The Human-Computer Interface is the System: A Plea for a Poor Man's HCI Component in Software Engineering Curricula

van der Veer, G.; van Vliet, H.

published in
Proceedings 14th Conference on Software Engineering Education & Training
2001

document version
Publisher's PDF, also known as Version of record

The Human-Computer Interface *is* the System;
A Plea for a Poor Man's HCI Component in Software Engineering Curricula

Gerrit van der Veer and Hans van Vliet
Division of Mathematics and Computer Science
Vrije Universiteit, Amsterdam, The Netherlands
Email: {gerrit, hans}@cs.vu.nl

Abstract

Most software engineering approaches restrict the user interface to everything a user may perceive or experience. As a result, it is often designed rather independently of the system's functionality. Chances are then that it does not get the attention it deserves. In the approach to software development we sketch, the design of the user interface and the design of the functionality go hand in hand. We give a number of examples of user interface problems, and illustrate how these can be caught early if a more integrated approach is taken. We conclude with an outline of a minimal course on human-computer interaction that we feel should be part of everyone's software engineering curriculum.

1 Introduction

> *We can't worry about these user interface issues now. We haven't even gotten this thing to work yet!*
> [Mulligan *et al.*, 1991]

The user interface of a system is important. About half of the code of an interactive system is devoted to the user interface. A recent study found that 60% of software defects arise from usability errors, while only 15% of software defects are related to functionality [Vinter *et al.*, 1996]. The quality of the user interface is a critical success factor. Good user interfaces increase the efficiency and productivity of their users, reduce errors and training time, and improve user acceptance.

What then *is* the role of the user interface? And where is it located? What should every software engineer know about user interfaces and human-computer interaction?

In 1983, a workshop on user interface management systems took place at Seeheim in West Germany [Pfaff, 1985]. At this workshop, a model was proposed which separates the application proper from the user interface. This model has become known as the **Seeheim model**. The Seeheim model describes the user interface as the outer layer of the system. This outer layer is an agent responsible for the actual interaction between the user and the application. It, in turn, consists of two layers supporting (1) the presentation – perceptible aspects including screen design and keyboard layout, and (2) the dialog, i.e. the syntax of the interaction including metacommunication (help functions, error messages, and state information). If the machine is said to apply a model of its human partner in the dialog, e.g. by choosing the user's native language for command names, this model is also located in the dialog layer.

This conceptualization of the user interface does not include the application semantics, or 'functionality'. In the Seeheim model, the tasks the user can ask the machine to perform are located in another layer, the application interface.

The Seeheim model provides some very relevant advantages. For example, we may provide the same outer layer to different applications. We may apply the same look and feel to a text editor, a spreadsheet, and so on, as in Microsoft products. In this way the user need not learn different dialog languages for different applications. Conversely, we may provide a single application to be implemented behind different outer layers, so as to allow different companies to adopt the same application with their own corporate interface style.

The Seeheim model, as well as other software engineering models such as the model-view-controller paradigm of Smalltalk restrict the user interface to everything that a user may perceive or experience. The SWEBOK view on human-computer interaction (HCI) is no exception to this line of thought. As a result of this limited view of what a user interface is, it is often designed rather independently of the system's functionality. Chances are then that the user interface does not get the attention it deserves.

The Stone Man version of SWEBOK [SWEBOK, April 2000] lists HCI and related knowledge domains as "related disciplines" of software engineering. We agree with this view, but not with the importance HCI has been assigned. According to appendix D of the SWEBOK report, all human-computer interaction aspects mentioned are of relevance only for the areas "Software Testing" and "Software Engineering Management". It seems SWEBOK takes the traditional view that human factors become relevant at the test phase of software engineering only, as an addition to other evaluation measures after the design phase is finished. Human factors input then is only needed to verify that the look and feel of the interface matches user needs, and it apparently does not relate to deeper design and requirements issues. Additionally, human factors are related to management which, of course, includes managing the (human) team of software engineers.

Our position is that this totally ignores the fact that many software engineering projects currently and in the future will aim to develop systems where human use and human factors in the context of use are decisive factors for product quality. As a consequence, we take a different stance in this paper. In the approach we sketch, the design of the interface and the design of the functionality go hand in hand. Hence the provocative title: 'The user interface *is* the system'. Within our approach, the design of the user interface replaces what we are used to call requirements engineering.

There are two main reasons for taking this broader view of what a user interface is:

- The system, and hence its interface, should help the user perform certain tasks. The user interface should therefore reflect the structure of the task domain. The design of tasks and the design of the corresponding user interface influence each other and should be part of the same cyclical process. Like quality, the user interface is not a supplement.
- The dialog and representation alone do not provide sufficient information to the user. In order to be able to work with a system, the user sometimes needs to know 'what is going on behind the screen'.

The viability of our approach is corroborated by various studies. [Lethbridge, 2000] for example addresses the question what knowledge is important to a software professional. He found that human-computer interaction is one of the topics with the widest educational knowledge gap. Practitioners found HCI a very important topic, of which they had learned very little in education. [Bevan, 1999] shows that the traditional quality assurance approach, emphasizing static and dynamic properties of software, needs to be expanded to incorporate quality in use aspects that address broader ergonomic issues.

Section 2 discusses various models that play a role in human-computer interaction. Section 3 discusses a number of user interface problems, and indicates the type of model mismatch they originate from. Section 4 sketches an eclectic approach to user interface design whose aim is to circumvent this type of model mismatches to occur. Finally, section 5 indicates how a minimal course on human-computer interaction might look like.

2 Models, models, models

The concept 'user interface' has several meanings. It may denote the layout of the screen, 'windows', or a shell or layer in the architecture of a system or the application. Each of these meanings denotes a *designer's* point of view. Alternatively, the user interface can be defined from the point of view of the intended *user* of a system. In most cases, users do not make a distinction between layers in an architecture and they often do not even have a clear view of the difference between hardware and software. For most users an information system as a whole is a tool to perform certain tasks. To them, the user interface *is* the system.

In this paper we use the term **user interface** to denote all aspects of an information system that are relevant to a user. This includes not only everything that a user can perceive or experience (as far as it has a meaning), but also aspects of internal structure and processes *as far as the user should be aware of them*. E.g., the user of a suite of programs including a text editor, a spreadsheet and a graphics editor should know that a clipboard is a memory structure whose contents remain unchanged until overwritten.

We define the user interface in this broad sense as the **user virtual machine** (UVM). The UVM includes both hardware and software. It includes the workstation or device (gadget) with which the user is in physical contact as well as everything that is 'behind' it like a network and remote data collections. We take the whole UVM, including the application semantics, as the subject of (user interface) design.

When thinking about user interface design, it is important to make a distinction between the user's mental model, the user virtual machine, and the conceptual model.

The *mental model* is a model in human memory. It is the user's model of the system he uses. It is based on education, knowledge of other systems, knowledge of the application domain, general knowledge about the world, etc. The mental model is used during interaction with the system, to plan actions and interpret system reactions. The mental model is often incomplete and inconsistent.

The *user virtual machine* (UVM) includes everything the user should know about the system in order to use it. It includes aspects ranging from the physical outlook of the computer and connected devices to the style of interaction and the form and content of the information exchange.

The *conceptual model* is the explicit model of the system created by designers and teachers. It is a consistent and complete representation of the system as far as relevant for the users. The conceptual model shows itself in the interface. If there is only one class of users, the user virtual machine and the conceptual model are the same. If there is more than one class of users (such as ATM clients, ATM maintainers, and lawyers), there is one UVM for each class, and the conceptual model is the union of those UVMs.

The central issue in human–computer interaction is to attune the user's mental model and the conceptual model as closely as possible. When this is achieved, the system becomes easier to learn and easier to use. When the models conflict, the user gets confused and starts making errors. Good design starts with the derivation of a conceptual model from an analysis of users and their tasks. This conceptual model is then built into the system (the UVM) in such a way that it induces

adequate mental models in the users.

3 Model mismatches

Many user interface problems can be explained in terms of mismatches between the mental model, user virtual machine and conceptual model. They may concern plain representation issues as well as deeper problems relating to user task structure and semantics. In this section, we give a few examples to illustrate the range of issues involved.

3.1 Task delegation

The Dutch railway company planned to remove the human operated ticket desks at small railway stations. In order to gradually pursue this business goal, a first step was to develop ticket selling machines for day trips, to be placed on the platforms. The early machines provided users with the possibility to buy tickets to all railway stations in the country. However, the journey had to start at the location the ticket was issued. The machine did not mention this and neither did the information campaign designed to change travelers' behavior to buy tickets at the machine instead of at the desk. As a result, many travelers with a month pass for a certain trajectory were fighting the machines when they wanted to extend their journey to a destination not covered by their pass. Consequently, the intended buyer behavior did not develop according to plans.

The requirements defined for the first generation machines did not take into account actual traveller behavior patterns, even at the "high" level of planning and economic buying strategies, which could have been found out with early user studies. This is an example of a mismatch between the mental model users had of the machine and the conceptual model built into it.

3.2 Task semantics, functionality

[Sommerville *et al.*, 1994] discuss an air traffic control system redesign project. Part of their discussion concerns the question what objects are relevant for the task space, and, hence, should be part of the system's task model.

Traditionally, air traffic controllers in many countries use paper "flight strips" symbolising individual aircraft, with a number of attributes of that flight printed on the strip. While handling the flight, the strips get notes scribbled on them, and they get physically positioned and manipulated at the work desk area of the group managing the space the aircraft is in. All information on the strip, as well as its history, obviously can be handled by creating an electronic record and manipulating that record. At the abstract level, the functionality is obvious, and its automation seems fine. Human users, though, in a situation of mental overload turn out to use elements in the physical work situation to help them stay aware both of the level of work load in general, and of the individual subtasks that are currently running. The paper flight strips turned out to have precisely that function in being visibly available, together with their attributes, being grouped and otherwise manipulated and making the users literally "feel" the work in progress.

Observing and analysing users in actual work situations, using special human factors techniques, provides insight in task semantics even before any detail design decisions have to be made. The mental model of the task space in this case requires that certain information be permanently perceptible and manipulatible by the users. The (first version of the) user virtual machine did not provide this functionality.

3.3 The syntax level of the dialogue

The dialog design should relate to human goal-driven behaviour. E.g., if a person's goal in using an ATM is to get money, the user tends to stop his dialog with the machine as soon as he receives that money. Identification of the user with the help of a bank card and PIN is only a lower-level goal, triggered by the need of the system to replace (physical) identification and verification by bank employees during transactions. Users will accept this goal only as far as, and as long as, it helps them reach their primary goal, getting money. The first generation of ATM's often returned the bank cards only after the whole transaction was finished, which resulted in a lot of users forgetting to take out their card.

Psychological analysis of the users' goal structure in task situations will reveal what dialog structure best matches expected behavior. By using appropriate human factors techniques, this insight can be developed before any working prototype of the system is implemented. In the ATM case, the aim should be to develop a user virtual machine and conceptual model that invites users to develop a mental model that includes card handling (as a subgoal and subtask).

3.4 Representation level

Specifying the system's image (the perceptible user interface) should take into account a whole series of aspects:

- generic issues like human perception characteristics in relation to the actual work situation, the cultural meaning of symbols, etc;
- the task- and situation-dependent need for specific information, like the actual system state, options for next user actions, ranges of values to be put in, etc.

Only an analysis of the system under development with the help of early usability tests will help the designer to make a good choice. The types of users should be taken into account in these studies. E.g., if a system is designed to provide local information to be installed in a public place, some experimentation might show what questions are the first ones users would want to ask, as well as how best to state each individual question. In a menu type dialog these questions should be immediately recognizable at the user interface. Depending on the local situation the options may be in one or more languages, or represented by pictograms that appropriately reflect the cultural symbol values of the types of users expected. If 95% of the people requesting this information does not speak the local language, it does not make sense to use that language to indicate the option to choose another language.

On-the-spot analysis of user behavior and user characteristics, with the help of an early prototype, prevents the development of representations that are unusable, or otherwise do not fit for the potential user population. Again, the user virtual machine should refer to functionality that relates to the users' "natural" mental models of or in the current situation.

4 A unified process model

Traditional user interface design mainly concerns the situation of a single user and a monolithic system. In current applications, computers are mostly part of a network, and users are collaborating, or at least communicating, with others through networks. Consequently, the UVM should include all aspects of communication between users as far as this communication is routed through the system. It should also include aspects of distributed computing and networking as far as this is

relevant for the user, such as access, structural, and time aspects of remote sources of data and computing. For example, when using a Web browser, it is relevant to understand mechanisms of caching and of refreshing or reloading a page, both in terms of the content that may have changed since the previous loading operation and in terms of the time needed for operations to complete.

These newer types of application bring another dimension of complexity into view. People are collaborating in various ways mediated by information technology. Collaboration via systems requires special aspects of functionality. It requires facilities for the integration of actions originating from different users on shared objects and environments, facilities to manage and coordinate the collaboration, and communication functionality. Such systems are often denoted as **groupware**. Modern user interface design techniques cater for both the situation of the classical single user system and groupware. We expect this distinction to disappear in the near future.

There are several classes of stakeholders in system development. These include at least the clients, i.e. the people or organizations that pay for the design or acquisition of systems, and the users, i.e. the people or groups that apply the systems as part of their daily work, often referred to as the **end users**. In many situations there will be additional classes of stakeholders to cater for, like people who are involved in maintaining the system, and people who need traces or logs of the system to monitor cases of failure or abuse, such as lawyers.

Making a distinction between classes of stakeholders does not solve the problem of user diversity. In complex systems design, we are confronted with different end users playing different roles, as well as end-user groups that, as a group, have knowledge or a view on the task domain that need not be equivalent to the (average or aggregated) knowledge and views of the individuals.

4.1 Design as an activity structure

Viewing design as a structure of interrelated activities, we need a process model. The model we use will be familiar to software engineers: it is a cyclical process with phases devoted to analysis, specification, and evaluation. Figure 1 depicts this process model.

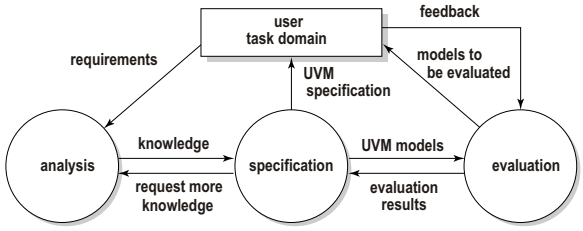

Figure 1. A process model for user interface design

Analysis Since the system to be developed will feature in a task situation, we start with task analysis. We further structure this activity into the development of two models, labeled task model 1 and task model 2. The first one models the current task situation. Task model 2 models the task domain of the future situation, where the system to be developed will be used, including changes in the organization of people and work procedures. The relationship between task models 1 and 2 reflects the change in the structure and organization of the task world as caused by the implementation of the system to be developed. This difference is relevant both for the client and the user.

The development of task model 2 from task model 1 uses knowledge of current inadequacies and problems concerning the existing task situation, needs for change as articulated by the clients, and insight into current technological developments. For a detailed discussion of task analysis techniques, see [Kotonya and Sommerville, 1997], [SWEBOK, April 2000, chapter 2] or [van Vliet, 2000, chapter 9].

Specification The specification of the system to be designed is based on task model 2. It has to be modeled in all details that are relevant to the users, including cooperation technology and user-relevant system structure and network characteristics. Differences between the specification of the new system (the user's virtual machine or UVM) and task model 2 must be considered explicitly and lead to design iteration.

Evaluation The specification of the new system incurs many design decisions that have to be considered in relation to the system's prospective use. For some design decisions, guidelines and standards might be used as checklists. In other situations, formal evaluation may be applied, using formal modeling tools that provide an indication of the complexity of use or learning effort required. For many design decisions, however, evaluation requires confronting the future user with relevant aspects of the intended system. Some kind of prototyping is a good way to confront the user with the solution proposed.

The design of an interactive system as discussed here is very akin to the requirements engineering activity as discussed in [Loucopoulos and Karakostas, 1995] or other textbooks on that subject. The terminology is slightly different and reflects the user-centered stance taken in this paper. For example, 'analysis' sounds more active than the commonly used term 'elicitation'. 'Evaluation' entails more than 'validation'; it includes usability testing as well. Finally, we treat the user and the task domain as one entity from which requirements are elicited. In the approach advocated here, the user is observed *within* the task domain.

4.2 Design as multi-disciplinary collaboration

The main problem with the design activities discussed in the previous section is that different methods may provide conflicting viewpoints and goals. A psychological focus on individual users and their capacities tends to lead to Taylorism, neglecting the reality of a multitude of goals and methods in any task domain. On the other hand, sociological and ethnographical approaches towards groupware design tend to omit analysis of individual knowledge and needs. Still, both extremes provide unique contributions.

In order to design for people, we have to take into account both sides of the coin: the individual users and clients of the system, and the structure and organization of the group for which the system is intended. We need to know the individuals' knowledge and views on the task, on applying the technology, and the relation between using technology and task-relevant user characteristics (expertise, knowledge, and skills). With respect to the group, we need to know its structure and dynamics, the phenomenon of 'group knowledge' and work practice and culture.

For example, in a traditional bank setting, the client and the bank employee are on different sides of a counter. The bank employee is probably using a computer, but the client cannot see the screen, and does not know what the clerk is doing. In a service-oriented bank setting, the clerk and client may be looking at the screen *together*. They are together searching for a solution to the client's question. This overturns the existing culture of the bank and an ethnographer may be asked to observe what this new set up brings about.

The general framework for our approach to user interface design is depicted in figure 2. It is a

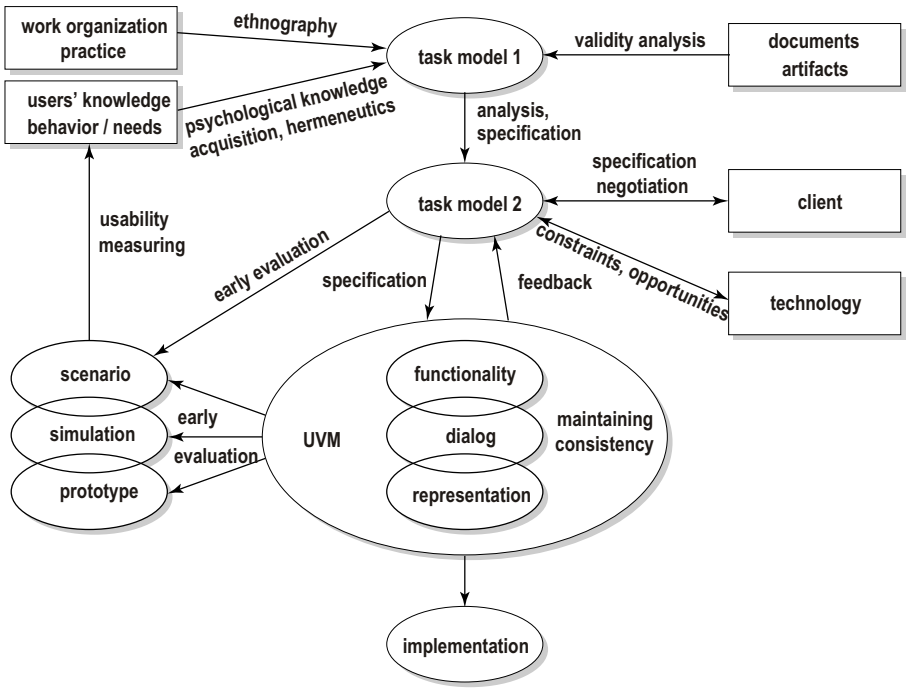

Figure 2. Structure of design team activities

refinement of figure 1, emphasizing the specialties involved in carrying out different activities. It shows the structural relations between design activities on the one hand, and the involvement of users and clients, and work practice on the other hand. Task model 1 is based on knowledge of single users (psychological variables, task-related variables, knowledge and skills) and on complex phenomena in the task situation (roles, official and actual procedures, variation in strategies, and variation in the application of procedures). The integration of this insight in a model often does not provide a single (or a single best) decomposition of tasks and a unique structure of relationships between people, activities, and environments. The model often shows alternative ways to perform a certain task, role-specific and situation-specific methods and procedures, and a variety of alternative assignments of subtasks to people. For example, the joint problem-solving approach to the bank counter as sketched above cannot be applied to the drive-in counter of the bank. The drive-in counter requires a different approach and a different user interface.

Again, when detailed design decisions are being considered, early evaluation needs to include analytical methods (formal evaluation and cognitive walkthrough techniques) in combination with usability testing where users in different roles are studied both in the sense of traditional individual measures and in the sense of ethnographic interaction analysis.

5 A poor man's HCI curriculum

An approach to system design as sketched in the previous section requires that every designer of software has a basic understanding of underlying issues. In this section we sketch a minimal course on human-computer interaction that should be part of everyone's software engineering curriculum.

For an example of how to implement this, see [van Vliet, 2000, chapter 16]. The main ingredients of such a minimal course are:

- An introduction to the issue of human use of computers and an explanation of the main concepts of human-computer interaction (HCI).
  Most users nowadays are not IT experts. They need knowledge about relevant aspects of the system not all of which can be made visible, at least not all at once. So users need to build and maintain a mental model of the system, in order to plan ahead, to execute task delegation to the system, to evaluate results, and to interpret unexpected events.
  The contents of this mental model should be equivalent to all the relevant knowledge of a certain user group, which leads to the concept of the user's virtual machine (UVM). The finished specification of the UVM can be seen as a complete description of the system from the point of view of the user and, at the same time, as a subset of the requirements for the development of the "actual" system.
  The problem of HCI has been approached from various relevant points of view. An overview of the most important streams or "schools" in human-computer interaction shows how they developed from different disciplines like:
  - Applied cognitive psychology: focusing on the human user's behavior, characteristics and needs;
  - Ergonomics: aiming first of all at adapting technology to human possibilities;
  - Software engineering: developing views on the architecture of user interfaces and related engineering processes;
  - Graphical design and other "arts and crafts": applying knowledge developed in typography, theatre, cinematography, and advertisement, on how to present information in such a way that the "audience" gets the message;
  - Interaction design: creative design methods based on an understanding of the task dynamics.
- Some basics of human information processing as developed in cognitive psychology.
  This part certainly can not provide a general psychological theory, and can only be based on a choice for one of the various existing models. Topics to be covered include:
  - Perception encoding phenomena, both bottom-up/data-driven processes like the semi-automatic formation of perceptual "gestalts", and top-down knowledge-driven processes like recognition;
  - Physical behavior in the sense of coordination processes at the level of keystroke interaction, including an explanation of Fitts' law;
  - Issues of human attention and the model of a "central executive" or cognitive processing unit, with an illustration of the limited capacity of explicit thought processes.

  This section should also cover decision making, as well as the instantiation of mental models triggered by the situation and current human needs:
  - The model of the restricted capacity of working memory, the meaning of chunks of information and the importance of expertise (and information chunking) for high-capacity information processing;
  - An account of long-term memory, the distinction between semantic knowledge structures and episodic memory, the individual differences in preference for image or verbal

knowledge processing, and the models for information recognition and retrieval. This section should include an account of planning and of learning.

Each of the psychological issues mentioned can be illustrated in a way that shows its relevance and importance for the design of human-machine interaction in which the capacities and limitations of human users are optimally matched by the machine.

- Ergonomics of information systems: design as an issue of mutual adaptation of human users and technology.
  Some attention should be given to the notion of human cognitive functions that may adapt to the system (by learning, instruction, and exploration) and others that the interface needs to be adapted to (general human capacities and constraints, individual differences in cognitive styles). Examples of concepts to cover:
  - How to introduce information systems, when to provide instruction and when to provide safe interfaces for user exploration;
  - How to design self explaining, ”intuitive” systems, including the concept of ”affordance”;
  - The notion of ”impulsiveness”, and the need for undo facilities;
  - How to cope with people with low spatial ability: providing navigation and overview facilities.
- An account of designing the User Virtual Machine in a process that provides a structure of design activities.
  Figure 2 could be a guideline for this section. At a global level, design activities for interactive systems can be subdivided into groups of techniques like analysis, specification, and evaluation, in an iterative process. For each of these groups, some example techniques can be provided, including the theory they are based on, and in some cases accompanied by a small exercise, e.g.:
  - Modeling the current task situation (task model 1, see section 4.1) is in some cases based on the knowledge elicitation from expert users. The Graesser Questioning technique, based on the psychology of long-term memory, can be explained and is simple enough to be demonstrated or practiced in a classroom situation. Alternative techniques for acquiring task knowledge can be found in ethnography (relevant depending on the type of work culture). A technique like interaction analysis can be explained, and demonstrated by analyzing an ethnographic video record as a group exercise.
  - Deciding on dialog styles (as part of the UVM) may well depend on user- and task-characteristics. Based on a well described user group and a fragment of a task model, Mayhew’s technique for choosing optimal interaction may be demonstrated or performed as a classroom exercise [Mayhew, 1992]. In the case of a specification of the UVM there is also the opportunity to show the feasibility of applying user interface design patterns for choosing among representations or dialog styles.
  - Evaluation techniques are frequently based on available representations of the analysis or specification. While developing task model 2 or in the early stages of specifying the functionality of the UVM, scenarios are an adequate representation that can provide insight in the development even if details are still to be filled in. A video tape of such a scenario can be the basis for a claims analysis, to be performed as a group exercise. If the specification of UVM details has proceeded far enough to produce a rapid interactive

prototype, students can be taught to perform a cognitive walkthrough or to apply a heuristic usability checklist.

Obviously, strong emphasis should be put on those evaluation techniques in all phases where prospective users of the interactive system are involved. It will probably not be possible to provide for actual experience with these techniques, but some tools can be shown, like a video clip of a usability lab situation and a fragment of a subjective usability rating scale.

6 Conclusions

In the received view of software engineering, the design of the user interface is seen as a separate activity, not in the mainstream requirements engineering – design – implementation – testing process model. In this paper, we argue for an eclectic approach, in which user interface issues are given attention from the very beginning. Slightly provocative, we state that the user interface *is* the system. This requires that software engineers have a basic understanding of human factors issues involved and, consequently, that a software engineering curriculum contains, as a minimum, an introductory course on human-computer interaction. An outline for such a course is given in this paper as well.

References

[Bevan, 1999] N. Bevan. Quality in Use: Meeting User Needs for Quality. *Journal of Systems and Software*, 49(1):89–96, 1999.

[Kotonya and Sommerville, 1997] G. Kotonya and I. Sommerville. *Requirements Engineering, Processes and Techniques*. John Wiley & Sons, 1997.

[Lethbridge, 2000] T.C. Lethbridge. What Knowledge Is Important to a Software Professional? *IEEE Computer*, 33(5):44–50, 2000.

[Loucopoulos and Karakostas, 1995] P. Loucopoulos and V. Karakostas. *Systems Requirements Engineering*. McGraw-Hill, 1995.

[Mayhew, 1992] D.J. Mayhew. *Principles and Guidelines in Software User Interface Design*. Prentice-Hall, 1992.

[Mulligan *et al.*, 1991] R.M. Mulligan, M.W. Altom, and D.K. Simkin. User Interface Design in the Trenches: Some Tips on Shooting from the Hip. In *Proceedings CHI'91*, pages 232–236. ACM, 1991.

[Pfaff, 1985] G.E. Pfaff, editor. *User Interface Management Systems*. Springer Verlag, 1985.

[Sommerville *et al.*, 1994] I. Sommerville, R. Bentley, T. Rodden, and P. Sawyer. Cooperative System Design. *The Computer Journal*, 37(5):357–366, 1994.

[SWEBOK, April 2000] *Guide to the Software Engineering Body of Knowledge, Stone Man Version*. Software Engineering Coordinating Committee, IEEE, April 2000.

[van Vliet, 2000] H. van Vliet. *Software Engineering: Principles and Practice*. John Wiley & Sons, second edition, 2000.

[Vinter *et al.*, 1996] O. Vinter, P.M. Poulsen, and S. Lauesen. Experience Driven Software Process Improvement. In *Software Process Improvement*, Brighton, 1996.