# User Interface Design

## A Software Engineering Perspective

Soren Lauesen

ADDISON
WESLEY

# User Interface Design

A Software Engineering Perspective


We work with leading authors to develop the strongest educational materials in computing, bringing cutting-edge thinking and best learning practice to a global market.

Under a range of well-known imprints, including Addison Wesley, we craft high-quality print and electronic publications that help readers to understand and apply their content, whether studying or at work.

To find out more about the complete range of our publishing, please visit us on the World Wide Web at: www.pearsoned.co.uk

# User Interface Design

A Software Engineering Perspective

Soren Lauesen


Harlow, England • London • New York • Boston • San Francisco • Toronto
Sydney • Tokyo • Singapore • Hong Kong • Seoul • Taipei • New Delhi
Cape Town • Madrid • Mexico City • Amsterdam • Munich • Paris • Milan

PEARSON EDUCATION LIMITED

Edinburgh Gate
Harlow
Essex CM20 2JE
England

*and Associated Companies throughout the world*

*Visit us on the World Wide Web at:*
www.pearsoned.co.uk

---

First published 2005



Cover Picture by Otto Frello (1995): The Road



ISBN 0 321 18143 3

**British Library Cataloguing-in-Publication Data**

A catalogue record for this book is available from the British Library

**Library of Congress Cataloging-in-Publication Data**

A catalog record for this book is available from the Library of Congress

10 9 8 7 6 5 4 3 2 1
10 09 08 07 06 05

Typeset in Palatino 9.5/12 and Helvetica by 59

Printed and bound in Great Britain by Biddles Ltd., King's Lynn

*The publisher's policy is to use paper manufactured from sustainable forests.*

# Contents

# Preface

When you design the user interface to a computer system, you decide which screens the system will show, what exactly will be in each screen and how it will look. You also decide what the user can click on and what happens when he does so, plus all the other details of the user interface. It is the designer's responsibility that the system has adequate *usability* – it can do what is needed and is easy to use. It is the programmer's responsibility that the computer actually behaves as the designer prescribed.

If you are involved in designing user interfaces, this book is for you. You don't just learn about design and why it is important, you actually learn how to do it on a real-life scale.

Designing the user interface is only a small part of developing a computer system. Analysing and specifying the requirements to the system, programming the software and testing and installing the system usually require much more effort, and we will barely touch on it in this book.

## Making the user interface

How do you design the user interface to a computer system? Ask a programmer and he may say:

> *The user interface? Oh, it is so boring. We add it when the important parts of the program have been made.*

Ask a specialist in human–computer interaction (HCI) and he might say:

> *The user interface? Oh, you have to study the users and their tasks. To do this you must know a lot about psychology, ergonomics and sociology. Designing it? Well, you have to come up with a prototype of the user interface and review it with the users. Programming? Oh, that is what the programmers do when the user interface has been designed.*

Is there a communication gap here? Yes, for sure. The truth is that it is easy to make a user interface – exactly as the programmer says – but it is hard to make a *good* user interface – and that is what the HCI specialist tries to do.

There are thousands of books on programming. Common to all those I have seen is that the user interface is rather unimportant – it is just a matter of input to and output from the program. Programming *is* very difficult and there is nothing wrong with dedicating thousands of books to it. The only problem is that the programmers

don't learn to design the user interface, although typically more than half of a program deals with the user interface.

There are a few dozen books on HCI. They tell a lot about human psychology, how to study users and their tasks, how to test a prototype or a finished system for usability and many other good things. Amazingly, they say very little about the actual design of real-life user interfaces: how many screens are needed, what should they contain, how should we present data, how can we make sure that our first prototype is close to a good solution. To be fair, I have seen a few books that explain about design: Redmond-Pyle and Moore (1995), and to some extent Cox and Walker (1993) and Constantine and Lockwood (1999). However, even here most of the screen design pops out of the air.

Many programmers have looked into the books about HCI. They read and read, but find little they can use right away. And they don't get an answer on how to design a user interface. The books somehow assume that it is a trial and error process, but programmers don't trust this. They have learnt that trial and error doesn't work for programming. Why should it work for user interface design?

## Bridging the two worlds

This book tries to bridge the gap. A crucial part of the book is how to design the user screens in a systematic way so that they are easy to understand and support the user efficiently. We call this approach the *virtual window method*. The virtual windows are an early graphical realization of the data presentation. Task analysis comes before the virtual windows and dialogue design after.

I have developed the virtual window method over many years with colleagues and students, and it has become a routine way to develop a user interface. We don't claim that the result is free of usability problems right from the beginning, just as a good program isn't free of bugs right from the beginning. But the large-scale structure of the interface becomes right and the problems can be corrected much better than trial and error.

The cover picture illustrates this. It was painted by the Danish artist Otto Frello in 1995. Imagine that it is the road to good user interfaces. It is a firm road, but it is not straight. However, there is support at every corner (notice the lady who appears repeatedly along the road).

In order to design in a systematic way, we have to draw on the programmer's world as well as the HCI specialist's world, but we pick only what it necessary and adapt it for our specific purpose. Here are some of the major things we draw on:

- How to measure usability
- Usability testing and heuristic evaluation
- Different kinds of prototypes

- Data presentation
- Psychological laws for how we perceive screens
- Task analysis and use cases
- Data modelling (programmer's world)
- State diagrams (programmer's world)
- Checking the design and keeping track of the problems (programmer's world)

To get from these classical issues to the systematic design, we add some novelties:

- Mental models for how users understand what they don't see
- Virtual windows: Presenting data in few screens that cover many tasks efficiently
- Function design of the interface: Adding buttons, menus, etc., to the screens in a structured and consistent fashion.

## Programming the user interface

The book is about design of user interfaces down to the detail. Programming the user interface is closely related, so we need teaching material on how to do this too. Here is a pedagogical and even a political problem: which platform to use for the programming. There are many choices, Microsoft Windows, UNIX, HTML, Swing, etc. Ideally, we should have teaching material for each of these.

I have chosen to release a free companion to this book: a booklet on user interface programming with Microsoft Access. The main reason for this choice is that Microsoft Access is readily available to most students as part of Microsoft Office; the connection to a real database is easy; you can quite quickly systems that look real, and you can gradually add the full functionality. Access also exhibits the techniques found on other platforms, such as GUI objects, embedded SQL and event handling. My dream is to have additional booklets that cover other platforms.

The Access booklet is free for download from www.booksites.net/lauesen. It uses examples from this book and shows how to turn them into a functional system.

## Design cases covered

The book illustrates most of the design techniques with a system for supporting a hotel reception. This is sufficiently complex to illustrate what happens in larger projects. However, it makes some people believe that this is the only kind of system you can design with the virtual window method. Fortunately, this is not true.

The book shows how to design different kinds of systems too with the same method. Chapter 10 shows how to design a Web system for course evaluation and

management. Chapter 11 shows how to design an advanced e-mail system. In section 3.7 we show part of a complex planning system designed with the method. The design exercises in Chapter 17 deal with other kinds of systems, for instance a Web-based knowledge management system, a photocopier, a system for supporting small building contractors and an IT project management system.

## Uncovered issues

In the book we pay little attention to aesthetics – how to make the screens look pretty and attractive – or to the entertainment value of the system. These issues are very important in some systems, but in this book we mainly look at systems that people use to carry out some tasks, and here aesthetics is not quite as important. And to be honest, I have not yet figured out how to deal with a esthetics in a systematic way.

The book focuses on systems with a visual interface where a lot of data has to be shown in some way. As a result, we don't discuss verbal interfaces (e.g. phone response systems) or user interfaces for the blind.

We also pay little attention to low-level aspects of the user interface, for instance how to display a button so that it appears in or out of the surface, how a scroll bar works, whether to use a mouse or a touch screen. We take these things for granted and determined by the platform our system has to run on. These issues are important in some projects, and they are excellently covered by, for instance, Preece *et al.* (1994, 2002), Cooper (1995) and Dix *et al.* (1998).

Why doesn't the author mention object orientation? Doesn't he know about it? Yes, he does, very well indeed, but the traditional object-oriented approaches don't really help us designing a user interface in a systematic way. However, we can describe the design results as objects, and this gives an interesting perspective on what the design produces and how it relates to traditional object-oriented approaches (see section 9.2.3).

## How the book is organized

The book is organized as three parts that may be combined in various ways for different kinds of courses.

**Part A: Best of the classics (Chapters 1–4)**

Some classical usability topics: defining and measuring usability; using prototypes, usability tests and iterative design; data presentation and how users perceive what they see on the screen.

**Part B: Systematic interface design (Chapters 5–9)**

How to design the prototype in a systematic way so that it is close to being right early on. This includes task analysis, designing the virtual windows, defining the system functions and their graphical representation. Chapter 9

reflects on the virtual window approach and compares it with other design approaches.

**Part C: Supplementary design issues (Chapters 10–16)**
Optional topics that may be included depending on the audience. Examples are systems development in general, data modelling, user documentation and support, design cases for different kinds of systems.

Throughout parts A and B we present the topics in a learn-and-apply sequence: After each chapter you can do something that would be useful in a real development project. You then try it out in a design exercise. Here is a summary:

**Chapter 1: Usability.** Here you learn to identify the usability problems of an existing system through a usability test. You also learn how to specify and measure usability. In an exercise you specify and measure usability for an existing system, for instance an existing Web site.

**Chapter 2: Prototyping and iterative design.** You learn the classical design approach for usability: Make a prototype, test it for usability, revise the design, test again and so on. In an exercise you design a prototype of a small system with a few screens, and test it for usability.

**Chapter 3: Data presentation.** You learn about the many ways to present data on the screen, and the psychology behind them. You apply the rules on existing designs, and you try to design non-trivial data presentations.

**Chapter 4: Mental models and interface design.** You learn to use the psychological law of object permanence to explain why users often misunderstand what the system does, and how you can avoid it with a systematic way of designing the user interface. You also learn about the two extreme interface architectures: the database oriented and the step-by-step oriented.

**Chapter 5: Analysis, visions and domain description.** You learn how to model user tasks and data in a way suitable for user interface design. You apply it in a larger design project that continues over the next chapters. (Parts of task analysis and data modelling are explained in the supplementary part, since they are well-known topics needed only for some audiences.)

**Chapter 6: Virtual windows design.** You learn how to design *virtual windows*: the large-scale structure of the user interface – how many screens, which data they show, how they show it, how you make sure that they support user tasks efficiently and how you check the results. You try all of this in a larger design project.

**Chapter 7: Function design.** You learn how to convert the virtual windows into computer screens, and how to add system functions to the screens based on the task descriptions and state diagrams. You also learn how to decide

whether a function is to be shown as a button, a menu, drag-and-drop, etc. Again you apply it in the large design project.

**Chapter 8: Prototypes and defect correction.** Based on the previous work products, it is rather easy to make a prototype – ready for usability testing. This chapter has a case study of how this worked in a real project, which problems were found, how they were removed and what happened when the final system was tested for usability. Your design exercise is to make a prototype for your design project, test it for usability and suggest ways to correct the problems.

**Chapter 9: Reflections on user interface design.** This chapter is theoretical. We review the virtual window method and compare it with other systematic design methods, for instance object-oriented design.

## Courses

The book is based on material I have developed over many years and used at courses for many kinds of audiences.

**IT beginner's course.** The first semester for computer science and software engineering students is hard to fill in. Programming is mandatory in the first semester, and a prerequisite for most other IT courses. So what else can you teach? An introductory course in general computer technology and a database course are often the attempt. We have good experience with replacing either of these with user interface design based on this book. The course has a strong industrial flavour and students feel they learn to design real systems. And in fact, the course is readily useful. It is also a bit difficult, but in another way than programming.

Parts A and B of the book are aimed at this audience. Depending on what else they learn at the same time, we include data modelling from part C and constructing the user interface (the Access booklet). For some audiences, the full course is more suited for two semesters, and will then be an introductory systems development course at the same time.

When we include data modelling, the course replaces a large part of the traditional database course, and prepares students for an advanced database course. On our courses we often have students who have already followed a traditional database course, and to our surprise they follow our data model part enthusiastically. *Now we learn how to use it in practice*, they say.

We take the data model part concurrently with the soft parts about data presentation and mental models (part A). In this way we combine hard and soft topics and keep everyone on board.

We usually finish the course with a 4-hour written exam. This is not a multiple-choice or repeat-the-book exam. The students are, for instance, asked to specify

usability requirements, make a data model and design screens for a real-life case. They may bring whatever books they like to the exam. Chapter 17 contains sample exam questions.

**IT-convert course.** In recent years we have had many students that come from humanities or social sciences and want to become IT professionals. This book has worked amazingly well with this audience. The students are mature and learn fast. They appreciate very much the real-life attitude of the whole thing, and they feel motivated to enter also the hard technical part of the course (the Access booklet).

**Courses for mature IT-students.** The book makes excellent courses for software engineering or information systems students who know about programming and the technical aspects of design. Parts A and B of the book are suited for such courses. If the students have followed a traditional HCI course already, we treat part A lightly.

In these courses the data model part need not be included, and the Access booklet might be replaced by programming the user interface for any other platform the students might know.

**Professional courses.** We have used parts A and B of the book for professional courses taking 2–3 days. The participants are systems developers and expert users participating in development. Data modelling is not essential here; the professional developers usually know it already, and the expert users don't have to learn it when they cooperate with developers. Actually, we invented the novel parts of the book through such courses. We observed that some professional design teams produced excellent user interfaces, while others ended up with a messy interface that users found hard to use and understand. By comparing their work, we learned the cause of these differences and used it to improve the systematic design approach.

## Course material

Overheads corresponding to all the figures of the book, and solutions to most of the exercises, are available for teachers. See www.booksites.net/lauesen or e-mail the author at slauesen@itu.dk.

The Access booklet is free for download from www.booksites.net/lauesen. The package includes a handy reference card for Access and Visual Basic, plus the hotel system application in various stages of completeness corresponding to the exercises in the Access booklet.

## The author's background

At the age of 19, I started to work in the software industry for the Danish computer manufacturer *Regnecentralen*. At that time user interfaces were extremely primitive. We fed data into the system through punched cards or paper tape, and output was printed reports. When we later began to talk about on-line computing, input/output

was through typewriter-like equipment. Renting a computer for an hour cost the same as paying someone to work for 30 hours; and computers were 5000 times slower than they are today.

Nobody thought of usability. The customer paid until he had a program that printed results he could use with some effort. Everything to do with computers was a specialist's job.

My first project in 1962 was to develop a program that made the computer play a game invented by the Danish poet and designer Piet Hein. Neither he nor anybody else knew how to win the game. I had never programmed before, so I had to learn that too. And of course I didn't know that it was difficult to make such a program, so it took me a couple of weeks to solve the problem. Piet Hein wanted to have people play the game against the computer at exhibitions, but how could the user interact with the computer? Our hardware developers found a way to connect a bunch of buttons and lamps directly to the multiplier register of the CPU, and I made the program without multiplying anything. We then designed a nice user interface that people could use immediately. I didn't realize that this was the only good user interface I would make until 1973.

In the period until 1973, I developed many kinds of systems, for instance computation of molecule shapes based on X-ray diffraction, administration of pension contributions, optimization of tram schedules and rosters. Later I moved to another department in the company, where we developed compilers and operating systems. These systems became extremely fast, compact and reliable technical wonders. It took me many years to realize that we had made these miracles without understanding that the customers had other interests than speed and reliability. They also wanted usability – efficient task support – which we didn't provide and didn't understand.

In 1973, I moved to Brown Boveri, now part of ABB (Asea Brown Boveri). We were a new department and our first project was to develop a new line of process control systems with colour screens and special-purpose keyboards that could be operated by users with big, insulating gloves. Our first delivery was power distribution control for a part of Denmark. We knew how to make reliable, fast and compact code, but this was the first time I realized that ease of use was also important. If our system failed technically, 300,000 people wouldn't have power. But if the user couldn't figure out how to operate the system, the consequences might be the same. I had never been involved in anything as serious before. We made it, and usability became very good because we were inspired by the way users had controlled the power before.

In 1974–1975, I took temporary leave from Brown Boveri and worked for ILO in Ghana, helping with management consultancy in IT issues. This was the most fascinating year of my life. I learned how different other cultures and value systems could be, and that our own society had gained much in economic welfare and security, but lost a lot in other aspects of life quality. I also learned that I didn't know

anything about management and that such knowledge was important. Returning home, I took a business diploma as fast as possible.

In 1979–1984, I moved to a new software division established by NCR in Copenhagen. For some reason I became a manager, but kept being involved in software development – even on the programming level. I soon got two management responsibilities: (1) developing experimental technology for the next generation of products, and (2) assuring that the rest of the division delivered adequate quality in the present products. My main lesson in these years was that there was a very, very long way from proving a new technology to having a multinational company accept it for new products.

Now, where did I learn about usability? Not in industry! In 1984, I became a full professor at the Copenhagen Business School. They had decided to establish a new degree program – Business and Software Engineering. I found it an excellent idea since I had seen so many bright IT people who didn't understand business; and I had seen so many excellent managers who didn't understand technology. There was a communication gap, and educating young people in both areas at the same time seemed a great idea. I was willing to take responsibility for the IT side of the new education.

Working here without the daily pressure of finishing new products gave me time to wonder why the users and customers of all the technical wonders we had made over the years didn't like our systems as much as we did ourselves. I started to study usability and wondered whether users tried to understand what the computer did, or whether they just thought of carrying out their tasks, as most researchers assumed. I ran several experiments and concluded that if the system has a certain complexity, users form a mental model of how it works. They do so unconsciously, and often their model doesn't match with what the system actually does. This is the source of many usability problems. (Norman, 1988, found similar results in the same period.)

At that time, user manuals was a big issue, and I used the mental-model insights to develop ways to write manuals that combined two things at the same time: learning how to carry out the tasks, and understanding – at a non-technical level – what happens in the system. The approach was quite successful, and I served as a consultant for many technical writers and gave courses on writing user documentation.

Later I became more interested in designing good user interfaces – where manuals weren't necessary. During a long cooperation with Morten Borup Harning and varying master's students, we developed the approach that is central to this book.

I had expected that I would return to industry after about five years, but found that the combination of IT and business was fascinating, and that working at a business school helped me open the doors to a wide range of companies. Gradually I became a researcher who worked closely with industry.

In 1999, I moved to the new IT University established in Copenhagen. Now my role seemed to be reversed. At the business school, I had been regarded as the technical guy who didn't quite understand that business was the most important thing, while technology had minor importance. Now my computer science colleagues regarded me as the business guy who thought that business and usability were more important than technology.

This taught me one more thing: balancing between the extremes is very hard but also very important. I have tried to strike such a balance in this book.

## Acknowledgements

The ideas behind virtual windows (Chapter 6) were developed by Morten Borup Harning and I, with some input from Carsten Grønning.

I have also learned a lot from cooperation with many of my students, in particular Susanne Salbo and Ann Thomsen who planned and carried out a hit-rate comparison of mock-ups against the real system. I want to thank William Wong for reviewing part of the book and encouraging me to publish it, Jean-Guy Schneider and Lorraine Johnston for helping me develop terminology in some of the areas, Klaus Jul Jeppesen for many insights in the user support area, and Flemming Hedegaard and Jacob Winther Jespersen for trying the entire course material in their own courses and giving me much valuable feedback.

Finally, my colleagues Peter Carstensen, Jan C. Clausen, Anker Helms Jørgensen and Rolf Molich have for many years been my excellent sparring partners in the HCI area, and I am most grateful for the insights I have got from them in many matters.

# Part A

## Best of the classics

In part A of the book we look at some classical techniques in user interface design:

- How to detect what users find difficult or inconvenient when they use a system, and how to measure the goodness of the system (Chapter 1).
- How to use prototypes and iterative design to come up with a good user interface (Chapter 2).
- How users perceive the contents of a screen and how we can present data to them (Chapter 3).
- What users believe goes on in the system and how it relates to various kinds of user interfaces (Chapter 4).

The classical approaches to user interface design are to a large extent based on trial and error. The designers compose the user interface with their intuition and general understanding of users, and then measure how good the user interface is. If it isn't satisfactory – and they have time to improve it – they revise their design and measure once again how good it is. They will usually have to revise the design many times to get an adequate user interface.

There is one more important classical technique: How to study and describe the users, their tasks and the data they use. In part A of the book, we assume that this is part of an analysis and requirements phase that precedes the user interface design work. In part B of the book, we will expand our perspective and treat parts of the analysis phase together with a more systematic design approach.