements.

An interesting tool that probably can support the difficult work of verifying that software requirements reflect user requirements is presented in Shipman and McCall (1994). This Hyper-Object Substrate (HOS) system supports incremental formalization of information expressed in an informal way by users.

#### *8.2.6 Software Design*

##### Short summary of system development activity:

In software design, functions are allocated to different software modules and software is structured in some convenient way (for example, object-oriented design and/or functional design). Further, flow of information, data structures and algorithms are defined and described. In software design, also user interface design is performed.

##### Short summary of usability work:

The usability work advocated here, promoted the use of guidelines and styleguides, in the sense that relevant guidelines and styleguides should be identified. Also, prototyping of user interface design was advocated. Prototyping is carried out using simple as well as more advanced prototyping techniques. The prototype is then evaluated using guidelines and styleguides, heuristic evaluation, cognitive walkthrough (jogthrough) and use testing in a combination adapted to the situation. Use testing with real users and work



tasks is done later, when design has reached a more final form. Also, further development of the usability specification is performed.

#### Possible computer support:

To identify guidelines and styleguides, and to evaluate the user interface design with these as a basis, is a laborious process (see, for example, de Souza & Bevan, 1990; Mosier & Smith, 1986; Tetzlaff & Schwartz, 1991). As a consequence, guidelines and styleguides are not used (Smith & Mosier, 1984; Thovtrup & Nielsen, 1991). From this, it is possible to conclude that computer support will assist this work and encourage guidelines and styleguide use. Also a number of computer support tools have been developed. Perlman (1989a, b) has developed a hyper-text based checklist. Sadler (1993) has developed an interactive media to support user interface design. Reiterer (1994) presents a multimedia tool and expert system to support the process of developing user interfaces. Studies 1 and 2 in this thesis present a knowledge-based tool, integrated with a UIMS, that automatically check for user interface compliance with defined guidelines and styleguides. Study 3, describes a computer-based tool for automatic inspection of user interfaces (MS Windows based user interfaces) compliance with MS Windows styleguides.

In software design, prototyping is focused on modeling user interface design. For this purpose, it can be appropriate to use more advanced prototyping techniques. To support this work, it is possible to utilize some of the User Interface Tools described earlier and to develop, for example, limited functionality prototypes, high functionality prototypes or selective fidelity prototypes. However, for User Interface Tools to be of real value in a design situation, it is necessary to supplement them with design knowledge on at least two levels. The first level is related to general design of user interface, where the knowledge is focused on what interaction technique to choose in different situations. The second level concerns more detailed knowledge for

design of user interface elements. With respect to the first level, it is possible to utilize different kinds of hyper-text tools (MITRE, 1991; Perlman, 1989, a, b), or multimedia tools (Reiterer, 1994). The computer support tools presented in Studies 1, 2 and 3, can be used in evaluation based on level 2 knowledge. However, these tools cannot be used before a design proposal has been developed.

Concerning heuristic evaluation and cognitive walkthrough (jogthrough) there is value in using simple computer support to assist in documentation of results from these inspections.

In use testing it can be of value to have computer support to register and analyze user interaction with the prototype. Registration can be supported on at least two levels. The first level, described in, for example, Dumas and Redish (1994), supports evaluation by giving access to a computer-based form, where an evaluator can make notes concerning observations about what a user is doing or saying. The second level, supports an evaluation by automatically registering user actions on a prototype. For example, button presses, selection of menu items and time to react on information presented. This type of registration usually requires some kind of analysis tool to compile and present low level interactions for usability work to be efficient. From my experience, it is laborious and time consuming to manually compile and analyze this kind of registrations, also for simple applications (see also Harrison, Owen & Baecker, 1994). The VANNA system and Timelines system (Harrison et al., 1994), are two support systems for collection and analysis of data generated in conjunction with use testing. With these systems it is possible to collect, analyze and visualize quantitative and qualitative data, by using, for example, pre-defined event and interval markers and color coding. Perhaps also usable is the kind of tool described in study 2, where a runtime evaluation module was illustrated.

### *8.2.7 Implementation and Unit Testing*

#### Short summary of system development activity:

This activity mainly consists of implementation (programming) of different software modules in accordance with software design and testing of the modules to verify specified function.

#### Short summary of usability work:

Simple kinds of usability work was advocated here for example, cognitive walkthrough (jogthrough), heuristic evaluation and use of guidelines and styleguides. Also, utilizing usability specifications to continuously evaluate user interface elements implemented was recommended.

#### Possible computer support:

In the same way as in software design, it can be of value to have access to similar tools mentioned in this context. Where a usability specification was documented in a computer support, this tool can also be used during implementation and unit testing.

The above mentioned computer support can perhaps be supplemented with computer support (TURE from Study 4) recommended for identification of user requirements. Utilizing user requirements implemented in TURE, it might be possible to test implemented software modules taking into consideration the services implemented in the modules.

### *8.2.8 Integration and Testing*

#### Short summary of system development activity:

In integration and testing, software modules are integrated and then tested to ensure that integrated modules work as defined. This process is iterated until all software is integrated and tested.

#### Short summary of usability work:

During integration and testing, use testing with real users and real work tasks was advocated.

#### Possible computer support:

The same kind of computer support tools for use testing, described in connection with software design can be of value in integration and testing.

### *8.2.9 Operation and Maintenance*

#### Short summary of system development activity:

When the computer system has been installed and acceptance testing has been carried out, the system is set in operation. After this, continuous maintenance activities usually are accomplished as long as the system is used. The purpose with these maintenance activities is to correct errors and deficiencies, and to further develop the computer system.

Short summary of usability work:

Usability work advocated in connection with operation and maintenance was use testing in a realistic environment.

Possible computer support:

Use testing in the real environment implies further requirements for computer support. For example, in this context it can be of value to simulate actions from other businesses and other users, in order to achieve a more realistic evaluation situation. Also the kind of support tools for registration and analysis of user interaction described in connection with software design and integration & testing can be valuable (see, for example, Nielsen, 1993).

**8.3 Summary**

Above, a number of different computer support opportunities has been exemplified. These proposals are not complete. They should be seen as an attempt to initiate a discussion about the possible need for computer support tools and what they shall support. Hopefully, this brief description of different kinds of computer support will lead to initiation of work focused on supporting usability work in industrial system development. Although, a number of tools have been developed, most are prototypes or research tools.

Important aspects that should influence this future work (hopefully both research and development) are mentioned by Löwgren (1991). They can be summarized as need for increased focus on practical system development and continuous feedback of experiences from practical use. Besides the development of computer support that focuses on usability work, it is also necessary to study how these possible computer support tools can be



integrated with traditional computer support for system development, for example CASE systems and User Interface Tools.

## REFERENCES

- Adelman, L. (1992). *Evaluating Decision Support and Expert Systems*. John Wiley & Sons, Inc., New York. USA.
- Adelman, L., and Donnell, M. L. (1986). Evaluating Decision Support Systems: A General Framework and Case Study. In S. J. Andriole (ed.), *Microcomputer Decision Support Systems: Design, Implementation, and Evaluation*. QED Information Sciences, Inc., Wellesley, Massachusetts. USA, pp. 285-309.
- Adler, P., and Winograd, T. (1992). The Usability Challenge. In P. Adler, and T. Winograd, (eds.), *Usability: Turning Technologies into Tools*, New York: Oxford University Press, pp. 3-14.
- Allusi, E. A. (1991). The Development of Technology for Collective Training: SIMNET, a Case History. *Human Factors*. Reprinted in L. Voss. A Revolution in Simulation: Distributed Interaction in the '90s and Beyond. Pasha Publications Inc., 1993 , Arlington, VA. 22209. USA, pp. 168-187.
- Andersen, N-E, Kensing, F., Lundin, J., Mathiassen, L., Munk-Madsen, A., Rasbech, M., and Sörgard, P. (1990). *Professional Systems Development, Experiences, Ideas and Action*. Prentice Hall International Ltd., United Kingdom.
- Andriole, S. J. (1989). *Handbook of Decision Support Systems*, TAB Professional and Reference Books, Blue Ridge Summit, PA. USA.
- Andriole, S. J. (1990). *Information System Design Principles for the 90s, GETTING IT RIGHT!*. AFCEA International Press, USA.

- Andriole, S. J. (1991). Storyboard Prototyping for Requirements Verification. In S. J. Andriole and S. M. Halpin (eds.), *Information Technology for Command and Control, Methods and Tools for Systems Development and Evaluation*. IEEE Press, New Jersey, USA, pp. 82-98.
- Andriole, S. J. (1994). Prototype or else ....*IEEE Software*. May.
- Andriole, S. J. (1996). *Managing Systems Requirements: Methods, Tools, and Cases*. McGraw-Hill Companies, Inc., USA.
- Andriole, S. J., and Adelman, L. (1991). Prospects for Cognitive Systems Engineering. In S. J. Andriole and S. M. Halpin (eds.), *Information Technology for Command and Control, Methods and Tools for Systems Development and Evaluation*. IEEE Press, New Jersey, USA, pp. 52-59.
- Andriole, S. J., and Adelman, L. (1995). *Cognitive Systems Engineering for User-Computer Interface Design, Prototyping, and Evaluation*. Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey, USA.
- Andriole, S. J., and Monsanto, C. A. (1995). Interactive Collaborative Requirements Management. *Software Development*, May.
- Apple Computer (1992). *Human Interface Guidelines: The Apple Desktop Interface*. Author.
- Ashworth, C., and Goodland, M. (1990). *SSADM: A Practical Approach*. McGraw-Hill.
- ASTM (1991). Standard Guide for Rapid Prototyping of Computerized Systems. *ASTM Designation E 1340-91*, American Society for Testing and Materials, 1916 Race St., Philadelphia, PA 19103, USA.

- Bellotti, V. (1988). Implications of Current Design Practice for the Use of HCI Techniques. In D. M., Jones, and R., Winder (eds.), *People and Computers IV, Proceedings of the Fourth Conference of the British Computer Society, Human- Computer Interaction Specialist Group*, University of Manchester, 5-9 September, Cambridge University Press, Cambridge, pp. 13-34.
- Bergsten, P., Bern, M., Kool, P., and Wingstedt, U. (1993). *Verkt yg f r grafiska anv ndargr nssnitt*. Rapport nr. 20. Swedish Institute for System Development, Stockholm, Sweden.
- Beyer, H., and Holtzblatt, K. (1993). Contextual Design: Toward a Customer-Centered Development Process. *Software Development '93 Spring Proceedings*, Santa Clara, California, Feb., USA.
- Boar, B. (1984). *Application Prototyping: A Requirements Definition Strategy for the 80s*. New York: Wiley Interscience, USA.
- Bodart, F., Hennebert, A-M., Leheureux, J-M., Provot, I., and Vanderdonckt, J. (1994). A Model-Based Approach to Presentation: A Continuum from Task Analysis to Prototype. *Proceedings of Eurographics Workshop on Design, Specification, Verification of Interactive Systems*. Bocca di Magra, 8-10 June, pp. 25-39.
- Boehm, B. W. (1988). A Spiral Model of Software Development and Enhancement. *IEEE Computer*, 21, (5), May, pp. 61-72.
- Brown, C. M. L. (1988). *Human-Computer Interface Design Guidelines*. Ablex Publishing Company, Norwood, USA.

- Brown, A. W., Earl, A. N., and McDermid, J. A. (1992). *Software Engineering Environments, Automated Support for Software Engineering*. McGraw-Hill Book Company, London, United Kingdom.
- Bubenko, J. (1989). Selecting a Strategy for Computer-Aided Software Engineering. SYSLAB, Stockholm University, Sweden.
- Card, S., Moran, T., and Newell, A. (1983). *The Psychology of Human-Computer Interaction*. Hillsdale, NJ: Lawrence Erlbaum.
- Carroll, J. M. (1995). The Scenario Perspective on System Development. In J. M. Carroll (ed.), *Scenario-Based Design, Envisioning Work and Technology in System Development*. John Wiley & Sons, Inc., New York, USA, pp. 1-17.
- Carroll, J. M., and Rosson, M. B. (1985). Usability specifications as a tool in iterative development. In Hartson, H. R. (ed.), *Advances in Human-Computer Interaction*. Ablex, pp. 1-28.
- Carlshamre, P. (1994), A Collective Approach to Usability Engineering: Technical Communicators and System Developers in Usability-Oriented Systems Development, *Linköping Studies in Science and Technology, Thesis No. 455*. Department of Computer and Information Science, Linköping University, Sweden.
- Chapanis, A., and Budurka, W. J. (1990). Specifying human-computer interface requirements. *Behaviour & Information Technology*, vol. 9, No. 6, pp. 479-492.



- Christel, M. G., and Kang, K. C. (1992). Issues in Requirements Elicitation. *Technical Report CMU/SEI-92-TR-12 ESC-TR-92-012*. Software Engineering Institute, Carnegie Mellon University.
- Cronholm, S. (1994). Varför CASE-verktyg i systemutveckling? - En motiv- och konsekvensstudie avseende arbetssätt och arbetsformer. *Humaniora och Samhällsvetenskap FHS-rapport 5/94, Licentiatavhandling*. Institutionen för Datavetenskap, Linköpings Universitet, Linköping, Sweden.
- Davenport, T. H. (1993). *Process Innovation*. Harvard Business School Press, Boston, Massachusetts, USA.
- Davis, A. M. (1990). *Software Requirements: Analysis and Specification*. Prentice-Hall, Inc., USA.
- Davis, A. M. (1993). *Software Requirements: Objects, Functions and States*. Prentice-Hall, Inc., New Jersey, USA.
- Defense Information Systems Agency (1994). *Department of Defense Technical Architecture Framework for Information Management, Volume 8: Department of Defense HCI Style Guide*. Author, June.
- De Souza, F., and Bevan, N. (1990). The Use of Guidelines in Menu Interface Design: Evaluation of a Draft Standard. In D. Diaper, D. Gilmore, G. Cockton, and B. Schakel (Eds.), *Human-Computer Interaction - Interact '90*. North-Holland, pp. 435-440.
- Desurvire, H. W. (1994). Faster, Cheaper!! Are Usability Inspection Methods as Effective as Empirical Testing? In J. Nielsen and R. L. Mack (eds.), *Usability Inspection Methods*. John Wiley & Sons, pp. 173-202.

Desurvire, H. W., Kondziela, J. M., and Atwood, M. E. (1992). What is Gained and Lost When Using Interface Evaluation Methods Other than Empirical Testing? In A. Monk, D. Diaper, and M. D. Harrison (eds.), *People and Computers VII*. Cambridge: Cambridge University Press, pp. 89-102.

Diaper, D. (1989). Task Analysis for Knowledge Descriptions (TAKD): The Method and an example. In D. Diaper, (ed.), *Task Analysis for Human-Computer Interaction*. Ellis Horwood, pp. 108-159.

Dix, A., Finlay, J., Abowd, G. and Beale, R. (1993). *Human-Computer Interaction*. Prentice Hall International Limited. United Kingdom.

Dumas, J. S., and Redish, J. C. (1994). *A Practical Guide to Usability Testing*. Ablex Publishing Corporation, Norwood, New Jersey. USA.

Enqvist, H., and Lethovaara, K. (1996). Personal communication.

Eisner, H. (1987). *Computer Aided Systems Engineering*. Prentice Hall, Englewood Cliffs, New Jersey, USA.

Fernandez, K. (1992). *User Interface Specifications for Navy Command and Control Systems*. AC/141(IEG/5)WP/122, June.

Fischer, G., and Lemke, A. (1988). Framer: Integrating Working and Learning. Manuscript submitted to IJCAI 89.

Fischer, G., and Lemke, A. (1989). Design Environments: From Human-Computer Communication to Human Problem-Domain Communication and Beyond. In *IJCAI'89 Workshop: A New Generation of Intelligent Interfaces*, pp. 53-58.

- Fischer, G., Lemke, A., Mastaglio, T., and Morch, A. (1990). Using Critics to Empower Users. In *CHI'90 Proceedings*, pp. 337-347.
- Fisher, A. S. (1988). *CASE, Using Software Development Tools*. John Wiley & Sons, Inc., New York, USA.
- Flygvpapnet (1993). *Handbok Systemarbete LI FV*. Ver. 2.0. Author. Sweden.
- Foley, J., Kim, W., Kovacevic, S., and Murray, K. (1989). Defining Interfaces at a High Level of Abstraction. *IEEE Software*. January, pp. 25-32.
- Goddard Space Flight Center (1992). *Human-Computer Interface Guidelines*. Author, Aug.
- Goldkuhl, G. (1991). Stöd och Struktur i Systemutvecklingsprocessen. Paper presented at *the Conference on Systemutveckling i praktisk belysning*. Norrköping, Sweden.
- Goldkuhl, G. (1992). Metodanpassning av CASE-verktyg. Institutionen för datavetenskap, Linköpings Universitet, Sweden.
- Goldkuhl, G. and Röstlinger, A. (1988). *Förändringsanalys: Arbetsmetodik och förhållningssätt för goda förändringsbeslut*. Studentlitteratur, Lund, Sweden.
- Gordon, V. S., and Bieman, J. M. (1994). Rapid Prototyping: Lessons Learned. *IEEE Software*, pp. 85-95.
- Gould, J. D., Conti, J., and Hovanyecz, T. (1983). Composing Letters with a Simulated Listening Typewriter. *Communications of the ACM*. 26, 4, April.

- Green, M. (1985). Report on Dialogue Specification Tools. In G. Pfaff (ed.), *User Interface Management Systems* Springer Verlag, Berlin, Germany, pp. 9-20.
- Hammer, M., and Champy, J. (1993). *Reengineering the Corporation: A Manifesto for Business*. Harper Collins Publishers Inc., New York, USA.
- Harrison, B., L., Owen, R., and Baecker, R. M. (1994). Timelines: An Interactive System for the Collection and Visualization of Temporal Data. In W. D. Davis and B. Joe, (eds.), *Graphics Interface '94*. Banff, Alberta, 18-20 May, Palo Alto, CA: Morgan Kaufmann Publishers, pp. 141-148.
- High Performance Systems, Inc. (1994). The Visual Thinking Tools for the 90's. Author.
- Hollnagel, E., Mancini, G., and Woods, D. D. (1988). *Cognitive Engineering in Complex Dynamic Worlds*. London: Academic Press, United Kingdom.
- Holtzblatt, K., and Beyer, H. (1993). Making Customer-Centered Design Work for Teams, *Communications of the ACM*. October, Vol. 36, No. 10, pp. 93-103.
- Holtzblatt, K., and Jones, S. (1993). Contextual Inquiry: A Participatory Technique for System Design. In D. Schuler and A. Namioka (eds.), *Participatory Design: Principles and Practices*. Lawrence Erlbaum Associates Publishers, Hillsdale, New Jersey, pp. 177-210.
- Hughes, G. M. K. (1996). Process Reengineering Case Studies. In S. J. Andriole, *Managing Systems Requirements: Methods, Tools, and Cases*. McGraw-Hill, USA.

IEEE P1233-1993 (1994). *Guide for Developing System Requirements Specifications*. Draft. Institute of Electrical and Electronics Engineers, Inc. New York, USA.

IEEE std 830-1993 (1994). *IEEE Recommended Practice for Software Requirements Specifications*. Institute of Electrical and Electronics Engineers, Inc. New York, USA.

Jackson, M. A. (1983). *System Development*. Prentice-Hall.

Jacobson, I., Christenson, M., Jonsson, P., and Övergaard, G., (1992). *Object-Oriented Software Engineering*. Reading: Addison-Wesley Publishing Company, USA.

Jeffries, R., Miller, J. R., Wharton, C., and Uydea, K. M. (1991). User interface evaluation in the real world: A comparison of four techniques. *CHI'91 Conference Proceedings*, pp.119-124.

John, B. E. (1995). Why GOMS?. *Interactions*, October, pp. 80-89.

John, B. E., and Kieras, D. E. (1996). Using GOMS for User Interface Design and Evaluation: Which Technique? *ACM Transactions on Computer-Human Interaction*. Vol. 3, No. 4, December, pp. 287-319.

Johnson, H., and Johnson, P. (1989). Integrating task analysis into system design: Surveying designers' needs. *Ergonomics*. 32, pp. 1451-1467.

Johnson, H., and Johnson, P. (1990a). Integrating task analysis and system design: Surveying designer's needs, *Ergonomics*. 32, 11, pp. 1451-67.



- Johnson, H., and Johnson, P. (1991). Task Knowledge Structures: Psychological basis and integration into system design. *Acta Psychologica*. 78, pp. 3-26.
- Johnson, P. (1992). *Human Computer Interaction: Psychology, Task Analysis and Software Engineering*. McGraw-Hill Book Company, London.
- Johnson, P., Johnson H., Waddington, R., and Shouls, A. (1988). Task Related Knowledge Structures: Analysis, Modeling and Application. In D. M. Jones and R. Winders (eds.), *People and Computers: From Research to Implementation, HCI'88*. Cambridge University Press, United Kingdom, pp. 137-155.
- Johnson, P., and Johnson, H. (1990b). Knowledge Analysis of Tasks: Task Analysis and Specification for Human Computer Systems. In A. Downtown (ed.), *Engineering the Human-Computer Interface*. McGraw-Hill.
- Johnson, P., Drake, K., and Wilson, S. (1990). A Framework for Integrating UIMS and User Task models in the Design of User Interfaces. In D. A. Duce, M. R. Gomez, F. R. A. Hopgood, and J. R. Lee (eds.), *User Interface Management and Design; Proceedings of Workshop on User Interface Management Systems and Environments*. Springer Verlag, pp. 203-216.
- Johnson, P., Wilson, S., Markopoulos, P., and Pycock, J. (1993). ADEPT - Advanced Design Environment for Prototyping with Task Models. Demonstration Abstract. In *Proceedings of INTERCHI'93*. April, ACM Press, p. 56.
- Johnson, P., Johnson, H., and Wilson, S. (1995). Rapid Prototyping of User Interfaces Driven by Task Models. In J. M. Carroll (ed.), *Scenario-Based*

- Design, Envisioning Work and Technology in System Development*. John Wiley & Sons, Inc., New York, USA, pp. 209-246.
- Karat, C-M. (1992). Cost-Justifying Human Factors Support on Software Development Projects. *Human Factors Society Bulletin*. 35 (11), pp. 1-4.
- Karat, C-M. (1994). A Comparison of User Interface Evaluation Methods. In J. Nielsen and R. L. Mack (eds.), *Usability Inspection Methods*. John Wiley & Sons, pp. 203-233.
- Karat, C-M., Campbell, R. L., and Fiegel, T. (1992). Comparison of Empirical Testing and Walkthrough Methods in User Interface Evaluation. In *Proceedings of the ACM CHI'92 Conference*. (Monterey, CA, May 3-7), pp. 397-404.
- Kelly, C., and Colgan, L. (1992). User Modeling and User Interface Design. In *People and Computers VII, Proceedings of HCI'92 Conference*. Cambridge University Press, pp. 227-239.
- Landay, J. A., and Myers, B. A. (1995). Interactive Sketching for the Early Stages of User Interface Design. In *Proceedings of CHI'95, Mosaic of Creativity*. May 7-11, ACM Press, pp. 43-50.
- Lederer, A. L., and Prasad, J. (1992). Nine Management Guidelines for Better Cost Estimating. *Communications of the ACM*. 35, 2 (February), pp. 51-59.
- Lemke, A., and Fischer, G. (1990). A Cooperative Problem Solving System for User Interface Design. In *Proceedings of the Eight National Conference on Artificial Intelligence*, pp. 479-484.

- Lewis, C. (1982). Using the 'Thinking-Aloud' method in Cognitive Interface Design. *Research Report RC9265*. IBM T. J. Watson Research Center, Yorktown Heights, NY, USA.
- Lewis, C., and Rieman, J. (1993). *Task-Centered User Interface Design*. Shareware, ftp.cs.colorado.edu.
- Löwgren, J. (1991). Knowledge-Based Design Support and Discourse Management in User Interface Management Systems. *Linköping Studies in Science and Technology, Dissertations No. 239*. Department of Computer and Information Science, Linköping University, Sweden.
- Löwgren, J. (1993). *Human-Computer Interaction: What Every System Developer Should Know*. Studentlitteratur, Lund, Sweden.
- Löwgren, J., and Nordqvist, T. (1990). A Knowledge-Based Tool for User Interface Evaluation and its Integration in a UIMS. In D. Diaper, D. Gilmore, G. Cockton, and B. Shackel (eds.), *Human-Computer Interaction Interact '90*. North-Holland, pp. 395-400.
- Löwgren, J., and Nordqvist, T. (1992). Knowledge-Based Evaluation as Design Support for Graphical User Interfaces. In *CHI '92 Proceedings*, pp. 181-188.
- Maulsby, D., Greenberg, S., and Mander, R. (1993). Prototyping an Intelligent Agent Through Wizard of Oz. *Proceedings of ACM INTERCHI'93 Conference*. Amsterdam, The Netherlands, April 24-29.
- Mayhew, D. J. (1992). *Principles and Guidelines in Software User Interface Design*. Prentice Hall, Englewood Cliffs, NJ.

- McClure, C. (1989). *CASE is Software Automation*. Prentice Hall.
- Mercury Interactive Corporation (1993a). *X Runner/WinRunner Technical Overview*, ver. 1.0. California, USA: Author.
- Mercury Interactive Corporation (1993b). *WinRunner User's Guide*, California, USA.: Author.
- Mercury Interactive Corporation (1993c). *Context Sensitive Testing, User's Guide*, California, USA.: Author.
- Microsoft (1993). *The Windows Interface: An Application Design Guide*. Microsoft Press, Redmond, Washington, Author.
- Miller-Jacobs, H. H. (1991). Rapid-Prototyping: An Effective Technique for System Development. In J. Karat (ed.), *Taking Software Design Seriously*. Academic Press, Inc., pp. 273-286.
- MILSTD 498 (1994). Military Standard for Software Development and Documentation, AMSC No. N7069.
- MITRE (1991). *Dynamic Rules for User Interface Design, DRUID 2.0* B MITRE Corporation, Bedford Massachusetts, USA.
- Mosier, J. N., and Smith, S. L. (1986). Application of Guidelines for Designing User Interface Software, *Behaviour and Information Technology*. 3, pp. 39-46.
- Myers, B. A. (1989). User-Interface Tools: Introduction and Survey. *IEEE Software*. January, pp. 15-23.

- Myers, B. A. (1993). State of the Art in User Interface Software Tools. In H. R. Hartson and D. Hix (eds.), *Advances in Human-Computer Interaction*. Vol. 4, Ablex Publishing Corporation, Norwood, New Jersey, USA, pp. 110-150.
- Myers, B. A. (1995). User Interface Software Tools. *ACM Transactions on Computer-Human Interaction*, Vol. 2, No. 1, March, pp. 64-103.
- Myers, B. A., and Rosson, M. B. (1992). Survey on User Interface Programming. In *Proceedings of CHI'92 Conference on Human Factors in Computing Systems*. pp. 195-202.
- Nielsen, J. (1990). Paper versus computer implementations as mockup scenarios for heuristic evaluation. *Proceedings of IFIP INTERACT'90 Third International Conference on Human-Computer Interaction*. Cambridge, UK, 27-31 August, pp. 315-320.
- Nielsen, J. (1992). Finding Usability Problems Through Heuristic Evaluation. *Proceedings of the ACM CHI'92 Conference*. (Monterey, CA, May 3-7), pp. 373-380.
- Nielsen, J. (1993). *Usability Engineering*. Academic Press, Inc. San Diego CA.
- Nielsen, J. (1994). Heuristic Evaluation. In J. Nielsen and R. L. Mack (eds.), *Usability Inspection Methods*. John Wiley & Sons, Inc., pp. 25-62.
- Nielsen, J. (1995), Applying Discount Usability Engineering, *IEEE Software*. January, pp. 98-100.



- Nielsen, J., and Molich, R. (1990). Heuristic Evaluation of user interfaces. *Proceedings of ACM CHI'90 Conference*. (Seattle, WA, 1-5 April) pp. 249-256.
- Nordqvist, T. (1995). Computer-Supported for User Requirement Evaluation in System Development. *Research Report, TULEÅ 95:37*. Luleå University of Technology.
- Nordqvist, T. (1996). TUNE: A Tool for User Interface Evaluation. In *Proceedings of the Sixth Australian Conference on Computer-Human Interaction, (OZCHI'96)*. Hamilton, New Zealand, November 24-27, pp. 129-134.
- Näslund, T. (1994). Usability is extremely important - but it's somebody else's job, I hope. In P. Kerola, A. Juustila and J. Järvinen (eds.), *Proceedings of the 17th IRIS (Information Systems Research Seminar in Scandinavia)*. University of Oulu, Dept. of Information Processing Science, pp. 653-667.
- Olsen, D., Green, M., Lantz, K., Schulert, A., and Sibert, J. (1987). Whither (or wither) UIMS? In *Proceedings of CHI+GI'87*, pp. 311-314.
- Olsen, D. R., Jr., and Halversen, B. W. (1988). Interface Usage Measurements in a User Interface Management System. In *ACM SIGGRAPH Symposium on User Interface Software and Technology Proceedings UIST'88*. ACM, New York, pp. 102-108.
- Open Software Foundation (1988). *OSF/Motif Styleguide, revision 1.1*, Cambridge Massachusetts, USA.

- Open Software Foundation (1993). *OSF/Motif Styleguide, rev. 1.2*. Prentice Hall, Englewood Cliffs, New Jersey, USA.
- Palmer, J. D. (1990). Software System Requirements Engineering for Command and Control. In S. Andriole (ed.), *Advanced Technology for Command and Control Systems Engineering*. AFCEA International Press, Fairfax, Virginia, 22033-3899 USA, pp. 18-31.
- Perlman, G. (1989a). System Design and Evaluation with Hypertext Checklists. *Proceedings of the 1989 IEEE Conference on Systems, Man and Cybernetics*, pp. 1187-1193.
- Perlman, G. (1989b). The checklist method for applying guidelines to design and evaluation. *Proceedings of INTERFACE 89*, pp. 271-276.
- Polson, P. G., Lewis, C., Rieman, J., and Wharton, C. (1992). Cognitive Walkthroughs: A method for theory-based evaluations of user interfaces. *International Journal of Man-Machine Studies*. 36, pp. 741-773.
- Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., and Carey, T. (1994). *Human-Computer Interaction*. Addison-Wesley, Wokingham, England.
- Raghavan, S., Zelesnik, G., and Ford, G. (1994). Lecture Notes on Requirements Elicitation. *Educational Materials CMU/SEI-94-EM-10*. Carnegie Mellon University, Software Engineering Institute.
- Reiterer, H. (1994). *User Interface Evaluation and Design, Research Results of the Projects Evaluation of Dialogue Systems (EVADIS) and User Interface Design Assistance (IDA)*. R. Oldenbourg Verlag, Munich, Germany.

- Rettig, M. (1994). Prototyping for Tiny Fingers, *Communications of the ACM*. April.
- Robinson, P. J. (1992). *Hierarchical Object-Oriented Design*. Prentice Hall, Englewood Cliffs, New Jersey, USA.
- Rosson, M. B., Maass, S., and Kellogg, W. A. (1988). The Designer as User: Building Requirements for Design Tools from Design Practice. *Communications of the ACM*. Vol. 31. No. 11, November, pp. 1288-1298.
- Rowley, D. E., and Rhoades, D. G. (1992). The Cognitive Jogthrough: A Fast-Paced User Interface Evaluation Procedure. *Proceedings of the ACM CHI'92 Conference*. (Monterey, CA, May 3-7), pp. 389-395.
- Royce, W. W. (1970). Managing the Development of Large Software Systems: Concepts and Techniques. In *Proceedings IEEE WESTCON*. Los Angeles, USA, pp.1-9.
- Sadler, H. J. (1993). Making it Macintosh: An Interactive Human Interface Instructional Product for Software Developers. In *INTERCHI '93 Adjunct Proceedings*, pp. 37-38.
- Sage, A. P. (1992), *Systems Engineering*. John Wiley & Sons, Inc., New York, USA.
- Sage, A. P., and Palmer, J. D. (1990). *Software Systems Engineering*. John Wiley & Sons, Inc., USA.
- Schmucker, K. J. (1986). MacApp: An Application Framework. *Byte*, 11, Vol. 8, August, pp. 189-193.

- Sharon, D., and Bell, R. (1995). Tools that Bind: Creating Integrated Environments. *IEEE Software*. 12, (2), March, pp. 76-85.
- Shipman III, F. M., and McCall, R. (1994). Supporting Knowledge-Base Evolution with Incremental Formalization. *Proceedings of Human Factors in Computing Systems, CHI '94*. April 24-28, Boston, Massachusetts, USA, pp. 285-291.
- Shneiderman, B. (1992). *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Second edition. Addison-Wesley, Reading, Massachusetts, USA.
- Smith, S. L. (1988). Standards Versus Guidelines for Designing User Interface Software. In M. Helander (ed.), *Handbook of Human- Computer Interaction*. Elsevier Science Publishers B. V. North-Holland, pp. 877-889.
- Smith, S. L., and Mosier, J. N. (1984). The User Interface to Computer-Based Information Systems: A Survey of Current Software Design Practice. *Behaviour and Information Technology*. 3, pp. 195-203.
- Smith, S. L., and Mosier, J. N. (1986). Guidelines for Designing User Interface Software, *Technical Report, MTR-10090*. The MITRE Corporation, Bedford, MA, 01730, USA.
- Sommerville, I. (1992). *Software Engineering*. Fourth edition. Addison-Wesley Publishing Company Inc., USA.
- Sommerville, I. (1996). *Software Engineering*. Fifth edition. Addison-Wesley Publishing Company Inc., USA.

- Sukaviriya, P., Foley, J. D., and Griffith, T. (1993). A Second Generation User Interface Design Environment: The Model and the Runtime Architecture. In *Human Factors in Computing Systems Proceedings INTERCHI '93*. ACM, New York, pp. 375-382.
- Tanner, P., and Buxton, W. (1985). Some Issues in Future User Interface Management Systems (UIMS) Development. In G. Pfaff, (ed.), *User Interface Management Systems*. Springer Verlag, Berlin, Germany.
- Telesoft AB (1989). *TeleUSE Reference Manual*. 1.0 edition. Linköping, Sweden, Author.
- TELUB AB and System Development Associates (1990). *RASP: En översikt*. Author. Sweden.
- TELUB AB (1995). *RASP Handbok*. Author. Sweden.
- Tetzlaff, L., and Schwartz, D. R. (1991). The Use of Guidelines in User Interface Design. In *CHI '91 Proceedings*, pp. 329-333.
- Thovtrup, H., and Nielsen, J. (1991). Assessing the Usability of a User Interface Standard. In *CHI '91 Proceedings*, pp. 335-341.
- U.S. Department of Defense (1985). *Defense System Software Development*. DOD-STD-2167. June.
- Vlissides, J. M., and Linton, M. A. (1990). Unidraw: A Framework for Building Domain-Specific Graphical Editors. *ACM Transactions on Information Systems*. 8, Vol. 3. (July), pp. 204-236.



- Waddington, R., and Johnson, P. (1989a). Designing and Evaluating Interfaces Using Task Models. In G. X. Ritter (ed.), *11<sup>th</sup> World Computer Congress (IFIP Congress 1989)*. North-Holland.
- Waddington, R., and Johnson, P. (1989b). A Family of Task Models for Interface Design. In A. Sutcliffe and L. Macaulay (eds.), *HCI'89*. Cambridge University Press.
- Wasserman, A. I. (1990). Tool Integration in Software Engineering Environments. In *Proceedings of International Workshop on Environments*. Berlin, pp. 137-149.
- Wharton, C., Rieman, J., Lewis, J., and Polson, P. (1994). The cognitive walkthrough: A Practitioners Guide. In J. Nielsen and R. L. Mack (eds.), *Usability Inspection Methods*. John Wiley & Sons, Inc., pp. 105-140.
- Whiteside, J., and Wixon, D. (1987). The Dialectic of Usability Engineering. In H.-J., Bullinger, and B. Shackel, (eds.). *Human-Computer Interaction - Interact '87*. Amsterdam: Elsevier, pp. 17-20.
- Whiteside, J., Bennett, J., and Holtzblatt, K. (1988). Usability Engineering: Our Experience and Evolution. In M. Helander (ed.), *Handbook of Human- Computer Interaction*. Elsevier Science Publishers B. V. North-Holland, pp. 791-817.
- Wiklund, M. E. (1994), *Usability in Practice*, Academic Press, Inc., Cambridge, Massachusetts, USA.
- Willars, H. (1993a). TRIAD, Modelleringshandboken N 10:1, *SISU Rapport*. Sweden.

- Willars, H. (1993b). TRIAD, Modelleringshandboken N 10:2, *SISU Rapport*. Sweden.
- Wilson, S., Johnson, P., Kelly, C., Cunningham, J., and Markopoulos, P. (1993). Beyond Hacking: A Model Based Approach to User Interface Design. In J. Alty, D. Diaper, and S. Guest, (eds.), *Proceedings of CHI'93*. Cambridge University Press, pp. 217-231.
- Wilson, S., and Johnson, P. (1995). Empowering Users in a Task-Based Approach to Design. In G. M. Olson, and S. Schuon, (eds.), *Proceedings on Symposium on Designing Interactive Systems: Processes, Practices, Methods & Techniques, DIS '95*. ACM, pp. 25-31.
- Wixon, D., Holtzblatt, K., and Knox, S. (1990). Contextual Design: An Emergent View of System Design. In *Proceedings of CHI'90: Conference on Human Factors in Computing Systems*. Seattle, WA, New York: Association for Computing Machinery, pp. 329-336.
- Wood, D. P., and Kang, K. C. (1992). A Classification and Bibliography of Software Prototyping, *Technical Report, CMU/SEI-92-TR-13*. October. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, USA.
- Woods, D. D. (1988). Commentary: Cognitive Engineering in Complex and Dynamic Worlds. In E. Hollnagel, E., G. Mancini and D. D. Woods (eds.), *Cognitive Engineering in Complex Dynamic Worlds*. Academic Press, USA, pp. 115-129.
- Woods, D. D., and Roth, E. M. (1988). Cognitive Engineering: Human Problem Solving with Tools. *Human Factors*. 30(4), pp. 415-430.

Woods, D. D., and Roth, E. M. (1988). Cognitive Systems Engineering. In M. Helander (ed.), *Handbook of Human-Computer Interaction*. Elsevier Publishing Company, Amsterdam, Netherlands.

Workshop Proceedings (1991). Requirements Engineering and Analysis, *Technical Report, CMU/SEI-91-TR-30*.

Voss, L. D. (1993). *A Revolution in Simulation: Distributed Interaction in the '90s and Beyond*. Pasha Publications Inc., Arlington, VA. 22209, USA.

## **Study 1**

# A Knowledge-Based Tool for User Interface Evaluation and its Integration in a UIMS

Jonas Löwgren

Dept. of Computer and Information Science, Linköping University  
S-581 83 Linköping, SWEDEN

Tommy Nordqvist

National Defense Research Establishment (FOA52), P.O. Box 1165  
S-581 11 Linköping, SWEDEN

## Abstract

This paper describes and discusses a knowledge-based user interface evaluation tool, based on the *critiquing* paradigm. The tool uses knowledge acquired from experts and from collections of guidelines to evaluate a formal description of a user interface design, generating comments as well as suggesting improvements.

After describing the system architecture and reporting some experiences, the paper focuses on the possibility of incorporating a knowledge-based design tool in a *User Interface Management System* (UIMS), making it possible to give constructive advice to the designer as well as comments. We report some preliminary results from a project aimed at this integration.

## 1 Introduction

User Interface Management Systems (UIMSs) were originally conceived as tools for facilitating user interface development within the existing software development process. Issues such as rapid prototyping and reusability are well understood and often put forward as advantages gained from using a UIMS. Recently, however, there has been a notable interest in additional support and functionality, not earlier considered part of normal user interface development software. For instance, Myers writes:

[UIMSs] do not support evaluation. Very few user-interface tools provide any support for evaluating the user interface. More research into how the computer could do such evaluation is needed before such support is practical. (Myers 1989, p. 23)

Similar observations have been made by several authors, including Olsen *et al* (1987) and others. This paper presents a contribution to the research called for by Myers in that we present a knowledge-based system that illustrates the feasibility of computer-supported user interface evaluation. Furthermore, we show how a tool of this kind can be incorporated into a UIMS, providing support for user interface designers in designing and evaluating user interfaces.

Other researchers have contributed work in the same area, notable contributions including the Framer system

(Fischer and Lemke 1988, 1989) and a tool called Designer (Weitzman 1988). However, whereas the Framer project focussed on an argumentative environment for design, and Designer only represents low-level graphic design knowledge, our aim is to support *evaluation* of user interfaces on several levels, as we shall see presently.

## 2 The KRI system

The KRI system (Knowledge-based Review of user Interfaces) was developed as a pilot project in order to assess potential advantages and disadvantages with a knowledge-based critiquing approach to the problem of supporting evaluation of user interfaces. To be precise, we are dealing with what is known as *expert-based* evaluation (Howard and Murray 1987) which comprises evaluation based on an expert's subjective knowledge. The project addressed evaluation of form-filling user interfaces with menu-driven navigation by means of function keys. This section describes the prototype system and discusses some results and conclusions that arose.

### 2.1 System architecture

The KRI system, being a fairly traditional stand-alone knowledge-based system, comprises the following principal components:

- a knowledge base containing evaluation knowledge;
- a database with user interface design guidelines;



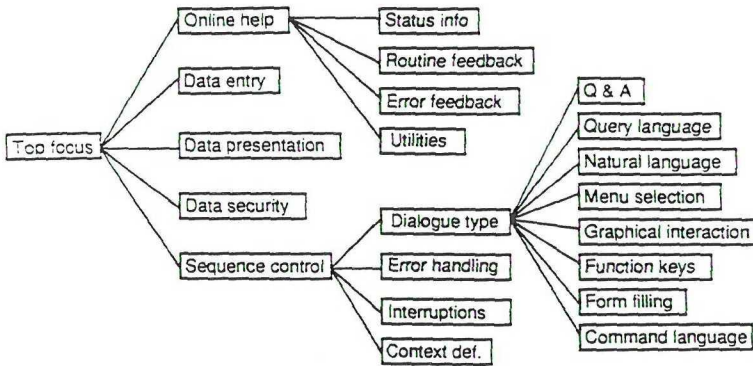


Figure 1: A part of the user interface aspect taxonomy.

• a user interface aspect taxonomy.

The *evaluation knowledge base* is represented in rule form and contains evaluation knowledge from two main sources: (i) transcripts of several expert evaluations of a user interface under development, and (ii) the expert's interpretations of the general user interface design knowledge compiled in guideline documents (Smith and Mosier (1986) and others). In the KRI system only knowledge pertaining to the user interface levels of layout and syntax was implemented. The reason for this, as we shall see in the subsequent section, was that the user interface representation used in the system only supported reasoning about these levels.

The inference mechanism of the system is forward chaining, with the rules designed to detect and report mistakes in the design. This is the most straightforward way of building a critiquing system, but as we discuss in section 4, it is not the only way.

The *guidelines*, which were taken from Smith and Mosier (1986), are not in themselves actively used in the reasoning process of the system. Since the rule base contains interpretations of some of the guidelines, the contents are still there, but the reason for storing the guidelines also in a textual form is different. They are used as justifications for some of the comments generated by the system. We found this to be reassuring to the users of the KRI system.

The *aspect taxonomy*, part of which is illustrated in Figure 1, is used in two ways. First, it is presented to the user of the KRI system as a graph, in which the user can mark the topics of interest for the current session. Secondly, it is used internally as a means of structuring the knowledge base.

## 2.2 User interface representation

In order for the KRI system to be able to reason about properties of the user interface that is being evaluated, the user interface has to be represented in the system in some way. Given the type of user interfaces that we chose to focus on, viz. systems where the user employs function keys to navigate in a number of menus and a tree of forms to be filled out, we selected a simple version of a transition network where the nodes contain information about which objects (menus and forms) that are currently visible and active, and the tokens labeling the arcs correspond to keystroke commands. The systems are supposed to run on a character graphics terminal with a keyboard featuring arrow and function keys. The objects of the interface are also represented separately with information about their appearance. Thus this representation gives us both lexical and syntactical properties of the user interface.

## 2.3 System operation

In this section, we describe the work sequence of an evaluation session using the KRI system.

When the designer has developed a design suggestion or a part of a design, it is possible to have this evaluated by the system in the following sequence. First, the user interface representation is loaded into the system.

Next, the evaluation session is initiated. The first thing that the user has to do is to select relevant aspects of the evaluation taxonomy for this session. This selection phase is performed in an interactive way, where the system decomposes the current selections into more detailed topics, at each stage giving the user the opportunity to select the ones that are of interest. To let the user compose his own evaluation plan is a convenient way of addressing the generally very difficult problem of

planning evaluation sessions in a supportive way. When the user is satisfied with the foci of interest for the review, the system starts evaluating the user interface design.

In this phase, the system walks through the evaluation plan that the user has just specified and executes the rules that are associated with each evaluation domain. The forward chaining reasoning process generates conclusions and comments about the aspects of the evaluated interface that the knowledge in the rulebase covers. The system also processes the messages somewhat; for instance, when the same flaw is detected in several components of the evaluated user interface, the messages are aggregated to one single comment.

When the system has completed the evaluation, it is possible for the user to browse through the results and examine the comments generated in the evaluation phase. The user can select evaluation domains to analyze further. He can also select specific messages and have the system present the reasons for generating the messages along with suggested improvements. It is also possible to have the system search the guidelines database and present directly quoted guidelines as a source of reference. The following example, where the KRI tool was applied to evaluate an independently developed application, illustrates the kind of comments that the system generates.

### 2.3.1 Example of evaluation comments

The user interface under evaluation consisted of three separate tables where the user could enter data. Six pull-down menus were available in the top area of the screen. Each table had to be activated before data entry, i.e., the cursor had to be moved to that table. This could be accomplished either by menu selection or by using dedicated function keys. When the evaluation reached the "Function keys" evaluation domain, the following comment was generated (translated to English by the present authors):

There is a mismatch between the presentation order of the tables and the implicit (ASCII code) order of the function keys used to access the tables.

The reason why the system generated this comment is that the tables (counting from the top of the screen) were activated with function keys 3, 1, and 2, respectively. The most interesting thing about this comment, however, is that it came as a surprise to the designer of the user interface in question. He had used the function keys to reflect the order that he intended to be the most suitable for carrying out the task, not considering the more simple-minded, lexical interpretation of the ordering. His conclusion was that it might be worth considering changing the screen layout.

## 3 Epistemological issues

In this section we discuss the evaluation knowledge represented in the system and how it can be acquired. Since the level of knowledge is inherently related to what it is intended to reason about, i.e., the user interface representation, we also discuss briefly the issue of user interface representation levels.

User interface design knowledge is compiled and publicly available in collections known as *guideline documents*. Consider and compare the following two guidelines:

1. [For a menu.] related options should be grouped from general to specific.
2. [For a button.] the selectable area should be at least 0.25 in (0.6 cm) square.

They are both taken from a collection of computer graphics guidelines compiled by Davis and Swezey (1983, p. 122), and illustrate well the span of such guideline collections. Ranging from presentation aspects through syntactic and semantic (related to meaning) to pragmatic (task-related) considerations, these guidelines are written for humans to use and interpret. When we want to implement this knowledge in specific design rules, we have to interpret and tailor the guidelines in order to arrive at something usable. As Smith (1988) points out, this tailoring is also related to the specificity of the guidelines: the more general they are, the more they have to be qualified before they can actually be applied.

### 3.1 Knowledge acquisition issues

As pointed out above, the available collections of guidelines provide an immense source of knowledge about user interface design. This knowledge has to be classified and sometimes specialized before it can be used in a reasoning system, and a highly relevant question is to what extent the guidelines are applicable at all for this purpose. Let us dwell for a moment upon how the guidelines relate to the actual knowledge acquisition that was carried out within the KRI project.

Our main method of knowledge acquisition was collecting transcripts of a human factors expert evaluating several user interfaces. The transcripts were then "played back" to the expert and the resulting discussion generated the major part of the knowledge implemented in the system. However, we found that many of the expert's comments pertained to higher levels such as task- and user-related issues (pragmatics) that we were unable to handle due to the fact that our user interface representation concerned only presentation and syntax. The issue of user interface representation level is further discussed below.



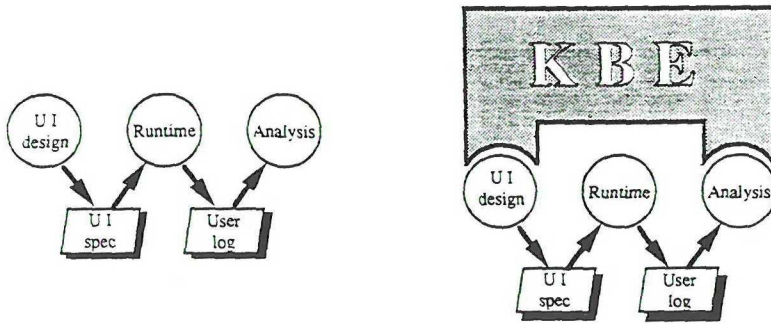


Figure 2: The Seeheim UIMS runtime model (adapted from Tanner and Buxton (1985)), left, and our proposed enhancement, right.

The generality of the guidelines was demonstrated by the observations that (i) it was almost always possible to find a guideline that catered for a remark made by the expert, but (ii) there were almost no guidelines that were specific enough to be implemented directly in the system. Those that were, tended to generate comments that the expert perceived as trivial. In conclusion, guidelines do not seem to replace human experts for knowledge engineering purposes.

### 3.2 User interface representation

When experts examined the comments generated by the KRI system, a number of these comments were judged either trivial or failing to take semantic aspects or user- and task characteristics into account. The reason for the system's inability to evaluate user interfaces on these levels is of course that the user interface representation used is not concerned with them. This turns out to be a difficult tradeoff situation: high-level representation techniques such as, for instance, the semantic-level representation used in the UIDE system (Foley *et al* 1987) are not commercially feasible when considering compatibility and methodology issues. They are also sometimes very demanding to use. On the other hand, they open up possibilities for user interface evaluation on a level that can not be attained in the more conventional presentation and syntax representations.

## 4 Enhancing a UIMS

In the previous section, we saw that the KRI system indeed demonstrated the feasibility of a knowledge-based critiquing approach to user interface evaluation support. However, for a system of this kind to support design-time evaluation and hence the user interface designers, it has to be integrated in the design environment (the UIMS). Furthermore, as pointed out by Fischer and Lemke (1988), the integration of working and learn-

ing that would be obtained by integrating an evaluation package in the design environment has many potential educational benefits. The rest of this section is devoted to describing a current project that is being carried out with the aim of augmenting an existing UIMS with a knowledge-based evaluation module, designed along the lines of the KRI system. In this context, we also discuss how some of the problems of attaining an adequate user interface representation can be addressed using the UIMS runtime structure. For reasons of space, we can not go into detailed discussions. The interested reader is referred to Löwgren *et al* (1989) for a more thorough treatment of this integration project.

### 4.1 An architecture

Already in 1983, Tanner and Buxton formulated a model of the runtime structure of a UIMS (Figure 2, left). This model, which has gained widespread acceptance, covers the activities involved in designing a user interface using a UIMS, and the resulting specifications and data. The design process results in a user interface specification (in some representation format) that is executed together with the application at runtime. The UIMS is responsible for collecting a log of all interactions occurring between user and application across the interface. This log may then be evaluated in some way, not further detailed by Tanner and Buxton.

Our proposed enhancement is shown to the right in Figure 2. We can see that the knowledge-based evaluation module (KBE) is intended to support user interface generation as well as evaluation of the interaction log. The current project that we describe below mainly addresses the issue of design-time support. However, we submit that using the interaction log can contribute to the quality of the evaluation in several ways. For example, it is possible to use information from the log to compensate for a less expressive user interface representation. These two aspects of evaluation are discussed in the two following subsections.

## 4.2 Design-time support

During the phase of user interface design, the KBE module is used for evaluating the user interface specification being constructed. This is accomplished by integrating the evaluation functionality into a design tool, a UIMS. We are currently in the process of integrating evaluation support into TeleUSE, a commercially available UIMS developed by TeleSoft (TeleSoft 1989). It is a general UIMS for graphical interfaces based on the Seeheim model, dividing the user interface into presentation, syntax, and semantic components. The presentation level is expressed in terms of X Windows widgets, while the syntactic aspects of a user interface is implemented in an event handling language based on the D language developed by Hill (1986) in his Sassafra UIMS. This language supports multithreaded dialogue and is responsible for synchronizing the presentation with the application functionality.

The current objective of the project is to support evaluation on the presentation level, i.e., we are augmenting the TeleUSE graphical editor with a knowledge-based module that is capable of evaluating a textual representation of a collection of X widgets. We have decided to implement evaluation on demand as opposed to continuous monitoring. In other terms, there is an evaluation command available for the user of the graphical user interface editor. When this command is invoked, the selected interface objects or the whole interface constructed so far is sent to the evaluation module which generates comments and possibly suggestions for changes.

### 4.2.1 Functionality of the KBE at design-time

There are a number of interesting design decisions to be made when integrating a KBE into a user interface design tool, including:

- **TYPE OF ADVICE.** Should the system only point out flaws in the design (like the Framer system), or should it have (at least limited) capabilities of generating design solutions?
- **SPECIFICITY.** A system based on general design knowledge of the type found in guidelines collections can of course only generate comments on a general, domain-independent level. We feel, however, that one of the most important benefits of an evaluation system integrated into the design environment is its potential to support and enforce organization- and end user-specific design rules.
- **LOCUS OF CONTROL.** Should the system automatically comment upon every mistake it detects, or should we leave to the designer to call upon the evaluation functionality?

## 4.3 Post-runtime evaluation

As was demonstrated earlier, the level of user interface representation determines the level of reasoning in the evaluation system. X widgets only determine appearance, and hence that is all that we can evaluate at design-time. But by using the interaction log, it is possible to compensate to some extent for this deficiency. This log, which is essentially a time-stamped protocol of all events pertaining to the user interface, contains a lot of information that can be potentially useful for evaluation purposes. Even though the information is on a lexical level, it allows us to reason about several aspects of the user interface design, including *selection frequencies* (for menu items and the like); *user proficiency*, quantified analogously to the keystroke model (Card *et al* 1983); *common subsequences* that could possibly be factored out; the *empirical syntax* implicitly formed at runtime; *errors* and *help requests*, indicating the dialogue states that are particularly difficult for the user to handle.

In conclusion, we believe that using the runtime log for evaluation purposes is a way of addressing the difficult tradeoff between powerful user interface representations and designer acceptance.

## 5 Conclusions

The KRI project has indicated a certain potential for success in using knowledge-based techniques for UI design support and evaluation. We have illustrated how this kind of support tool may be used to enhance a traditional UIMS. In addition to supporting the designer in his construction of user interfaces, the tool we propose would also make use of the interaction log collected at runtime. This would to some extent address the problem of needing a very rich user interface representation for the purposes of adequate evaluation, a representation that may be too demanding to use to gain general acceptance. The interaction log to some extent compensates for deficiencies in the user interface representation of the UIMS. Work is under way to implement this architecture, which we feel would be a most valuable tool in the hands of a user interface designer.

Smith (1988) acknowledges that a design tool such as the one outlined in the present paper would shorten the design time and ensure design consistency. However, as he correctly points out, a tool that enforces design guidelines may not be capable of accommodating desirable exceptions and innovative concepts. This is precisely why a critiquing approach to design support is so attractive, combining compliance and non-intrusiveness with the design power equivalent to that of an enforcing tool.



## Acknowledgments

The authors are grateful to Sture Hägglund for his valuable comments which helped improve this paper. Göran Forslund and Björn Peters did a nice job of implementing the KRI system. The current project group includes Per Asplund at FOA, Kent Lundberg, Karl-Erik Hedin and Leif Larsson at TeleSoft, Staffan Löf and Göran Forslund at Epitec, and Sture Hägglund at Linköping University, all of whom contributed to the work described in the latter parts of this paper.

## References

- S. Card, T. Moran, and A. Newell (1983). *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, Hillsdale, NJ.
- E. Davis and R. Swezey (1983). Human factors guidelines in computer graphics: a case study. *Int. Journal of Man-Machine Studies*, 18:113-133.
- G. Fischer and A. Lemke (1988). Framer: integrating working and learning. Manuscript submitted to IJCAI 89.
- G. Fischer and A. Lemke (1989). Design environments: from human-computer communication to human problem-domain communication and beyond. In *IJCAI'89 Workshop: A new generation of intelligent interfaces*, pages 53-58. Position paper.
- J. Foley, C. Gibbs, W. C. Kim, and S. Kovacevic (1987). *A Knowledge Base for User-Computer Interface Design*. Technical Report GWU-IIST-87-11, The George Washington University, Washington DC, August.
- R. Hill (1986). Supporting concurrency, communication, and synchronization in human-computer interaction—the Sassafras UIMS. *ACM Trans. on Graphics*, 5(3):179-210.
- S. Howard and M. D. Murray (1987). A taxonomy of evaluation techniques for HCI. In *Proc. Interact '87*, pages 453-459.
- J. Löwgren, T. Nordqvist, and S. Löf (1989). *Knowledge-Based Support for User Interface Evaluation in User Interface Management Systems*. Research report LiTH-IDA-R-89-32, Linköping University.
- B. Myers (1989). User-interface tools: introduction and survey. *IEEE Software*, January.
- D. Olsen, M. Green, K. Lantz, A. Schulert, and J. Sibert (1987). Whither (or wither) UIMS? In *Proceedings of CHI+GI'87*, pages 311-314.
- S. L. Smith and J. N. Mosier (1986). *Guidelines for Designing User Interface Software*. Report ESD-TR-36-278, Mitre Corp., Bedford, MA.
- S. L. Smith (1988). Standards versus guidelines for designing user interface software. In M. Helander, editor, *Handbook of Human-Computer Interaction*, pages 877-889, Elsevier Science Publishers (North-Holland).
- P. Tanner and W. Buxton (1985). Some issues in future user interface management system (UIMS) development. In G. Pfaff, editor, *User Interface Management Systems*, Springer Verlag, Berlin.
- TeleUSE Reference Manual* (1989). 1.0 edition, TeleSoft AB, Linköping, Sweden.
- L. Weitzman (1988). *Designer: A Knowledge-Based Graphic Design Assistant*. Technical Report ACA-HI-017-88, MCC, Texas.



## **Study 2**

# KNOWLEDGE-BASED EVALUATION AS DESIGN SUPPORT FOR GRAPHICAL USER INTERFACES

Jonas Löwgren

Dept. of Computer and Info. Science  
Linköping University  
S-581 83 Linköping, Sweden  
jlo@ida.liu.se

Tommy Nordqvist

Nat. Defense Research Est. (FOA 531)  
P.O. Box 1165  
S-581 11 Linköping, Sweden

## ABSTRACT

The motivation for our work is that even though user interface guidelines and style guides contain much useful knowledge, they are hard for user interface designers to use. We want to investigate ways of bringing the human factors knowledge closer to the design process, thus making it more accessible to designers. To this end, we present a knowledge-based tool, containing design knowledge drawn from general guideline documents and toolkit-specific style guides, capable of evaluating a user interface design produced in a UIMS. Our assessment shows that part of what the designers consider relevant design knowledge is related to the user's tasks and thus cannot be applied to the static design representation of the UIMS. The final section of the paper discusses ways of using this task-related knowledge.

**Keywords:** user interface evaluation, design support, guidelines, style guides.

## INTRODUCTION

The need for human factors knowledge in the design of information systems has been increasingly acknowledged over the last decade. It is by now unanimously agreed that issues such as usability, consistency and overall appreciation can all be facilitated by the application of human factors expertise to the design process.

A popular medium for the propagation of human factors knowledge has been documents containing general or environment-specific design rules. The former kind

is called *guidelines*; the latter *style guides*. The knowledge in these documents is characterized by being supported by general consensus, often validated through experience or controlled experiments and by being exhaustive. Style guides in particular often represent *de facto* industrial standards and the knowledge is often prescriptive rather than suggestive (i.e., "must" rather than "should"). However, several objections have been put forward to this type of knowledge dissemination. Hammond *et al* [9] point to the problem that guidelines have to be general in order to be applicable in most situations, which in turn makes them too general for any specific situation. The context dependencies present in real design problems are also hard to capture in general guidelines. Hence, human factors knowledge in the form of guidelines can be hard for designers to use in their daily work. There is also some empirical evidence to support this conclusion: de Souza and Bevan [4] showed by means of an experiment that designers had difficulty in interpreting over 90 percent of the general guidelines given for a design task. Tetzlaff and Schwartz [10] reported similar findings.

## The Need For Support

It would appear that guidelines and style guide documents are inefficient ways of communicating human factors knowledge to the designer. Not only are the documents difficult to use, but it is also hard for the designers to remember to apply all relevant rules to a particular design problem. Our answer to this dilemma is to investigate ways of bringing human factors support closer to the design process, thus making the human factors knowledge more accessible and operative. The approach we have chosen is to augment the design and implementation environment of a User Interface Management System (UIMS) with a knowledge base containing human factors knowledge. This knowledge is used to evaluate the design built in the UIMS on the designer's request, yielding what is known as a *critiquing* system. The aim is to provide *formative* evaluation, which is defined as evaluation during system design, intended to

<sup>1</sup>Both authors contributed equally to the contents and the presentation of this work.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

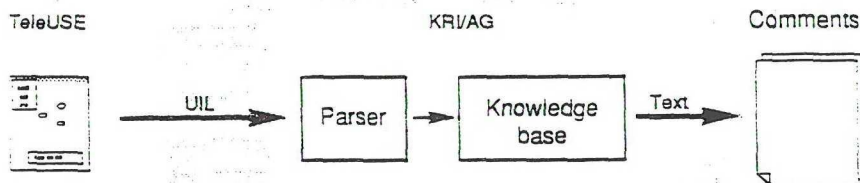


Figure 1: The overall architecture of the current implementation.

provide feedback for subsequent design iterations [10]. We have chosen to address user interfaces built with the Motif<sup>TM</sup> toolkit since it is one of the emerging *de facto* standards in the software industry, and since the Motif Style Guide [15] contains much design knowledge on a detailed level.

### Related Work

Automatic evaluation of user interface design representations has been investigated for at least ten years; some early examples include Reisner's work [16] on assessing simplicity and consistency of commands represented in a BNF grammar and the work by Bleser and Foley [1] on evaluation of a grammar representation with respect to high-level design issues. A more recent approach, using knowledge-based techniques, is illustrated by the Framer system by Lemke and Fischer [11] which is a user interface design environment containing a critiquing system based on general design knowledge. The major differences between Framer and our work are our emphasis on improving upon available knowledge sources, particularly guidelines and style guides, and our notion of runtime evaluation as described below.

### ARCHITECTURE

The overall architecture of the KRI/AG prototype system is illustrated in figure 1. The current design environment is the widget editor of the TeleUSE UIMS from Telesoft, which runs under the X Window System<sup>TM</sup>. This editor, called the VIP, is a graphical widget builder where the various Motif widgets are used as building blocks in constructing a user interface. The design representation can be stored in UIL [5], a *de facto* standard representation for widget instances, which is the language understood by the evaluation system. It is important to point out that the UIL representation covers only the "static" user interface, i.e., the components which can be designed in the UIMS prior to execution of

the system. This includes buttons, menus, forms, etc. but typically excludes the appearance and behaviour of the domain objects. In the current prototype, the VIP runs on a Sun SPARCstation.

KRI/AG is implemented in Epitool, a hybrid expert system shell from Epitex featuring an object-oriented concept representation with inheritance as well as a rule language for writing forward or backward chaining rules. The UIL representation of the user interface to be evaluated is transferred to the DECstation on which KRI/AG runs and parsed into the internal object representation of Epitool. The knowledge base is then applied to the user interface representation, possibly yielding a number of comments on the design.

### The Knowledge Base

As stated above, the knowledge base of KRI/AG is built mainly from publicly available sources such as guidelines collections (e.g., Smith and Mosier [18] and Brown [3]) and the Motif Style Guide [15]. The reason for this is that in an earlier project [13], we performed knowledge acquisition almost exclusively along more conventional lines (i.e., eliciting knowledge from a user interface evaluation expert). In this project, we wanted to represent the human factors knowledge of the public sources in a more accessible form.

It can be noted that there is still a fair amount of human expertise represented in the process. One of us, who did most of the interpretation of the guideline documents, is an expert in user interface evaluation. We also used a scenario technique, where an independent expert was given 20 examples of user interface design flaws together with our tentative comments upon them. This material formed the basis for the knowledge acquisition session with the expert, and the results served to validate our analyses of the guidelines. To summarize, the task of building a knowledge base from guidelines and style guides is by no means trivial or mechanical.



```

Rule PopupMenuTitle in OSF_Motif Is
ForAll ?inst WhichIs Motif$XmPopupMenu;
If
  Not (Class(MenuItem(?inst, 1) = "XmLabel"))
Then
  MakeComment("The popup menu ", ?inst.Name,
    "does not have a title. Every menu should
    have a unique title placed at the top.
    (Motif Style Guide 4.2.3)");
End;

```

Figure 2: A rule from the KRI/AG knowledge base. Note the typical structure where the user interface representation is examined with respect to design flaws (in this case a missing title in a popup menu).

In its current state, the knowledge base of KRI/AG comprises about 70 rules and 30 functions<sup>1</sup>. The technique used for producing the comments is what is known as *analytical critiquing* [6], which means that the proposed solution (in this case, the user interface design) is analyzed with respect to possible flaws. The alternative is the *differential* approach, where the critiquing system generates its own solution to the problem and compares it with the one proposed by the user, pointing out the differences and deviations. We have argued elsewhere [12] that the domain of user interface design in general is not eligible to a differential treatment: the reasons are mainly that the problem is not well-defined and that there are many examples of multiple solutions with equal validity.

Figure 2 shows an actual rule from the KRI/AG knowledge base, illustrating the type of knowledge used in analytical critiquing systems. The level of the knowledge is obviously limited to what can be represented in UIL, viz. the layout and composition of widgets. This means that the level of evaluation is accordingly limited to the levels of presentation and syntax.

Table 1 shows a more detailed view of the current contents of the knowledge base in KRI/AG. We can see that roughly 60% of the knowledge base consist of general rules, constructed from the guideline documents. The reason for this is mainly that we spent more time on analysing those documents and validating the results. The remaining 40% consist of Motif-specific knowledge. Almost all of it is concerned with menu layout, organization and interaction. This is an important part of

	General	Motif
Graphical layout	10	4
Menu layout	16	28
Menu dialogue	15	7
Other dialogue	20	0

Table 1: A breakdown of the topics covered by the current KRI/AG knowledge base and the distribution over general guidelines and Motif Style Guide rules. Numbers are percent of the total knowledge base.

the Motif Style Guide, but not as dominant as it might appear in our knowledge base. We expect to be able to extend the Motif-specific part of the knowledge base as the analysis of the Style Guide proceeds.

## EXAMPLE

This section illustrates the use of KRI/AG to evaluate the user interface of an actual application, built using the TeleUSE UIMS.

### The Tactical Map Editor...

Figure 3 illustrates the appearance of the application we chose for evaluation. It is an editor for tactical maps in a military setting, developed at FOA 531. The main window shows a detailed view of an area, with a static map overlayed with symbols representing military units and borders between the areas of responsibility for the different units. The small window to the left is an overview of the whole area covered by the geographic data available. The square indicates the area currently presented in the main window.

Six tools (shown beneath the overview window) are available for the manipulation of the overlay symbols on the map: **Create Unit**, **Create Border**, **Create Position**, **Clear**, **Move** and **Edit** ("Förband", "Gräns", "Position", "Radera", "Flytta" and "Redigera", respectively). The form in the lower left corner is used to inspect or edit attribute values for the selected unit and to provide new values when a new unit is created. Seven of the nine fields are actually option menus which pop up on a mouse click, giving the user a choice of all permissible values for the field in question.

There are two pulldown menus containing global commands. The left one ("Arkiv") is the typical File menu, containing commands such as **Load**, **Save** and **Exit**. The right one contains commands to set various presentation properties. Finally, the text field at the bottom right is used to present various kinds of textual information.

<sup>1</sup>The Editool environment uses the concept of *functions* to denote procedural domain knowledge units which return values. For purposes of knowledge base size assessments, they may be considered equal to rules.

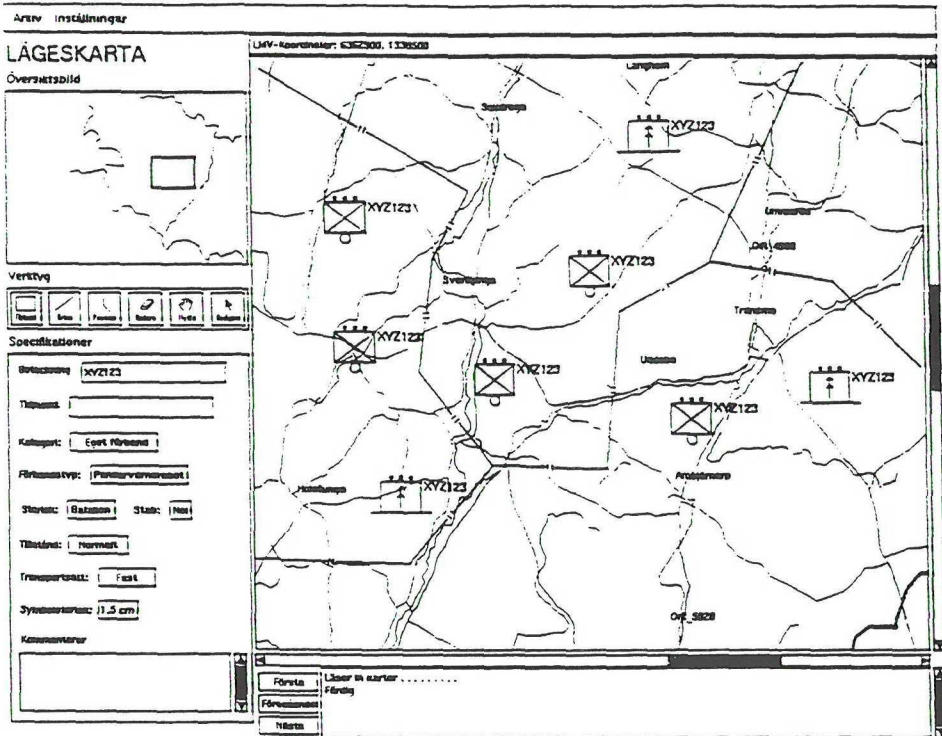


Figure 3: The user interface of the tactical map editor.

### ...Evaluated

We used VIP to generate a UIL description of the user interface shown in figure 3 and passed the description to KRI/AG. The system generated a number of comments in Swedish, which we present below (translated to English and aggregated since the system, for example, generated the same comment for each of the seven option menus).

- The [text field at the bottom] does not have a label. There should be a label or header above or to the left of it. (Smith and Mosier 1.4:5 and 1.4:17)
- The text fields in [the dialog boxes which appear when the user selects **Save As** or **Open**] do not have default values. (Smith and Mosier 3.1.2:3 and 1.8:1)
- The items in the option menus are in alphabetical order. If there is a logical order, it should be used. Otherwise, if the frequency of use is known, it should be used in ordering the items. (Smith and Mosier 2.5:16-17)
- There is no **Help** menu in the menu bar. Every application should have a **Help** menu. The recom-

mended standard menus in the menu bar are **File**, **Edit**, **View**, **Options** and **Help**, in that order. (Motif Style Guide p. 7-42)

- The menus in the menu bar do not have mnemonics. Specifically, the **File** menu should have the mnemonic **F**. (Motif Style Guide 3.3.3, pp. 7-42, 7-46)
- The items in the **File** menu are not standard. The following items should be in the menu: **New**, **Open...**, **Save**, **Save As...**, **Print** or **Print...**, **Close** and **Exit**. (Motif Style Guide p. 7-23)
- None of the items in the menus of the menubar have accelerators. It is a good idea to use accelerators for the most frequently used items. (Motif Style Guide 3.3.2, 4.2.3, pp. 7-3, 7-4)

By empirical assessment of our previous project [13], we found that references to the guidelines documents were central to acceptance of the evaluation tool. In that system, there was an option which displayed the relevant guideline for each comment generated during evaluation. In KRI/AG, we provide only a reference to the



source documents. It would, however, be straightforward to provide an option to present the actual guideline texts and pictures online.

## DISCUSSION

Recall that the motivation for our work was the observation that guidelines and style guides seem to be hard to use in practice. The KRI/AG prototype described above represents a first step towards facilitating the use of these knowledge sources in design. This section discusses two of the most important issues raised by our approach: *how to support designers*, and the *appropriate level of evaluation for a design support tool*.

### How To Support Designers

In their study of the use of guidelines for user interface design, Tetzlaff and Schwartz [19] concluded that since guidelines were found hard to use, the dependence upon them should be minimized. Instead, toolkits and interactive examples of good designs should be used and the role of the guidelines should be mainly to provide information which is intrinsically unavailable through those vehicles. The similar idea can be found in implemented form in the Framer design environment [11] where a library of initial design skeletons is available to provide starting points for the designer.

Widget builders such as the VIP actually represent a move towards the idea of reusable examples, since some of the widget templates in the modern toolkits are fairly complex and come with a good deal of encapsulated appearance and behaviour. Prominent examples are the FileSelectionDialog and other ready-to-use popup dialogues in Motif. However, as many examples from practice show, it is still not impossible to construct user interfaces which violate general design rules and toolkit-specific style rules. These violations, of course, impair usability as well as the overall impression of the produced system. A particular issue when toolkit-specific style rules are concerned is inter-application consistency (recall the missing mnemonics and Help menu in the map editor example above). We believe that a good way of reducing these violations is to augment the design environment with knowledge of general design and specific style, as demonstrated by the KRI/AG system.

**When And How?** The current system prototype reviews the design only when the designer explicitly requests comments. This is contrary to other work in the area of knowledge-based design environments. In particular, Lemke and Fischer [11] report that their initial Framer system worked in the same way as KRI/AG. They found that it was sometimes hard for the system to give meaningful comments on a design, since the designer had chosen a suboptimal path in the design space early on and pursued it too far before submitting the design for comments. When they reworked

the critiquing module to continuously monitor the designer's work and react as soon as it found anything worth commenting, the resulting system was "more effective." Unfortunately, they do not report any controlled experiments. We regard the issue of critiquing strategy to be a question in need of empirical studies, and we hope to be able to carry out such studies in the near future. This can be done either by implementing an active design evaluation module or by Wizard-of-Oz techniques.

### Level Of Evaluation

During the evaluation of our previous user interface evaluation tool [13], we found that evaluation on the level of user tasks was highly desirable. This can also easily be established by examining the knowledge sources used for KRI/AG: both the general guidelines and the Motif Style Guide contain many rules concerning the user's behaviour and tasks. One example from Motif [15, p. 4-21] is the following.

Applications should provide accelerators for frequently used menu items. In general, accelerators should not be assigned for every menu item in an application.

The crucial word here is "frequently", since there is no way of determining by analysis of a Motif design representation whether a menu item is going to be used frequently. This means that rules such as the one above cannot be properly implemented in an evaluation tool of the KRI/AG type. What we had to do there (compare the seventh comment to the map editor example above) was to leave the judgment to the designer.

There are in general two ways of achieving evaluation on the task level. One is to use a rich design representation where user tasks and domain semantics are specified in the design tool. An example of a design environment based on this idea is UIDE by Foley *et al* [7] where the designer specifies the semantics of the user actions and the domain objects. The other way is to collect and analyze logs from actual tests of the user interface under construction. We believe that the second method, which we call *runtime evaluation* or RTE, is preferable since it is more compatible with existing design tools, does not introduce additional complexity for the designer and relies less on *a priori* assumptions. The rest of this section is devoted to a discussion of how to combine runtime evaluation with the design-time evaluation techniques described so far.

**Runtime Evaluation.** Other researchers have touched upon the subject of logging interaction and automatically evaluating the resulting data. Siochi and Hix [17] started from the hypothesis that repetitions indicate interesting user behaviour. In a small study, they let subjects use a test interface, collected logs of all the

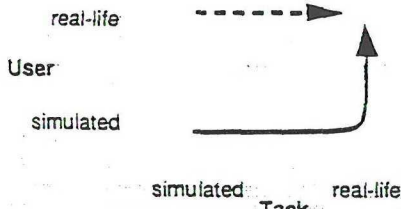


Figure 4: The space of RTE. The solid line shows the approximate time order of different evaluation forms in a traditional waterfall approach to software development, whereas the dashed line illustrates an extremely user-oriented prototyping approach.

interactions and also determined two major usability problems by observation. When their system analyzed the logs with respect to maximal repeating patterns, the same problems were indicated. Olsen and Halversen [14] had earlier shown how logging could be integrated in a UIMS architecture to give metrics concerning the use of different commands. Before we discuss the technical feasibility in our setting, let us introduce a conceptual framework which is intended to relate the idea of RTE to different philosophies of software development.

We propose a two-by-two matrix of RTE where the dimensions are the *task* and the *user* involved in the test situation. The task can be either *simulated* or *real-life*. Simulated tasks can be defined based on the requirement specification or the activity analysis, depending on whether they have been formulated, or they can consist of general handling of the user interface without consideration of the particular tasks in the target environment. Simulated tasks can be tested in the development environment. Real-life tasks, on the other hand, have to be the real tasks that the system is intended to support. Moreover, the tests have to take place in the delivery environment.

On the user dimension, we have *simulated* and *real-life* users. The simulated user can be the original developer, a customer representative, a subject person chosen at random or anyone else who is willing to pretend being the intended user of the system. If a user analysis has been produced earlier in the project, it may be used to aid the "impersonator." A real-life user, as the term implies, is one of the users for whom the system is intended.

This matrix can be used to relate the different forms of RTE to different software development philosophies,

as shown in figure 4. Two examples of different philosophies are illustrated, with the traditional waterfall approach (denoted by a solid line) progressing from simulated tasks through real-life tasks with simulated users and then, in the test phase of the project, to real-life tasks and users. The other example (the dashed line) is an extremely situated design approach where prototyping and development with real-life users are paramount (see, for example, Bodker [2]).

A general property of the matrix is that the cost associated with different forms of evaluation increases with the degree of realism. For example, it is more expensive (in terms of money, time or effort) to carry out a test with real-life tasks than with simulated. It also seems to be the case that the degree of realism is transitive, i.e., a user interface property which can be tested with simulated users or with simulated tasks can also be tested with real-life users or tasks. The reverse relation does not obviously hold.

**Properties To Evaluate.** The TeleUSE architecture is based on the Seeheim model [8] and uses its own event mechanism and language, called D, for synchronizing the user interface with the functionality of the application. We can expect to be able to collect logs consisting of D events as well as the X events which give low-level information such as keyboard input and mouse position. The idea is then to evaluate these logs using a combination of knowledge-based and algorithmic techniques and to generate comments on the user interface design in analogy with KRI/AG.

While we performed knowledge acquisition for KRI/AG, we formulated many user interface properties of the kind that could not be assessed in design-time evaluation. We will now present some of those properties and indicate how they could be measured using the logs collected during user interface testing.

1. *Long sequences for common operations.* In the way demonstrated by Siochi and Hix, the system can detect repeating sequences and comment upon them if they occur often enough. A case which requires particular attention is when the user has to traverse submenus to reach the desired (frequent) operation.
2. *Switching of interaction techniques during the same task.* If the user is found to be switching from, say, keyboard to mouse and then back again for the same input focus and within a small amount of time, it is worth commenting.
3. *Syntactical inconsistency.* In a graphical user interface, it is desirable that the manipulation syntax is the same throughout the system. This means to consistently use either Object-Command syntax (first select an object or several objects and then



	real-life	(4)	4 5
User			
	simulated	2 3	1
	simulated		real-life
	Task		

Figure 5: Our sample properties inserted in the evaluation space. An entry in parentheses means that the property can be evaluated to some extent.

apply an operation on it or them) or Command-Object (operation first, then objects). To analyze this, the system can assume that the operations are invoked via the static part of the user interface (buttons, menus, etc.) which is constructed at design-time, whereas the objects of the operations are application-driven.

4. *Detecting errors and help requests.* If the design adheres to Motif standards, errors and help requests can be detected by looking for WarningDialogs and use of the **Help** menus, respectively. Otherwise, the D events corresponding to help request and application errors would have to be tagged in a special way. In both cases, comments showing the dialogue states where more than an average of errors or help requests occurred would be valuable for the designer.
5. *Accelerators for the most frequent operations.* It would be easy to count the number of times different menu items are used and then check for accelerators for the most frequent ones, pointing out possible deficiencies to the designer. Similarly, the system could suggest that a frequently used button in a form containing text input components should be made the default. This would in effect assign the carriage return key as an accelerator for the frequently used button.

To put these properties into the context of user interface development, let us insert them in their cheapest possible places in the RTE matrix, as shown in figure 5. A number in parentheses means that the property can be addressed to some extent. For example, the detection of errors and help requests (property 4) can be done with respect to syntactical errors for a simulated task,

but in order to detect domain errors, a real-life task is needed. If we would now draw a time order arrow reflecting the software development approach used, the resulting picture would show us when we can expect to evaluate the different properties.

## SUMMARY

We have shown by means of the KRI/AG prototype how some design knowledge, general guidelines as well as toolkit-specific style guides, can be applied to evaluate a user interface design produced in a UIMS. We believe this to be a valuable step towards bringing human factors knowledge closer to the design process, thus making it more accessible and operative. Our analyses, however, indicate that much of the design knowledge can be applied only by taking into account the actual use situation. We have outlined how data collected during tests of the produced prototype can be used to bring also this use-related design knowledge to bear and presented a framework for relating these tests to the software development approach in use.

## ACKNOWLEDGMENTS

The authors want to thank Peter Ericsson from FOA 531 and Lennart Olsson from Enator for implementing the current KRI/AG prototype, and Staffan L f from FOA 531 for many excellent design ideas and creative discussions. Prof. Sture H gglund read an earlier version of this paper and gave many useful comments, for which we are grateful. We would also like to thank Nils-Erik Gustafsson from Ellemtel, our expert, for giving us the time we needed with him.

This work has been funded by the National Defense Research Establishment (FOA) and the Swedish Board for Industrial Development of Information Technology (IT4).

Motif is a trademark of The Open Software Foundation, Inc. X Window System is a trademark of the Massachusetts Institute of Technology.

## REFERENCES

- [1] T. Bleser and J. Foley. Towards specifying and evaluating the human factors of user-computer interfaces. In *CHI'82 Proceedings*, pages 309-314. 1982.
- [2] S. B dker. Through the interface—a human activity approach to user interface design. Lic. thesis DAIMI PB-224, Aarhus University, 1987.
- [3] C. Brown. *Human-Computer Interface Design Guidelines*. Ablex Publishing Corp., NJ, 1988.
- [4] F. de Souza and N. Bevan. The use of guidelines in menu interface design: Evaluation of a draft standard. In D. Diaper, D. Gilmore, G. Cockton,

- and B. Shackel, editors, *Human-Computer Interaction — Interact'90*, pages 435-440. North-Holland, 1990. Participants Edition.
- [5] Digital Equipment Corp. *Guide to the XUI User Interface Language Compiler*. 2.0 edition, 1988.
  - [6] G. Fischer, A. Lemke, T. Mastaglio, and A. Morch. Using critics to empower users. In *CHI'90 Proceedings*, pages 337-347, 1990.
  - [7] J. Foley, W. Kim, S. Kovačević, and K. Murray. Defining interfaces at a high level of abstraction. *IEEE Software*, pages 25-32, January 1989.
  - [8] M. Green. Report on dialogue specification tools. In G. Pfaff, editor, *User Interface Management Systems*, pages 9-20. Springer Verlag, Berlin, 1985.
  - [9] N. Hammond, M. Gardiner, B. Christie, and C. Marshall. The role of cognitive psychology in user-interface design. In M. Gardiner and B. Christie, editors, *Applying Cognitive Psychology to User-interface Design*, chapter 2, pages 13-53. John Wiley & Sons, Chichester, 1987.
  - [10] S. Howard and M. D. Murray. A taxonomy of evaluation techniques for HCI. In H.-J. Bullinger and B. Shackel, editors, *Human-Computer Interaction — Interact'87*, pages 453-459, 1987.
  - [11] A. Lemke and G. Fischer. A cooperative problem solving system for user interface design. In *Proceedings Eight National Conference on Artificial Intelligence (AAAI-90)*, pages 479-484, 1990.
  - [12] Jonas Löwgren. *Knowledge-Based Design Support and Discourse Management in User Interface Management Systems*. Ph. D. dissertation, Linköping University. March 1991. Linköping Studies in Science and Technology # 239.
  - [13] Jonas Löwgren and Tommy Nordqvist. A knowledge-based tool for user interface evaluation and its integration in a UIMS. In D. Diaper, D. Gilmore, G. Cockton, and B. Shackel, editors, *Human-Computer Interaction — Interact'90*, pages 395-400. North-Holland, August 1990. Also as research report LiTH-IDA-R-90-15.
  - [14] D. Olsen and B. Halversen. Interface usage measurements in a user interface management system. In *Proc. ACM SIGGRAPH Symposium on User Interface Software (UIST'88)*, pages 102-108. ACM Press, 1988.
  - [15] Open Software Foundation, Cambridge, MA. *OSF/Motif Style Guide*, 1988. Revision 1.1.
  - [16] P. Reisner. Formal grammar and human factors design of an interactive graphics system. *IEEE Trans. on Software Engineering*, SE-7(2):229-240, March 1981.
  - [17] A. Siochi and D. Hix. A study of computer-supported user interface evaluation using maximal repeating pattern analysis. In *CHI'91 Proceedings*, pages 301-305, 1991.
  - [18] S. L. Smith and J. N. Mosier. Guidelines for designing user interface software. Report ESD-TR-86-278, Mitre Corp., Bedford, MA, 1986.
  - [19] L. Tetzlaff and D. Schwartz. The use of guidelines in interface design. In *CHI'91 Proceedings*, pages 329-333, 1991.

## **Study 3**



# TUNE: A Tool for User Interface Evaluation

Tommy Nordqvist, Defence Material Administration and Luleå University of Technology, Sweden

## Abstract

*The present paper describes and discusses a prototype tool (TUNE) for computer-supported evaluation of guideline and styleguide compliance in user interfaces. The aim of the tool is to facilitate the use of human factors knowledge, in the form of guidelines and style guides, GLSG, when developing user interfaces.*

*After discussing the increased interest in GLSG compliance and reasons for computer-support in this area, TUNE is presented shortly, together with experiences from practical use. With these experiences as a basis the possible benefits in using the tool to evaluate GLSG compliance is discussed. The paper finally presents future development of TUNE, for example implementing more GLSG.*

## 1. Introduction

When developing interactive information systems, great efforts are focused on development of the user interface. A study by Myers and Rosson [7] for instance, points to the fact that about 50% of the software code relate to the user interface.

This has resulted in much work focused on the development of user interfaces within the research community as well as within large vendor companies. For example documents containing general or platform-specific GLSG's [5;9;13] have been developed to support development of user interfaces.

Unfortunately, such documents have proven to be difficult to use for designers [1;6;14]. One reason is that many of the GLSG documents are very comprehensive, which makes it difficult for designers to find specific GLSG's. Another reason is that GLSG's are, in many cases, difficult to interpret in the practical design situation. Accordingly, very few designers use existing GLSG's collections in practice [12;15].

Researchers have tried to solve these problems in many ways. One example is formulating a smaller amount of usability heuristics to be used in heuristic evaluation.

Another example is a knowledge-based system, integrated with a user interface management system (UIMS), enabling automatic evaluation of a user interface compliance with GLSG's [3;4]. A third example is hypertext checklists for evaluation of a user interface compliance with GLSG's [11].

In our business, we have experienced an increased interest in user interface compliance with GLSG. This interest is often expressed in a formal requirement to conform to a specific GLSG, particularly 'The Windows Interface: An Application Design Guide' [5]. A requirement of this kind makes it difficult to use only heuristic evaluation, because we can not guarantee that the user interface comply with specific GLSG's by using this method. Nor can we use knowledge-based design support integrated with an UIMS, since UIMS tools are rarely used in our system development projects. Hypertext checklists are also inconvenient because of the time needed for evaluation of a user interface compliance with all GLSG in above mentioned documents, even if the work is supported by a hypertext tool.

At the same time we need computer-support to evaluate user interface GLSG compliance. The primary reason for this is that manual evaluation of user interface GLSG compliance is a time-consuming activity. Another reason is our experience that it is a very difficult, if not impossible, task to evaluate that all user interface elements comply with defined GLSG's when evaluating manually. A third reason is that we are often developing tactical support systems, which makes consistency especially important (see also, [8], p. 7 and p. 132). The computer-support should also support iterative design, otherwise it will not be used.

From above reasons, a number of goals for a tool for evaluation of GLSG compliance have been defined:

1. Reduce the time needed for evaluation of GLSG compliance,
2. support the task of evaluating all user interface elements,
3. enhance consistency in user interfaces,
4. support iterative design of user interfaces.

In an attempt to fulfill these goals we have developed a prototype tool for evaluation of GLSG compliance, TUNE (Tool for User Interface Evaluation).

## 2. The Evaluation Tool: TUNE

### 2.1. User Interface Elements Evaluated

The TUNE prototype support evaluation of GLSG compliance of user interface elements frequently present in the computer systems we are developing. The user interface (ui) elements selected were pull-down menus, menu items, mnemonics, shortcuts, dialog boxes and buttons (OK and Cancel buttons in dialog boxes). (About 70% of the GLSG's in [5] concerning above mentioned user interface elements are implemented in TUNE for now).

The aspects to be evaluated were;

- existence of ui elements,
- uniqueness of menu titles, mnemonics and shortcuts,
- correspondence between the title of a dialog box and the name of the selected menu item,
- order of ui elements (menus, menu items),
- proper action when selecting ui elements.

Dynamic tests are used to evaluate that user interface elements behave as specified in the GLSG. Examples of dynamic tests are:

- function of menu items, mnemonics, short cuts and buttons,
- presentation of dialog boxes when selecting menu items followed by three dots.

The rules for performing the tests are either located in the GLSG database or in the test programs. In the GLSG database are rules that could be selected depending on the application evaluated. In the test programs are rules that are seen as generic for all applications. Examples of rules located in the GLSG database are; mandatory menus in the menu bar, mandatory menu items. Examples of rules located in the test programs are; no space in a menu name, unique menu items and mnemonics within a menu.

With the rules as a basis TUNE evaluates the user interface elements in the application by reading the information concerning ui elements using windows standard functions and check that the static properties of the user interface elements are as specified in the GLSG's. For GLSG's concerning dynamic behavior TUNE activate the user interface elements and check that they behave as specified in the GLSG. Deviations from the GLSG's are then noted in the result file.

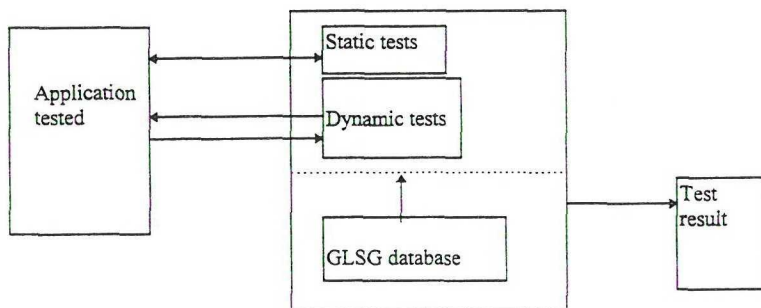


Figure 1: Overall architecture of TUNE

### 2.2. Architecture

The overall architecture of TUNE is presented in Figure 1 above. TUNE consists mainly of test programs (implemented in C++) for static and dynamic tests, and a GLSG database. Static tests are used to evaluate that user interface elements exist and their appearance. Examples of static tests are:

- existence of menus and menu items,
- appearance of mnemonics and short cuts for menu items.

### 2.3. The procedure for using TUNE

The procedure for evaluation of GLSG compliance using TUNE is illustrated below:

1. Start of application to be evaluated.
2. Start of TUNE.
3. Selection of test programs and GLSG's from the database.
4. Start of test.
5. Evaluation of user interface elements. (Here it is possible to choose between interactive or batch-mode evaluation.)
6. Opening test report.

3. Evaluation of GLSG Compliance: An example

3.1. Description of the evaluated application

The evaluated computer system was a military application consisting of a main window and a great number of menus. In the main window, different types of forms are presented for receiving, writing and sending data messages from/to different combat net radios out in the field. The menus are partly used to present the different forms to be used in the main window, partly to define properties of the system to support work with the application. Figure 2 below, illustrates the appearance of the application. Our task was to evaluate the user interface compliance according to the GLSG's implemented in TUNE. The result from the evaluation was going to be used as input for the final implementation of the application.

cases adjusted to the GLSG's in the original volume. The result report is structured in the following way: First, the evaluated interface elements are presented as they appear in the evaluated application. Then the comments generated in the evaluation of the user interface elements are presented.

The result from the evaluation was presented to the designers of the evaluated application. The reactions from them were that almost all of the comments were actually design flaws when developing the prototype. They also commented the fact that many of the design flaws were repeated. For example, when designing the user interface, they had forgotten the GLSG regarding shortcuts and hence all the shortcuts deviated from the GLSG. The only objection to the evaluation result was that in their opinion Save form was a better name than Save. After discussing with the designers their conclusion still was that the designer-selected name of the menu item in a better way reflected the functionality. However, the designers realized the importance of consistency with other applications in a

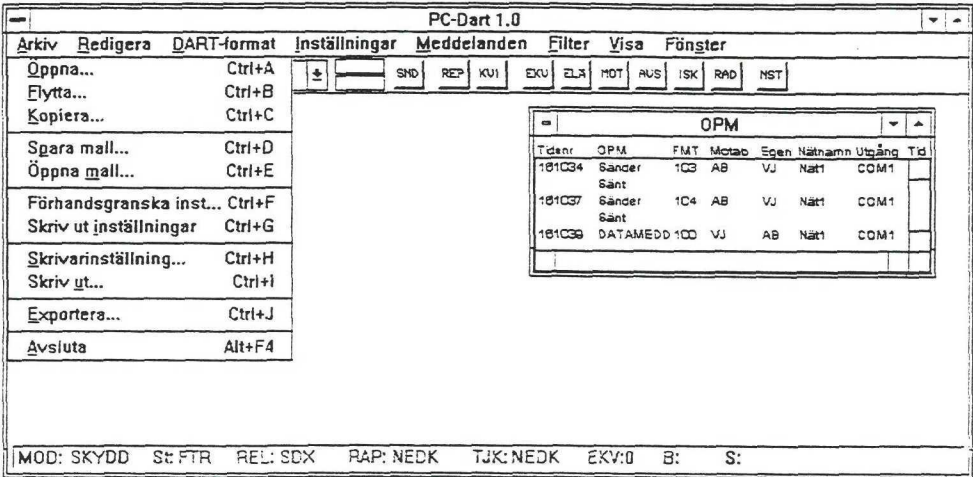


Figure 2: The appearance of the application evaluated.

3.2. The evaluation

The evaluation of the application's user interface GLSG compliance was performed as described in section 2.3.

3.3. Results.

The evaluation resulted in 85 deviations from the GLSG's defined. An excerpt from the result report, generated from the evaluation, is presented in Figure 3 below. The results are translated into English and in some

military situation and chose the name Save. The result from the evaluation was then introduced into the design specifications for the implementation of the final system (Further aspects of evaluating the application were handled by human evaluators using the GLSG's not yet implemented.)

4. Further applications evaluated

TUNE has also been used to evaluate two other applications. One was an application for presenting and



## File menu

Menu item:	Shortcut:	Mnemonic:	Menu item in Swedish:
Open...	Ctrl+A	O	Öppna...
Move...	Ctrl+B	M	Flytta...
Copy...	Ctrl+C	C	Kopiera...
Save form...	Ctrl+D	F	Spara mall...
Open form...	Ctrl+E	P	Öppna mall...
Control preview...	Ctrl+F		Förhandsgranska inst...

*Presentation of the user interface elements in the application evaluated.*

## Comments:

Menu item New is not found.  
Shortcut for Open... is not as defined in GLSG.  
Use of the mnemonic for Open... does not result in any action.  
The dialog box presented when Open form... is selected has no OK button.  
When selecting Control preview... no dialog box is presented.

*Comments generated in the evaluation of the user interface elements.*

**Figure 3:** An excerpt from the result report.

handling geographical information. Another was an application for presenting attacking enemy aircraft in an anti-aircraft setting. The results from these evaluations were mainly the same as for the evaluation presented here.

## 5. Discussion

The work presented above illustrates how to automatically evaluate GLSG compliance in user interface design. Even though TUNE only focused on above mentioned user interface elements, the work points to the potential of automatic evaluation of user interface elements and thus facilitating the use of human factors knowledge in user interface development.

### 5.1 TUNE and the defined goals

To evaluate TUNE against the goals listed earlier following study were conducted. In parallel with the TUNE testing of the three applications, manual evaluation was also accomplished by three usability experts. (The number of usability experts was determined with [15] as a basis.) The data collected in manual evaluation and TUNE evaluation were:

- time used for performing the evaluation,
- number of evaluated user interface elements,
- number of identified deviations from GLSG's,
- interviews with developers who had used TUNE when developing user interfaces.

Time used (minutes) was registered by the usability experts and by the TUNE operator. (Time for preparation of the manual evaluation and TUNE evaluation was not included in the registration.) Number of evaluated user interface elements was registered through analysis of the reports from the usability experts and from TUNE. Number of identified deviations was registered by letting the usability experts note every deficiency on paper, which was then analyzed, and by studying the result report from TUNE. The interviews were performed individually for the three developers. It is important to notice here that when we are talking about time and number below, we only discuss in terms of the user interface elements mentioned earlier. Also the status of development for the different applications are very different, so the numbers given in the figures should not be compared over applications.

Goal 1): Reduce the time needed for evaluation of GLSG compliance. In Figure 4 below the time used for manual evaluation and TUNE evaluation of the three applications is presented. Time used for manual evaluation is here presented as mean values for the three evaluators.

	A	B	C
Manual	115	21	135
TUNE	20	4	27

**Figure 4:** Time used, in minutes, for manual evaluation and TUNE evaluation for the three applications.

As can be noticed in Figure 4, time used for TUNE evaluation is about 20% of the time used for manual evaluation. It is therefore possible to conclude that TUNE reduces the time needed to evaluate GLSG compliance.

Goal 2): Support the task of evaluating all ui elements in the application. In Figure 5 below, number of evaluated user interface elements is presented together with the total number of user interface elements for each application. It should be noticed here that when we refer to the total number of user interface elements we mean those elements mentioned in section 2.1. Number of evaluated user interface elements is here presented as mean values for the three usability experts.

	A	B	C
Manual	231	38	290
TUNE	252	36	305
Total number	255	39	308

**Figure 5:** Number of evaluated ui elements together with total number of elements.

As can be noticed in Figure 5, TUNE evaluation results in that more user interface elements are evaluated and also that almost all user interface elements are evaluated. The reason for the difference between TUNE and the total number is that we are not evaluating the menu item Exit (together with mnemonic and shortcut). Figure 5 also indicate that the difference in evaluated user interface elements between manual evaluation and TUNE evaluation is greater when the application is more complex. A precondition for TUNE evaluating all user interface elements is of course that GLSG are implemented for every element.

Goal 3): Enhance consistency in user interfaces. In Figure 6 below, the number of recognized deviations from GLSG in manual evaluation and TUNE evaluation is reported for the three applications. The reason why we present those numbers is our hypothesis that the more

deviations recognized (and corrected) the more consistent user interface. For the manual evaluation only the deviations that are unique are presented, in other words, if two evaluators has recognized the same deviation it is only regarded as one deviation.

	A	B	C
Manual	69 (222)	33 (73)	58 (97)
TUNE	85	42	83

**Figure 6:** Number of recognized deviations in manual and TUNE evaluation for the three applications.

As can be noticed in Figure 6, the number of recognized deviations is higher for TUNE evaluation then in manual evaluation. This is true only as you have the implemented GLSG's as a basis. Manual evaluators recognize more deviations (the figures within brackets in Figure 6) if you have all GLSG and the usability experts 'design expertise' as a basis.

Goal 4): Support iterative design of user interfaces. Because of the possibility to use TUNE to evaluate a user interface design in progress it is possible to receive comments on different design suggestions. It is also possible to use TUNE as a personal support tool in the design and implementation. For instance, to use TUNE in a design situation where the designer can evaluate a design prototype. In this case it is possible to have comments presented for the designer together with the specific user interface element evaluated. Also, interviews with designers indicate that they consider TUNE as a support for iterative design.

## 5.2. Related work

Of course it is not possible to replace usability experts with a tool for user interface evaluation. Nevertheless, a tool like TUNE could unburden usability experts from the task of evaluating user interface compliance with respect to existence, layout and functionality GLSG's. Thus, making it possible for them to concentrate on more important design issues.

It is important to remember that TUNE only handle a minor part of the usability issue. Compliance with GLSG does not in any way guarantee that an interactive information system will be usable. Therefore, it is important to regard TUNE as a supplement to other usability activities, for example heuristic evaluation, cognitive walkthrough and user testing that are useful for other aspects of the usability issue [2;8].



## 6. Future work

In the future development of TUNE we will focus our efforts on extending the tool in three ways. First, to evaluate additional user interface elements, for instance, other types of menus, different kinds of buttons, list boxes and text boxes.

We estimate that it is possible to implement about 70% of the GLSG's in Microsoft [5]. Concerning the remaining 30%, we think it is necessary to investigate the possibility to make GLSG's more accurate in order to implement them in TUNE. Some of these remaining GLSG's can possibly be defined more accurately. For example, consider following GLSG:

'If an object is so small or thin that pointing or clicking to select it would require extremely precise mouse positioning, provide a hot zone around the object to increase the area where clicking will select the object' ([5], p. 10).

If we supplement this GLSG with:

'The selectable area (of an object) should be at least 0.6 square', [10], it would be possible to implement it in TUNE.

Other GLSG's are more difficult to formulate in the precise way needed. The following GLSG need further information to be possible to implement:

'Each menu item should be represented by a descriptive name or graphic' ([5], p. 83).

To implement this GLSG you need to know the function of the item and to have knowledge about the semantic meaning of the name of the menu item.

Second, we intend to develop tests with OSF/Motif GLSG's as a basis, making it possible to use TUNE when evaluating GLSG compliance in both MS Windows and Motif user interfaces.

It is also important to develop TUNE further so that the evaluator know which GLSG's are (or are not) implemented in TUNE and which user interface elements are (or are not) evaluated in a specific application.

## Acknowledgments

The author wants to thank Per Asplund, Peter Ericsson and Johan Strand, all at Enator Telub AB, for their work at implementing TUNE. Leonard Adelman, Jonas Löwgren and Kjell Ohlsson read an earlier version of this paper and had many insightful comments for which I am very grateful.

This study was founded by the Swedish National Board for industrial and Technical Development. Enator

Telub AB and the Defence Material Administration, Sweden.

OSF/Motif is a trademark of the Open Software Foundation, Inc. Windows is a trademark of Microsoft Corporation. Enator Telub AB is a consultant company within information technology.

## References

- [1] F. De Souza, and N. Bevan. The Use of Guidelines in Menu Interface Design: Evaluation of a Draft Standard. In D. Diaper, D. Gilmore, G. Cockton and B. Shackel Eds., *Human-Computer Interaction - Interact '90*, pp. 435-440. North-Holland, 1990
- [2] R. Jeffries, J.R. Miller, C. Wharton and K. M. Uydea. User interface evaluation in the real world: A comparison of four techniques. *CHI'91 Conference Proceedings*, pp. 119-124. 1991.
- [3] J. Löwgren and T. Nordqvist, A Knowledge-Based Tool for User Interface Evaluation and its integration in a UIMS. In D. Diaper, D. Gilmore, G. Cockton, and B. Shackel Eds., *Human Computer Interaction - Interact '90*, (pp. 395-400). North-Holland, 1990.
- [4] J. Löwgren and T. Nordqvist, Knowledge-Based Evaluation as Design Support for Graphical User Interfaces. In *CHI '92 Proceedings*, pp. 181-188, 1992
- [5] Microsoft, (1992). *The Windows Interface: An Application Design Guide*. Microsoft Press. Redmond, Washington: Author.
- [6] J. N. Mosier, and S. L. Smith. Application of Guidelines for Designing User Interface Software, *Behaviour and Information Technology*, 3, pp. 39-46, 1986.
- [7] B. A. Myers and M. B. Rosson. A Survey on User Interface programming. In *Proceedings of CHI '92*, pp. 195-202, 1992.
- [8] J. Nielsen. *Usability Engineering*. Academic Press. London, 1993.
- [9] Open Software Foundation, *OSF/Motif Style Guide*. Prentice Hall, 1992.
- [10] R. N. Parrish, *Development of Design Guidelines and Criteria for User/Operator Transactions with Battlefield Automated Systems*. Fairfax, Virginia: Synetics (August), 1980.
- [11] G. Perlman. System Design and Evaluation with Hypertext Checklists. *IEEE*, pp. 1187-1193, 1989.
- [12] S. L. Smith and J. N. Mosier. The User Interface to Computer-Based Information Systems: A Survey of Current Software Design Practice. *Behaviour and Information Technology*, 3, pp. 195-203, 1984.
- [13] S. L. Smith and J. N. Mosier, *Guidelines for Designing User Interface Software*. (Technical Report MTR-10090). The MITRE Corporation, Bedford, MA 01730. USA, 1986.
- [14] L. Tetzlaff and D. R. Schwartz. The Use of Guidelines in User Interface Design. In *CHI '91 Proceedings*, pp. 329-333, 1991.
- [15] H. Thovtrup and J. Nielsen, Assessing the Usability of a User Interface Standard. In *CHI '91 Proceedings*, pp. 335-341, 1991.

## **Study 4**

# Computer Support for User Requirement Evaluation in System Development

TOMMY NORDQVIST

**CONTENT**

**ABSTRACT** ..... 1

**1. INTRODUCTION** ..... 1

**2. COMPUTER SUPPORT FOR EVALUATION OF USER REQUIREMENT COMPLIANCE** ..... 5

2.1 USE OF TURE IN THEORY: ..... 7

2.2 PRACTICAL USE OF TURE, A CASE STUDY ..... 8

    2.2.1 *Creating the test-script:*..... 8

    2.2.2 *Evaluation of the computer system:* ..... 9

    2.2.3 *Analysis of the evaluation result:*..... 11

**3. DISCUSSION AND CONCLUSIONS**.....11

3.1 CREATION OF TEST-SCRIPTS: ..... 11

3.2 EVALUATION OF AN APPLICATION ..... 14

3.3 COMPARISON BETWEEN AUTOMATIC AND MANUAL EVALUATION:..... 15

3.4 USABILITY ASPECTS ..... 16

**4. FUTURE WORK**.....18

**REFERENCES**.....

**ACKNOWLEDGMENTS** .....

## **ABSTRACT**

The present paper describes and discusses a computer-supported tool for evaluation of a computer system's compliance with user requirements. The aim of the tool is to support the difficult but important work of validating that defined user requirements are implemented in the computer system developed.

After discussing development of computer systems according to some commercial standards, the need for evaluation of computer systems and reasons for computer-support in this area, the tool TURE (Tool for User Requirement Evaluation) is described. Also, experiences from practical use of TURE are reported. With these practical experiences as a basis, TURE is discussed in relation to the creation of test-scripts, evaluation of a computer system, comparison between automatic and manual evaluation, and usability.

Finally, possible future development of TURE is presented, focused on the need of implementing predefined test-functions, a function for attaching these test-functions to relevant user interface elements, and a function for registering user interaction.



## 1. INTRODUCTION

When developing large computer systems the following activities are usually accomplished (Dix et. al., 1993):

- Requirements specification.
- Architectural design.
- Detailed design.
- Coding and unit testing.
- Integration and testing.
- Operation and maintenance.

Requirements specification or requirements analysis, probably the most important activity when developing computer systems, can be divided in two activities (Palmer, 1990). The first activity is identification and definition of the users' requirements on the future system. Another term for this activity is development of system requirements (IEEE P1233, 1993; MIL-STD-498, 1994). The result from this activity is a system requirement specification (SyRS). The second activity is specification of the requirements on the software, in other words a requirement analysis from a software perspective. Another term for this activity is definition of the software requirements (IEEE std 830-1993; 1994; MIL-STD-498, 1994). The result from this activity is a software requirement specification (SRS). See also Andriole (1990) and Rombach (1990) for a discussion of user requirements and software requirements.

The system requirement specification, which represents the users' basic requirements on the future computer system, is often formulated in natural language. When developing the software requirement specification, these requirements are reformulated in a more formal language, for example object models or flowcharts. The software requirement

specification is then the basis for subsequent activities in the development of the system. When these activities are completed, the resulting computer system is validated against the user requirements specified in the system requirement specification.

Within the system development projects we are working with at Telub AB, our role is primarily to help and support the user/customer when developing computer systems. During requirement specification we are helping the users to identify and define their requirements on the system, in other words helping them to develop the system requirement specification. This system requirement specification is then delivered to the supplier of the future computer system. The supplier develops a software requirement specification, and carries out the subsequent activities in order to develop the computer system. When the system is delivered to the user/customer, validation of the computer system's compliance with the requirements specified in the system requirement specification is usually carried out (ISO 9000-1991, 1991). In this activity we are supporting the user/customer in the validation process.

The system requirement specification has two main objectives. First, it is the basis for the contract negotiations where the user/customer and the supplier will come to an agreement about what to be produced. Second, it is the basis when specifying software requirements. Therefore a system requirement specification should have the following characteristics according to IEEE P1233 (1993):

- "The requirements shall be formulated and organized to define the system's external behavior completely, consistently and unambiguously once the problem is thoroughly understood," (p. 4).
- "Each requirement shall be implementation independent.
- Each requirement shall be stated in such a way so that it can be interpreted in only one way.

- Each requirement shall have the ability to be traced to specific customer statements and to specific statements in the definition of the system given in the SyRS as evidence of the source of a requirement.
- Each requirement shall have the means to prove that the system satisfies the requirements," (p. 13).

The definition of requirements according to these characteristics and with above-mentioned objectives, usually implies that the system requirement specification is characterized by

- A focusing on the complete specification of the external behavior, or the services the computer system should fulfill from the users perspective.
- Requirements often specified in detail and handling the external design of the user interface. Layouts, interaction techniques, menus, dialog boxes, forms and so on, are often specified explicitly in the requirement specification. The reason for this is the need for validation.
- Large number of requirements' in the requirement specification even for small systems.

As a result, comprehensive procedures and specifications regarding the validation of the computer system under development have been developed. These procedures and specifications are for instance called acceptance tests, test specifications or system test specifications. In this paper these procedures and specifications will be called validation specifications. A validation specification for a specific computer system is a detailed check list for every requirement defined in the system requirement specification. If we for instance study following requirement:

"The user should be able to use the computer system to copy earlier created or received messages,"

the validation specification for this requirement could be formulated in the following way:

Do the following:

Select the command "Copy."

Select the type of message to be copied by selecting message type. Do this for every type.

Select message/messages to be copied by clicking on them in the window for messages. Do this for every message and combination of messages.

Type an appropriate name for the message which is copied

Select the catalogue where the copied message shall be saved by clicking on the catalogue name in the directory.

Press the OK button.

Check that:

A "Copy" form is presented.

Messages of selected type is presented in the window messages.

Messages can be selected and that selected messages are indicated or presented in the window for selected messages.

Any name is possible to type and both characters and numbers are accepted.

Appropriate catalogue could be selected as indicated.

Copied message is saved under the chosen name and under the selected catalogue. The "Copy" form is closed and the application returns to the earlier state.



Press the Interrupt button.

The "Copy" form is closed and the application returns to earlier state without any copy saved.

This implies that the possibility to validate the developed computer system's compliance with user requirements in the system requirement specification increase, but also that the validation is both comprehensive and laborious to accomplish. This is particularly the case when performing the validation manually, as in the projects we are working in. If this validation has to be done several times, as in iterative system development or development of several versions of the system, the risk increases that:

- The computer system is not validated against all the user requirements defined in the system requirement specification.
- The validation is very hard to replicate in exactly the same way.
- The validation is being so costly (in time and money) it is not performed for every iteration or version.
- The computer system is not evaluated against the user requirements in the system requirement specification but against the software requirements.

## 2. COMPUTER SUPPORT FOR EVALUATION OF USER REQUIREMENT COMPLIANCE

In an attempt to handle some of the problems in validating a computer systems compliance with user requirements, we have developed a tool called TURE (Tool for User Requirement Evaluation). With TURE it is possible to evaluate a computer system's compliance with user requirements related to the user interface.



The overall architecture for TURE is depicted in Figure 1 below. As a platform for the development of TURE we used WinRunner. WinRunner is developed by Mercury Interactive Corporation and is a tool for developing automatic software tests for Windows, Windows NT, and OS/2 applications. For a detailed description of WinRunner see Mercury Interactive Corporation (1993a, b, c). ( A similar tool is for example WinTest developed by Microsoft).

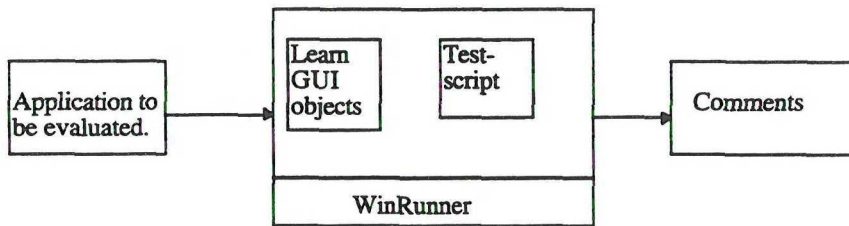


Figure 1: Overall architecture of TURE

TURE consists of the following main parts: The Learn GUI objects function in WinRunner and the test-scripts developed for the applications to be evaluated.

The Learn GUI objects function is used to create a representation of the user interface of the application to be evaluated. This representation includes the logical names of the user interface elements and their physical description. The logical name is for example a button label, a window label or a name defined by the developer of the application. The physical description is a list of attributes that identify the element. This list includes, for instance, type of user interface element (window, dialog box, push-button, menu, menu item), the label of the element, co-ordinates for the element on the screen, and text attached to the element. This list is created with the Learn GUI objects function and can also be supplemented manually.

The test-script consists of functions in a C-like programming language for validating that the implemented application complies with defined requirements. With these functions the application is evaluated to check that defined user requirements are implemented and that the application functions as specified.

### **2.1 Use of TURE in theory:**

When specifying the requirements and especially when developing the system requirement specification the user requirements are transformed to a test-script. The level of detail in the system requirement specification influences the development of the test-script. Sometimes it may be necessary to ask the users to articulate their requirements more precisely. If this is not possible at the moment, it may be necessary to wait with the details in the test-script until the development of the software specification.

When a prototype or a version of the computer system has been implemented, the Learn GUI objects function is used to generate a representation of the user interface of the computer system. If the system requirement specification was very detailed and all user interface elements were defined, the test-script created is sufficient for the validation to be carried out. If not, there is a need to develop the test-script further with the generated representation as a basis.

The test-script is then used to evaluate the computer system's compliance with user requirements. The possibility to do this evaluation on-line also makes it possible to continuously monitor the evaluation performed by TURE.

## 2.2 Practical use of TURE, a case study

To investigate the practical use of TURE we decided to set up a study comparing manual evaluation to evaluation with TURE. The study was designed to investigate the time needed for creating test-scripts/validation specifications and time needed for conducting the evaluation.

The study was accomplished by having different people conducting the manual evaluation (including creating the validation specification) and the evaluation with TURE (including creating the test-script). In this section we are focussing on the issues of creating test-script, evaluation of a computer system, and analysis of the result from the evaluation with TURE. The comparison between manual evaluation and evaluation with TURE is presented in the discussion and conclusions section.

### 2.2.1 Creating the test-script:

Simultaneous with the development of the system requirement specification for a computer system for writing, sending, receiving and administering different kinds of messages, we developed a test-script to be used when validating the computer system's compliance with user requirements. The requirement specification included user requirements in the form of detailed requirements on operations the user/operator should be able to do with the computer system. Examples of the requirements in the requirement specification are presented in Figure 2 below. (The requirements are translated to english).

- Selection of the command "List of adress" shall result in presentation of a dialog box "List of adress" in the working area.
- Selection of the command "Reset" shall result in presentation of a dialog box "Reset" in the working area. The button "All" shall be active.

- Selection of the command "Priority" shall result in presentation of a window "Priority" in the working area.
- Selection of the command "Erase" shall result in presentation of a dialog box "Erase" in the working area. The button "Yes" shall be active.

Figure 2: Examples on user requirements.

The requirements were then transformed to a test-script as described above. An example of the transformed requirements is depicted in Figure 3 below.

```

(1)      if (menu_select_item("Options; List of adress...")!=E_OK)
(2)          report_msg("2.4.7.6.4.4 There is no command for List of
adress");
(3)      else if(dialog_title(List of adress") !=E_OK)
(4)          report_msg("2.4.7.6.4.4 When selecting the command for
List of          adress no dialog box is presented")
(5)      else win_close("List of adress");

```

Figure 3: An example from the test-script.

Line 1: Tests if there is a command for "List of adress".

Line 2: Prints the defined text in the test report.

Line 3: Tests if there is a dialog box for "List of adress".

Line 4: Prints the defined text in the test report.

Line 5: Closes the window or dialog box.

## 2.2.2 Evaluation of the computer system:

After the implementation of the first version of the computer system the validation was carried out with TURE. A representation of the user

interface of the computer system was generated. The application was then evaluated according to the test-script and comments were generated.

The evaluation of the application's compliance with the defined user requirements resulted for instance in the comments presented in Figure 4 below.

<u>Requirement according to req. specification:</u>	<u>Evaluation carried out:</u>	<u>Comments:</u>
2.4.7.6.4.4	Evaluation of the presence and function of command "List of adress".	There is no command "List of adress."
2.4.7.6.4.5	Evaluation of the presence and function of command "Messages".	OK
2.4.7.6.4.8	Evaluation of the presence and function of command "Reset".	There is no command "Reset".
2.4.7.6.5.2	Evaluation of the presence and function of command "Priority".	There is no command "Priority".
2.4.7.6.5.6	Evaluation of the presence and function of command "Erase".	When selecting the Command "Erase" no dialog box is presented.

Figure 4: An excerpt from the evaluation report.

In addition to these requirements it was also a requirement that the user interface shall comply with the MS Windows styleguide. Evaluation of



this requirement was carried out with a tool called TUNE (Nordqvist, 1995).

### **2.2.3 Analysis of the evaluation result:**

When studying the result from the validation of the computer system it is possible to conclude the following. First, it was possible to identify a number of deviations from the requirements defined in the requirement specification by means of TURE. Second, some deviations identified by TURE were due to some defects in the test-script. Third, the traceability to the original requirements was simplified by the identification of the original requirements in the evaluation report. Fourth, the dialogue with the developers was facilitated by the possibility to replay the evaluation session.

When discussing the result from the evaluation with the developers, they agreed that they had sometimes deviated from the requirement specification. This discussion was to a great deal facilitated by the possibility to replay the evaluation and in the user interface point to discovered deviations and at the same time refer to the original requirement. Sometimes the developers pointed out that the result from the evaluation was erroneous. This was due to the fact that the developers had implemented some requirements with the help of other user interface elements. TURE was then not able to find the elements defined in the requirement specification, which resulted in TURE generating comments that the requirements related to these user interface elements were not fulfilled.

## **3. DISCUSSION AND CONCLUSIONS**

The discussion and conclusions are divided in following sections: creation of test-scripts, the evaluation of the application, comparison between automatic and manual evaluation and usability aspects.

### 3.1 Creating the test-scripts:

To create test-scripts for evaluation of an application's compliance with defined user requirements, the requirements have to be precise enough to implement in the C-like language supported by WinRunner. This means that considerable efforts have to be made to have the users define explicitly what they will be able to do with the computer system. A positive side-effect of this is that this procedure also ensures that one condition in IEEE P1233 (1993) is fulfilled, that all requirements on the system should be possible to validate.

The process of creating test-scripts is at the moment as time-consuming as creating the necessary validation specifications for manual validation, at least if the validation is performed only once. This depends mainly on the following:

- The requirements are often too generally formulated in the requirement specification. (This is of course a problem even in manual evaluation).
- Requirements without external behavior to the user can for the moment not be validated with TURE.
- Presently, every test-script is created from scratch.

If the requirements are too generally formulated, we must return to the users to have them elucidate what they want the application to do. Integration of the work of identifying and defining the user requirements on the application to be developed, with the work of creating test-scripts would probably facilitate the creation of test-scripts. This integration might also contribute to the solution of the problem that many large software development projects exceed their cost and time limits because of deficiencies concerning the development of the requirements on the computer system (see for example Lederer and Prasad, 1992) because of the necessity to be explicit in the requirements definition to be able to implement them in TURE.

Since TURE is based on the evaluation of compliance with user requirements through the user interface we have for the moment no possibility to validate requirements that relate to the application's inner functionality. This is a problem in many system development projects of today where requirement specifications are formulated in terms of technical functionality and not in terms of user requirements on the system (see for example Andriole, 1990, for a discussion of the importance of focusing on user requirements). This means that the functionality for the user risks to be hidden by technical requirements. If one instead chooses to let the user perspective dominate when identifying and defining requirements the creation of test-scripts would be much easier. The effect of this approach on the development of software requirement specifications is unknown. Possibly it could contribute to the solution of the problem that many large computer systems are so afflicted with serious deficiencies when delivered that they are not used (Ince, 1988). It has been shown that these deficiencies usually depend on problems in identifying and defining the requirements on the future computer system (Lederer and Prasad, 1992, Palmer, 1990).

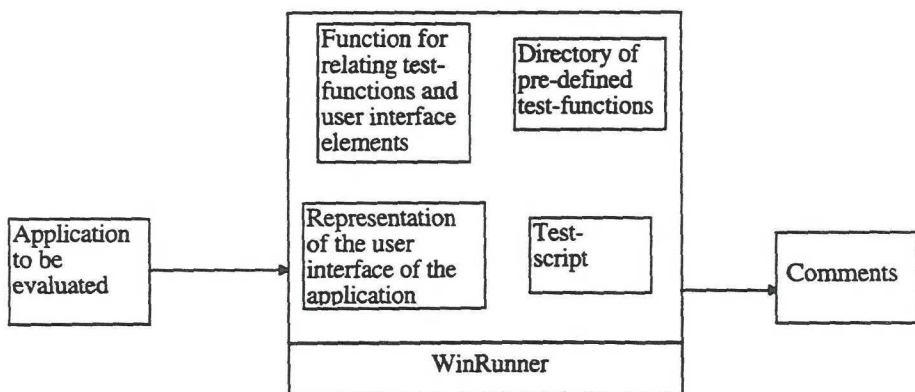


Figure 5: Possible future architecture of TURE

Since every test-script is created from scratch, it requires the same amount of effort for every application to be evaluated. A possible approach to this problem is to supplement TURE with a library of predefined test-functions for different types of user interface elements, and a function for relating applicable test-functions to the user interface elements in the representation of the application to be evaluated. An example of this further development of TURE is depicted in Figure 5.

### 3.2 Evaluation of an application

The evaluation of an application is rather straightforward since what is done is: 1) creation of a test-script according to the system requirement specification for the application to be evaluated, 2) generation of a representation of the user interface of the application, 3) evaluation that the application complies with defined user requirements, 4) analysis of evaluation results, 5) presentation of evaluation results for the developers.

However, some additional activities are necessary in relation to the evaluation of an application's compliance with user requirements using TURE.

First, a validation of the identified and defined requirements have to be conducted. Experiences from system development points to the importance of validating requirements to ensure that the requirements are the proper requirements, properly comprehended, properly formulated, consistent, complete and possible to test (possible to verify and validate, Boehm, 1984a). This is an extensive and difficult task that has to be done irrespectively if the subsequent validation is done manually or with the help of tools like TURE. In a way TURE could support this process since we have an opportunity continually to test the requirements as the application is implemented.



Second, to continuously check if the user interface elements in the application have got other names, or defined user interface elements have been changed so that for instance menus have been replaced with dialog boxes. As mentioned before, such changes mean that the user interface elements generated in the representation do not correspond to elements in the test-script.

Third, to ensure that changes of specific requirements, addition of new requirements, and omission of requirements are continuously taken care of and influence further development of the test-script.

### 3.3 Comparison between automatic and manual evaluation:

The results from the case study is illustrated in Figure 6.

	Manual	Automatic
Time elapsed for conducting the evaluation	8 h	1 h
Time elapsed for creating test-scripts/validation specifications	8 h	8 h

Figure 6: Comparison between automatic and automatic evaluation.

As the figure indicates, it takes equally long time to create a test-script and a validation specification. This depends mainly on the fact that for manual as well as automatic evaluation the test-script/validation specification is created from scratch for every application to be evaluated. With the further development of TURE illustrated in Figure 5, we estimate that reducing the time necessary for creating test-scripts with 50% is possible.



If we then consider manual and automatic evaluation with respect to the time elapsed to conduct the evaluation, it obviously takes much shorter time to conduct the automatic evaluation.

However, considering the magnitude of the time saved as possible to generalize to other applications is risky. Further use of TURE is necessary to create the necessary basis for any conclusion about the magnitude of the time saved when using TURE. Still it is possible to say that the time saved at least should be about 50% compared with manual evaluation. If we by this can unburden people from the task of manually evaluating applications and instead focusing on other usability aspects, the possibility to developing really usable computer systems in the future increases.

Another aspect worth pointing at when comparing manual and automatic evaluation is the possibility of considerable time and cost savings when evaluating a computer system's compliance with user requirements in iterative system development. Even if the time used initially when developing test-scripts or validation specifications is the same, our preliminary experience is that the cost in time and money for further development of the test-script is very small, as long as the application is not dramatically changed. For instance, in TURE replacing user interface elements or taking into consideration design changes where the user interface element's interrelations have been changed is very easy. It is likely that the time saved, indicated in Figure 6, for evaluation of the application will be the same for every iteration or version of the computer system also. According to my opinion, this means that when developing large systems where the number of iterations or versions could be more than 10, the time and cost savings are large.

### 3.4 Usability aspects

Even if TURE supports evaluation of an application's compliance with user requirements, TURE is of course not the answer to every question concerning development of usable computer systems. If we consider TURE from the perspective of Löwgrens (1993) definition of usability:

"Usability is a result of Relevance, Efficiency, Attitude and Learnability (REAL).

- The relevance of a system is how well it serves the users' needs.
- The efficiency states how efficiently the users can carry out their tasks using the system.
- Attitude is the users' subjective feelings towards the system.
- The learnability of a system is how easy it is to learn for initial use and how well the users remember the skills over time".

it is possible to say that TURE handles the relevance aspect of usability in the sense that it is possible to evaluate to what degree the application complies with the user requirements. This of course presupposes that specified requirements really represent the users requirements.

TURE does not handle the efficiency aspect of usability since there is no function in TURE to support the evaluation of how efficiently the users can carry out their tasks using the application. Supplementing TURE with a function for registering user interaction makes it possible also to handle the efficiency aspect since then it is possible to registrate the users' interaction with the computer system when carrying out relevant working tasks.

Nor does TURE adress the attitude aspect of usability since this is mainly a question of what the user feels about the computer system. A more

thorough treatment of usability requires that TURE is supplemented with different kinds of investigation methods to study this aspect.

With respect to the issue of how easy the computer system is to learn and how well the users remember the acquired skills, this is not handled by TURE which focuses on whether the user requirements are implemented in the computer system. But it is possible to imagine the use of a further developed TURE in registering and analyzing the use of the computer system when it is delivered to the users.

Finally, I would like to return to the relevance aspect and state that TURE's evaluation of the relevance aspect also indirectly affects the other aspects of usability. If the user functionality defined in the requirement specification is not implemented in the computer system this is going to influence the other usability aspects. For instance, if one studies the learnability aspect it is possible to come to the conclusion that the computer system is easy to learn, but this could still imply that the computer system is missing task relevance. My personal experience is also that it is possible to have a high value with respect to the user attitude of the computer system but this does not guarantee that the computer system is relevant for the task to be performed.

#### 4. FUTURE WORK

There is a need to further develop TURE in many ways. First, TURE needs to be extended according to Figure 5. The focus is then on developing predefined test-functions to handle the different kinds of user interface elements in a user interface. Developing a function for integrating these test-functions and the user interface elements identified with the Learn GUI objects function is also necessary.

Second, we see it as necessary to further develop TURE to deal with the user interface elements utilized in geographical applications (GIS). At the

moment TURE could not handle symbols presented in a computerized map display.

Third, in our opinion it is necessary to supplement TURE with a function for registering user interaction, specially if we aim at addressing the efficiency and learnability aspects of usability.

**Fourth, developing TURE further is necessary so that we can improve our dealing with situations where minor changes in the user interface design have lead to replacement of user interface elements when implementing the computer system. For the moment we have to manually check that the representation generated with TURE really contains the elements specified in the requirement specification. A possible solution to this problem is to develop a function to automatically check the correspondence between the user interface elements in the test-script and the user interface elements in the generated representation.**



## REFERENCES

- Andriole, S. J., (1990). **Information System Design Principles for the 90s, Getting IT Right**. AFCEA International Press, Fairfax, Virginia.
- Boehm, B. W., (1984a). Verifying and Validating Software Requirements and Design Specifications. **IEEE Software**. Vol. 1, Number 1, January 1984, pp. 75-88.
- Dix, A., Finlay, J., Abowd, G. and Beale. R. (1993). **Human-Computer Interaction**. Prentice Hall International Press (UK) Limited.
- IEEE Std. 830-1993, (1994). **IEEE Recommended Practice for Software Requirements Specifications**. Institute of Electrical and Electronics Engineers, Inc., New York.
- IEEE P1233, (1994). **Guide for developing System Requirements Specifications**. Institute of Electrical and Electronics Engineers, Inc., New York.
- ISO 9000:1991, (1991). **Quality Management and Quality Assurance Standard - Part 3, Guidelines for Application of ISO 9001 to the Development, Supply and Maintenance of Software**.
- Ince, D. (1988). **Software Development: Fashioning the Baroque**. Oxford University Press.
- Lederer, A. L. and Prasad, J. (1992). Nine Management Guidelines for Better Cost Estimating. **Communications of the ACM**, 35, 2 (February), pp. 51-59.
- Löwgren, J. (1993). **Human-Computer Interaction, what every system developer should know**. Studentlitteratur, Lund, Sweden.



Mercury Interactive Corporation, (1993a). **XRunner/WinRunner Technical Overview**. ver 1.0. California: Author.

Mercury Interactive Corporation, (1993b). **WinRunner User's Guide**, California: Author.

Mercury Interactive Corporation, (1993c). **Context Sensitive Testing, User's Guide**, California: Author.

MIL-STD-498, (1994), Military Standard for Software Development and Documentation. **AMSC No. N7069**.

Nordqvist, T. (1995). **Computer-Supported User Interface Evaluation**. Paper submitted for publication.

Palmer, J. D., (1990), Software Systems Requirements Engineering for Command and Control, in Andriole S.J. (ed.) **Advanced Technology for Command and Control Systems Engineering**. AFCEA International Press, Fairfax, Virginia, USA.

Rombach, H. D. (1990). **Software Specifications: A Framework**. Curriculum Module SEI-CM-11-2.1, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., January.

## ACKNOWLEDGMENTS

The author wants to thank Per Asplund at Telub AB, for his work at implementing the tool and the discussions concerning evaluation of user interfaces. Jonas Löwgren, Kjell Ohlsson, Kristian Sandahl and Joachim Karlsson read an earlier version of this paper and had many creative ideas for which I am very grateful.

This study was funded by the Swedish National Board for Industrial and Technical Development, Telub AB and the Defence Material Administration.

Winrunner is a trademark of Mercury Interactive Corporation.

OSF/Motif is a trademark of the Open Systems Foundation.

Windows is a trademark of Microsoft Corporation.

Telub AB is a company focused on consultancy services within information technology and telecommunications.

Institution/Department

**Arbetsvetenskap**

Upplaga/Number of Copies

**100 ex**

Avdelning/Division

Datum/Date

**12 februari -96**

Titel/Title

**Computer Support for User Requirement Evaluation in System Development**

Författare/Author(s)

**Tommy Nordqvist**

Uppdragsgivare/Commissioned by

Typ/Type

**Telub AB and Luleå University**

NUTEK

- ☐ Doktorsavhandling/Doctoral Thesis
- ☐ Licentiatuppsats/Licentiate Thesis
- ☒ Forskningsrapport/Research Report
- ☐ Teknisk Rapport/Technical Report
- ☐ Examensarbete/Master Thesis
- ☐ Övrig rapport/Other report

Språk/Language

☐ Svenska/Swedish


☒ Engelska/English

☐ .....

Sammanfattning, högst 150 ord / Abstract, max 150 words

Nyckelord, högst 8 / Keywords, max 8

Underskrift av granskare/handledare / Signature of examiner/supervisor

 Kjell Ohlsson  
Namnförtydligande:

