Position paper for Interact'97, Workshop W3

12-01-2004 10:22
SE&HCI.doc

Adding Usability to Software Engineering

Søren Lauesen
Copenhagen Business School, e-mail: slauesen@cbs.dk

Why is it apparently difficult to integrate HCI approaches with typical software development? The author's position is that it is not really difficult. The problem is that the two communities don't understand each other well, they have different priorities, and often they use different terms for essentially the same things.

Below I will show how some HCI approaches can be added to software development for improved usability. The discipline of Software Engineering offers several methods for the entire development of a software product. In contrast, HCI offers only some techniques, but no coherent development method, and hardly even a method for design of a user interface. For this reason I start with Software Engineering, adding techniques to it, rather than with HCI.

System development

Software Engineering deals with many issues during system development, for instance functionality of the product, correctness of it, security, cost, and deadlines. Each issue requires special techniques for dealing with it. Usability is just one more issue that has to be dealt with. However, usability has a growing importance and is handled badly in current system development. At times I wonder why some HCI researchers are studying system designers. Present designers don't seem to have the key to better usability.

Usability as "just another issue" explains the difference in priorities between HCI people and software developers. HCI people are concerned about the lack of attention to their issues and want a greater say, although they don't quite understand the development process. Developers have many other things to do, don't quite understand what HCI people are doing, and have enough problems controlling the process already.

Most Software Engineering methods are variations of a phased approach - the waterfall model. For simplicity, I will use the following model:

> **Phases or major activities:**
> Analysis (define requirements)
> Design (outline the solution)
> Programming (implement the solution)
> Testing (find and correct defects in the solution)
> Maintenance (repair defects found after deployment, and prepare next version)

The methods vary as to the amount of concurrency allowed between the phases, with the traditional waterfall model assuming a strictly sequential process, and a risk-driven model allowing more overlap of phases. The methods also vary as to the amount of iteration allowed. Again, the traditional waterfall model assumes no iteration, prototype-based methods use heavy iteration, particularly during design, while Rapid Application Development uses many complete iterations of the entire process.

Most real-life projects don't follow any strict method, although managers usually believe they do. However they always contain some elements of analysis, some of design, some of programming, etc.

Below I will discuss how various HCI techniques fit into the phase model and how they can complement the traditional system development. The table gives an overview of the suggestions. It shows the techniques (activities) that could be used in each phase, the results (deliverables) coming out, and the various user roles in a user-centered development. Some of the suggestions are not really HCI techniques, but techniques from other disciplines known to improve usability.

I have not seen all the suggestions used in any single project, but I have observed each suggestion being used successfully in at least one project.

Analysis

The main usability problem during analysis is to define what the new system should do as seen from the user's perspective.

Task analysis provides a thorough understanding of the current tasks to be supported. Task analysis uses many techniques such as observations of users, interviews, questionnaires, protocol analysis. In practice it is not essential to describe tasks in the detail traditionally used in HCI, since the tasks will be carried out differently with the new system. What is essential is to make a list of important tasks, identify the information needed by the user to carry out each task, and also describe the scenario for the task, e.g. the work context, the physical conditions, and the user profiles [Carlshamre & Karlsson, 1996]. Task analysis for real-life projects is still much of an art, but good task lists and scenario descriptions are very useful for checking the rest of the system development, e.g. whether the interface design supports all the tasks, whether the users can find out how to perform each task, whether all tasks are supported by manuals or help, whether the right test subjects are selected, etc.

The terminology used in task analysis and in object-oriented analysis can be very confusing, leading to misunderstandings between developers and HCI specialists. The term "task" can be roughly equivalent to a "use case", but as I used it above, it is more equivalent to "essential use cases". The term "scenario" is used with so many different meanings that it is confusing too.

Making **Rich Pictures** can give an understanding of the problems to be eliminated in the new system. This helps to define the system goals and the usability goals. Questionnaires and interviews can be used too for identifying current problems.

Focus Groups and **Future Workshops** also give an understanding of the problems to be eliminated, but they are particularly good to create ideas, visions, and goals for the new system, resolve conflicts between different user groups, and prioritize requirements. Harwood [1997] suggests that functional goals can be defined as high-level use cases; but in general it is still an art to define tangible system goals and usability goals in real-life projects [Cooper, 1996; Gould & al., 1991].

User roles: Some users will have a rather passive role in this phase, being observed during task analysis, interviewed, or asked to answer questionnaires. Other users will have a more active, participating role, for instance as members of focus groups or of groups making rich pictures.

Design

HCI techniques in the phase model

Phase:	HCI activities:	HCI deliverables:	User roles:
Analysis	Task analysis Making rich pictures Focus groups	System goals Usability goals Scenarios Task lists	Interviewed Observed Participating
Design	Visualizing data Interface design Usability test Iterative design Style guide	User data model Prototype	Evaluators Test subjects Co-designers
Programming		User manuals Help system Courses	Evaluators Test subjects Writers
Testing	Usability test		Test subjects Evaluators
Maintenance	Questionnaires Logs Focus groups	More releases with improved usability (RAD)	Interviewee Observed Participating

The main usability problem during design is to define the user interface of the new system in such a way that the necessary functionality is provided, the system is easy to learn, efficient for both daily and casual users, and attractive to users. The balance between these qualities depends of course on the application in question. For some systems, a long learning time is acceptable, for others easy learning and attractiveness are very important (e.g. most multimedia applications).

Visualizing data is a technique that can be used before the functionality of the interface is considered [Lauesen & al., 1994]. The technique produces a **user data model** consisting of slightly idealized screen pictures (*logical windows*) that also cover all data in the database. The logical windows are consciously designed to strike a balance between efficient task support and easy learning. It is essential that the logical windows are tried out with various tasks and filled with both typical and unusual data. This tends to reveal many problems in the logical windows, as well as in the technical database and the task list.

Most of the visual and graphical design of screens can be made as part of this technique. For traditional administrative applications, screens tend to have standard looks, but Nygren et. al. [1992] show how graphical design can improve task performance. For applications like process control or multimedia the graphical design is an art.

Interface design covers many techniques more or less well-developed, and more or less formal. The result of the interface design must include a prototype of the interface and usually also notes or specifications of how the various functions work. Prototypes can have many forms ranging from paper mockups to functional prototypes. Each form has its advantages and disadvantages.

If data has been visualized with a user data model, much of the interface design consists of adding functions to the logical windows, and modifying or splitting the windows to satisfy restrictions on screen size [Lauesen & Harning, 1993].

Usability testing is very important during design. Experience shows that inspection of the interface (*heuristic evaluation* in HCI terms), style guides, user evaluations, etc. are insufficient to prevent the real problems in the interface [Cuomo & Bowen, 1994; Desurvire & al., 1992]. Only a usability test with real users can find most of the problems. This is in contrast to the technical parts of design, where most problems can be found with inspections, etc. If usability tests are not made in this phase, the problems will be found so late that it is too expensive to repair many of them.

Scenario descriptions and a good task list can provide the necessary tasks for users to try out during the usability test. If a user data model has been made, a simplified form of usability testing of it can reveal many problems even before any functionality has been designed into the user interface.

Iterative design is necessary to repair the problems found during usability testing. In principle the designers should allow themselves to discard the entire design and start all over. In practice this is almost never done, so the quality of the first prototype becomes a limiting factor.

Style guides can help during design. They prevent many useless discussions on detailed design issues and help users move from one application to another. However, they cannot guarantee usability of an application and can prevent only a minor part of real-life usability problems.

User roles: Some users will have a rather passive role as test subjects in this phase. Others are more active as evaluators of designs, and some may even work as designers in a design team. It is important that test subjects are not also designers, because then they know too much about the system to be representative users. Participatory design also has its risks: Users easily come up with designs that are insufficient, inconsistent, or impossible to implement. Users - like many developers - easily become very enthusiastic about their designs, forgetting whether it fulfills the real requirements.

Programming

In principle there should not be usability problems during programming. The activity is just to implement what has been designed and usability tested already. Problems may occur, of course, for instance because a design cannot be implemented - or only with unreasonable costs. Probably the worst danger during programming is programmers who don't respect the design, believing that they can make

it much better. Programmers are even less likely than designers to foresee the problems real users will have with the interface, so deviations from a well tested design are dangerous.

However, the programming phase is potentially the time to develop the user documentation in the form of **manuals**, **on-line help**, **courses**, etc. This is only possible if the design is reasonably complete. A prototype, as described under design, is sufficiently finished for writers to work concurrently with programmers.

Traditionally, much of the user interface design took place during programming, making the documentation job very difficult, if not impossible. The result was that user documentation was always late.

Techniques for writing documentation don't fit into the table, but HCI has several things to offer here, like the Minimal Manual or user's conceptual models. Experience has shown that a user data model is a good basis for the conceptual model, and the task list is a good basis for the Minimal Manual.

User roles: Users are not involved in the programming itself, but they should be involved in evaluating and testing the manuals, etc. Often users can be of great help as writers, since they can write in the user's language. However, few users have the necessary skills to write and to understand the computer side of the system. Both are needed for good documentation.

Testing

The main usability issue during testing is to verify the usability requirements in the final system, e.g. that the necessary task support is present, that the system is easy to learn and efficient to use, etc. If design has been done carefully and tested well, few problems should remain. There are, however, problems that are difficult to find with a prototype, so usability testing and problem correction are important also in the test phase. Correcting a problem in the test phase is more expensive than in the design phase and sometimes unrealistic to do.

Usability testing is the main technique in this phase. **User roles** are test subjects and evaluators of whether the system goals and usability goals are fulfilled.

Maintenance

The main usability issue during maintenance is to find out how well the system works with the users in their normal environment, and gather ideas for improvement.

The usability techniques available are quite similar to the analysis phase. One additional technique is available, however: logs of what the users are actually doing. Many things can be logged such as windows and functions frequently used (giving also information about those rarely used), mistakes, error messages, keywords tried by users when searching for help. These data give clues for improvement, and can even be used for on-line, unsolicited advice to the user, as demonstrated by several newer Microsoft products.

The main result of HCI efforts in this phase is to release versions with improved usability. The idea of Rapid Application Development (RAD) is to take this to an extreme: provide limited functionality in the first release; based on experience with that release, find out what is needed and provide some of it in the next release, etc.

User roles are much the same here as in the analysis phase.

Some other issues

Some usability issues don't fit into the phase-based discussion above. Here are some of them.

Requirements

Should a requirement specification be made, or is the prototype sufficient? Experience shows that a prototype alone is dangerous for many reasons. For instance it does not specify things like response time, security requirements, maintainability, and other quality issues. More surprisingly, it often ignores important functional requirements or support for essential tasks. The reason is that the designers making the prototype became too enthusiastic about their work, and forgot essential requirements. Participatory design is no guarantee here, since users can become just as excited as developers.

So it is essential to produce and maintain an independent description of the requirements. In the above proposal, the task list, scenarios, and system goals serve this purpose. Checking against these items several times during development can keep the development on track.

If a subcontractor is to deliver the system, a more formal requirement document must be made. Should it describe the user interface, and in that case how? One possibility is to specify the user interface as a prototype. If this is done, the customer making the specification has essentially taken responsibility for the usability of the interface, so it is essential that he has made usability tests of it. Furthermore, a complying vendor has to develop the interface from scratch, and he cannot use a modified standard system.

What to do if the user interface is not specified as a prototype? One solution is to use the outcome of the analysis phase above, i.e. specify task lists, scenarios, system and usability goals as the essential part of the requirements. A prototype can still be part of the requirement document, but only as an *example* of a possible interface.

Object oriented development

The reader may have noticed that *objects* have hardly been mentioned above, apart from some reference to "use cases". The reason is simply that to the best of my experience, object-orientation in itself does not support or influence usability, just as the choice of programming language has little influence on usability. The use case approach, now being widely used in object-oriented development, is very close to task analysis, so some HCI techniques move into development in this indirect way.

The idea that "objects" in the real world are natural to users and can be modeled directly and shown on the user interface has very little success in real-life systems. On the contrary, all just modestly complex systems work with two independent sets of objects: those representing the "database" and those showing data to the user. In some critically important cases in each system, the relationship between the two sets is very complex with a many to many mapping between them. This means that the user interface can and should be designed independently of the object-oriented analysis of the application domain.

Conclusion

The worst danger in system development is the belief that a single technique or method solves all problems. Introducing some HCI into system development may remedy some usability problems, but not all. To get a reasonable result, many techniques are needed. Above, I have suggested about ten techniques and ten deliverables. Omitting any of them will reduce the usability of the product; adding more may increase the usability. Finding the right mix is essentially a cost/benefit issue.

References

Carlshamre, P. & Karlsson, J. (1996): A usability-oriented approach to requirements engineering. Proceedings of ICRE'96, pp. 145-152.

Cooper, A. (1996): Goal-directed software design. Dr. Dobb's Journal, September 1996, pp. 16-22.

Cuomo, D.L. & Bowen, C.D. (1994): Understanding usability issues addressed by three user-system interface evaluation techniques. Interacting with Computers, Vol.6, No.1, pp. 86-108.

Desurvire, H.W., Kondziela, J.M & Atwood, M.E. (1992): What is gained and lost when using evaluation methods other than empirical testing. Proceedings of HCI 92, pp. 89-102. Cambridge University Press.

Gould, J.D., Boies, S.J. & Lewis, C. (1991): Making usable, useful, productivity-enhancing computer applications. Comm. of the ACM, Jan 1991, Vol 34, No. 1, pp74-85.

Harwood, R.J. (1997): Use case formats: Requirements, analysis, and design. Journal of Object-oriented Programming, January 1997, pp. 54-57, 66.

Lauesen, S. & Harning, M.B. (1993): Dialogue design through modified dataflow and data modelling. In: Grechenig & Tscheligi (eds.): Human Computer Interaction, pp. 172-183, Springer-Verlag.

Lauesen, S., Harning, M.B. & Grønning, C. (1994): Screen design for task efficiency and system understanding. In S. Howard and Y.K. Leung (eds.): OZCHI 94 Proceedings, pp. 271-276.

Nygren, E., Lind M., Johnson, M. & Sandblad B.: The art of the obvious. CHI'92, pp. 235-239. ACM 0-89791-513-5/92/0005-0235.