A Plea for a Poor Man's HCI Component in Software Engineering and Computer Science Curricula;

After all: The Human-Computer Interface *is* the System

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

SUGGESTED SHORT TITLE: A Poor Man's HCI Component

Abstract

Most software development approaches and curricular guidelines seem to ignore the fact that in many software systems the user interface is a decisive factor for product quality. 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 or computer science 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). According to (Nielsen, 2000), adequate attention to the quality of the user interface can increase sales of e-commerce sites by 100 percent. For Web-based systems, usability goals are business goals.

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

The Seeheim model (Pfaff, 1985), as well as other software development models such as the model-view-controller paradigm of Smalltalk restrict the user interface to everything that a user may perceive or experience. The design of the user interface is then seen as a separate activity, not interwoven in the mainstream software development process. At best, it is considered to be part of detailed design. In a recent article in the leading *Requirements Engineering* journal, this situation was aptly phrased as "the user interface designer exists in the world of users and their tasks, and is primarily concerned with design activities that occur after functional requirements have been defined" (Overmyer, 2000). 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.

To improve upon the present state of the practice w.r.t. user interface concerns during software development, we should integrate appropriate techniques into our software development process. The place to start doing so is software engineering and computer science education.

How does the software engineering community fare in this respect? The two most relevant sources for an answer to this question are SWEBOK and SEEK. The Guide to the Software Engineering Body of Knowledge (SWEBOK) is a result of the Software Engineering Coordinating Committee, a Joint IEEE Computer Society - ACM committee. It reflects a widely agreed-upon view on what a software engineer should know. The Software Engineering Education Knowledge (SEEK) can be viewed as the education counterpart of SWEBOK, aimed

[1]This is an updated and extended version of (van der Veer and van Vliet, 2001)

at providing guidance for undergraduate curricula. It is being developed under the guidance of a joint task force of the IEEE Computer Society and ACM.

The trial version of SWEBOK (Abran et al., 2001) lists HCI as a "related discipline" of software engineering. It furthermore states that user interface design deals with specifying the external view of the system (Abran et al., 2001, p 3-8). A similar position is taken in the first draft of the SEEK report (Sobel, 2002), where user interface design entails topics like "use of modes", and "response time and feedback".

For Computer Science the situation is not much different. The ACM-IEEE Computing Curriculum guidelines as formulated in the "Steelman Draft" (CC-SE, 2001) shows two units related to HCI in the core curriculum: a six hour introduction to the foundations of HCI, and a two hour unit on "building a simple graphical user interface". All these sources reflect the limited view of the user interface as alluded to above.

Our position is that this totally ignores the fact that many software development 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 the approach we sketch, the design of the interface and the design of the functionality go hand-in-hand. Hence the provocative subtitle: '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 iterative refinement 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 plea to pay more attention to HCI in software engineering and computer science curricula 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.

The remainder of this paper is organized as follows. 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 proposes a minimal course on human-computer interaction.

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.

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 this has a meaning for the user within the present context), but also aspects of internal structure and processes *as far as the user should be aware of them.* For example, 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 being used. 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 responses. 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.

This view is related to that of Donald Norman. In (Norman, 1986), he uses the terms *user’s model*, *design model* and *system image.* Our mental model equals Norman’s user’s model; in fact, Norman uses both terms interchangeably. Our conceptual model equals Norman’s design model, while our user virtual machine includes both Norman’s system image and the relevant conceptual knowledge a user needs but which is not perceptible in the system image (such as an understanding of what goes on in an invisible clipboard). Note that Norman uses the same terms to denote (slightly) different things in some of his other publications.

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. Model matches, i.e. examples where the different models are well-aligned in the solutions chosen, can be found in pattern collections for user interface design, e.g. (van Duyne et al., 2002) and (van Welie et al., 2000).

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 route were fighting the machines when they wanted to extend their journey to a destination not covered by their pass. Clearly, 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 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 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.

This example shows that observing and analysing users in actual work situations 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.

3.3 The syntax level of the dialogue

The dialog design should relate to human goal-driven behaviour. Since 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).

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 to the user, such as access, structural, and temporal 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 applications 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 can be applied both for the situation of the classical single user system and for groupware. We expect this distinction to disappear in the near future.

4.1 Design as an activity structure

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

Analysis Since the system to be developed will be used for specific tasks, we start with task analysis. In our view, a task model should include both the task knowledge, the relevant knowledge of the user(s), and the knowledge of the context of use. In fact, the analysis activity should be structured into two steps:

1. modeling the current situation, a descriptive activity; and
2. modeling the envisioned future situation where the system to be developed will be used, including changes in the organization of people and work procedures.

Consequently, analysis is a process where two consecutive task models are being developed. The relationship between the model of the current task situation and the model of the envisioned future task situation 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 the envisioned task model from the current task model 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), (Abran et al., 2001, chapter 2) or (van Vliet, 2000, chapter 9).

Specification The specification of the system to be designed is based on the task model of the envisioned new situation. 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 (UVM) and task model for the new situation must be considered explicitly and lead to iterative refinement.

Evaluation The specification of the new system requires 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. Prototyping facilitates early and frequent user feedback. A prototype allows experimentation with selected elements or aspects of the UVM. It enables imitation of (aspects of) the presentation interface, enables the user to express himself in (fragments of) the interaction language, and can be used to simulate aspects of the functionality, including organizational and structural characteristics of the intended task structure.

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 the 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 may lead to Taylorism[2], 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 and valuable contributions.

In order to design for people, we have to take into account both 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. These aspects are needed in order to acquire insight into both existing and projected task situations where (new) cooperation technology is introduced. Insight from both individual and group perspectives are also needed for design decisions. Consequently, in prototyping and field-testing we need insight in acceptance and use of individuals, and in the effect of the new design on group processes and complex task dynamics.

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 or computer science 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 HCI.

  Most users nowadays are not IT experts. They need knowledge about relevant technical 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.

  This mental model should contain all the relevant knowledge of the target user group, which leads to the UVM concept. 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.

[2]Around the turn of the 20th century, Taylor introduced the notion of 'scientific management', in which tasks are recursively decomposed into simpler tasks and each task has one 'best way' to accomlish it. By careful observations and experiments, this one best way can be found and formalized into procedures and rules. Scientific management has been successfully applied in many a factory operation.

The challenges of HCI have been approached from various points of view. An overview of the most important approaches in HCI reveals the influences and contributions of different disciplines:

- Applied cognitive psychology: focusing on the human user's behavior, characteristics and needs (Locke and Lathman, 1990). This domain provides insight in the relation between human needs and goals that will develop as a result of these needs. Goals may trigger behavior based on the situation at hand. For example, if someone is hungry (a need), the goal of eating some food will develop. This goal, in a situation of spotting an advertisement of a pizza delivery on a web page and a credit card in the purse, may trigger the behavior of ordering a pizza. General insight in this mechanism is applicable to designing feasible triggering situation cues that evoke the type of behavior aimed at. Design could for example be aimed at enabling and facilitating behavior, warning for unexpected or undesirable effects, or 'seducing' users to act in a certain way.
- Ethnography and ethno-methodology: approaching human behavior from the point of view of the actual situation and culture (Jordan, 1996). In this discipline, the total environment, its history, and its group processes are considered of main importance for understanding (and, hence, influencing) human activities, including the use of technology. One of the techniques developed in this field, interaction analysis, aims at understanding the basic patterns in interaction between humans, artifacts and the situation. Based on the patterns found, technology can be designed to facilitate the interaction or to elicit an intended change of pattern. E.g., if the generic pattern found in buying interactions shows that prior knowledge of the identity of the buyer allows to skip verification of solvency, e-commerce systems may be designed to provide this shortcut (not asking for and checking credit card information when the machine and user ID of the buyer is recognized).
- Software engineering: developing views on the architecture of user interfaces and related engineering processes (van Vliet, 2000). Software engineering methods emphasize various notions of 'separation of concerns', to ease the development and evolution of systems. The Seeheim model and the Model-View-Control pattern discussed earlier are prime examples hereof.
- 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 (Laurel, 1990). The art of theatre staging teaches us how to manipulate attention, as well as how to provide awareness of relevant situation aspects (what are the characteristics of the location of this scene, who else is aware of what action is currently being performed, etc.). Cinematography (May and Barnard, 1995) shows us how to represent continuation and disruption in time (by preventing the camera to cross the (invisible) lines connecting interacting partners or by crossing those lines, respectively). The same art domain found out how to represent causality between action and result in subsequent scenes, by locating the representation of the result in the area towards which the general action is moving. Design of animation and design of dialogue scenarios will benefit from applying this kind of knowledge.
- Interaction design: creative design methods based on an understanding of the task

dynamics (Beyer and Holtzblatt, 1998). The creativity that can be found in this discipline is at least partly based on insight in the feasibility of certain types of humor and other emotional cues in relation to cultural values of the spectators. Applying this knowledge allows the designer to trigger and manipulate emotions that, in turn, will trigger intended user behavior, like buying, returning to the web site, or reading documentation.

- 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 (Goldstein, 2000). Designers have to understand if and how any representation will trigger perception. This is a complex phenomenon composed of both the automatic and semi-automatic mechanisms of stimulus coding and transformation into recognizable elements, and the mechanism of identification or construction of meaning based on knowledge and expectation. This insight will help in applying colors, shapes, contrasts, movement and sound, in relation to the semantics of the interaction in which the design features. In some cases correct or quick perception is important for safety. In other situations perception phenomena may be voluntarily applied to trigger the interest or induce emotions (e.g., movement in the periphery of the visual perceptual field may attract attention or may provide awareness that something meaningful will be available soon).
  - Physical behavior in the sense of coordination processes at the level of keystroke interaction, including an explanation of Fitts' law (MacKenzie, 1992). These mechanisms explain the speed and accuracy of simple user actions like keystrokes and pointing. Hitting a key is quicker and more precise if the layout of the user's input device (e.g. a mobile phone) is based on a systematic spatial grid. Up to a certain size defined by physical characteristics of the human body, small keys are fine as long as they are laid out in systematic rows and columns. Other types of phenomena in the domain of human physical behavior account for the learning of complex movements, like motor memory in typing skills.
  - 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 (Baddeley, 1990). Humans can only pay attention to a single phenomenon at a time. Consequently, in many cases humans should be able (i.e., be trained) to perform some activities automatically. Also, the nature of processes may develop (by learning). A regular user of an ATM will just type his PIN, which is qualitatively different from, and requires much less thought capacity than, remembering and typing four consecutive digits. The designer should be aware of the level of learning that may be reached by the intended user. In some cases, special short cuts may be provided for the expert users.

  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. Humans are able to keep up to about 7 'chunks' of information ready in mind (Miller, 1956). Expertise in a certain domain may in fact be defined as the amount of relevant domain knowledge that has been structured in hierarchical concepts. In this way, an air traffic control operator may have a single concept for issuing a rerouting command. In situations of high workload this chunk may be sufficient to perform the task, but if needed the operator could elaborate and decompose the knowledge on this command into other chunks that each represent a subtask. Decomposition will mainly be relevant in case of exceptions, where defaults are not valid or rules do not apply. In design it is needed to keep our memory restrictions in mind. In fact, for each user action the designer should know what information may be needed for the user in order to make his decision, and where the user could find this information. If this might cause a working memory problem, it will pay off to provide the information through the system.
  - An account of long-term memory, the distinction between semantic knowledge structures and episodic memory, and the models for information recognition and retrieval (Raaijmakers and Shiffrin, 1992). This should include an account of planning and learning. Episodic memory indicates knowledge of actual episodes in the individual's history, including knowledge of past interactions with the system, like "yesterday just before leaving the office I deleted all messages from my inbox". Semantic knowledge includes the individual's knowledge of concepts and their relations, like his assumptions of the conventions of MS Office. The notion of 'mental model' indicates the model that users of complex systems need in order to plan, execute, and evaluate task delegation to complex systems. Mental models are considered to be instantiated based on knowledge in long-term memory, and triggered and shaped by needs of the moment and characteristics of the current situation, including the system state visible to the user. Consequently, design includes considering how to stimulate the development of adequate knowledge and how to trigger relevant mental models when complex interaction is required.

  Each of the psychological issues mentioned can be illustrated in a way that shows its relevance and importance for the design of HCI 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 (through 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. Users generally do not want to read extensive manuals before starting to use a system. They generally prefer to learn by doing (Carroll, 1990).

  - How to design self explaining, "intuitive" systems, including the concept of "affordance" (Norman, 1988). Affordances provides clues to operations. A door with a knob suggests that the door can be opened by turning the knob. Likewise, interface elements like buttons and rulers provide clues on how to interact with the system.
  - The notion of "impulsiveness", and the need for undo facilities. People feel much more comfortable with systems if they know their actions are reversible. They are then willing to explore the system.
  - How to cope with people with low spatial ability: providing navigation and overview facilities.

- An account of designing the UVM in a process that provides a structure of design activities.

  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.:

  - Analysis is in some cases based on the knowledge elicitation from expert users. The Graesser Questioning technique (Sebillotte, 1988), 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.
  - Specification of 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. 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 (Carroll, 1995). 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 an 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 traditional view of software engineering, the design of the user interface is seen as a separate activity, not in the mainstream software development process model. In this paper, we argue for an eclectic approach, in which user interface issues are given attention from the very beginning. We put forward a rather provocative observation, namely that the user interface *is* the system. This view requires that software developers have a basic understanding of human factors issues involved and, consequently, that a software engineering or computer science curriculum contains, as a minimum, an introductory course on human-computer interaction. This paper provides an outline for such a course.
Disclaimer: There is obviously much more to teach and learn about HCI. Our university, for instance offers a variety of courses on this topic, such as Task Analysis, User-Interface Design, and Graphical Design. Also, ACM's Special Interest Group on Computer-Human Interaction has published a model curriculum in HCI (SIGCHI92, 1992). Finally, the IEEE/ACM Computing Curricula 2001 offers another possible course model (CC-HCI, 2001). Hopefully, this paper makes a strong case for a minimal HCI-component in *every* software engineering or computer science curriculum.

References

Abran, A., Moore, J., and Dupuis, R., editors (May 2001). *SWEBOK: Guide to the Software Engineering Body of Knowledge: Trial Version 1.00.* IEEE.

Baddeley, A. (1990). *Human memory: Theory and practice.* Erlbaum.

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

Beyer, H. and Holtzblatt, K. (1998). *Contextual Design : A Customer-Centered Approach to Systems Designs.* Morgan Kaufmann.

Carroll, J. (1990). *The Nurnberg Funnell: Designing Minimalist Instruction for Practical Computer Skill.* MIT Press.

Carroll, J. (1995). *Scenario-Based Design.* John Wiley & Sons.

CC-HCI (2001). Computing Curricula 2001, Approved Final Draft of the Computer Science Volume. *Joint ACM-IEEE Task Force on Computing Curricula*, (20 January 2003; www.computer.org/education/cc2001/final/cs250.htm).

CC-SE (2001). Computing Curricula 2001, Software Engineering (Steelman Draft). *Joint ACM-IEEE Task Force on Computing Curricula*, (20 January 2003; www.computer.org/education/cc2001/steelman/cc2001/).

Goldstein, B. (2000). *Handbook of Perception.* Blackwell.

Jordan, B. (1996). Ethnographic Workplace Studies and CSCW. In Shapiro, D., Tauber, M., and Traunmueller, R., editors, *The Design of Computer Supported Cooperative Work and Groupware Systems*, pages 17–42. North-Holland, Amsterdam.

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

Laurel, B., editor (1990). *The Art of Human-Computer Interface Design.* Addison-Wesley.

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

Locke, E. and Lathman, G. (1990). *A theory of goal setting and task performance.* Prentice-Hall.

Loucopoulos, P. and Karakostas, V. (1995). *Systems Requirements Engineering.* McGraw-Hill.

MacKenzie, I. (1992). Fitts' law as a research and design tool in human-computer interaction. *Human-Computer Interaction*, 7:91–139.

May, J. and Barnard, P. (1995). Cinematography and Interface Design. In *et al.*, K. N., editor, *Human-Computer Interaction: Proceedings Interact'95*, pages 26–31. Chapman and Hall.

Mayhew, D. (1992). *Principles and Guidelines in Software User Interface Design.* Prentice-Hall.

Miller, G. (1956). The Magical Number Seven, Plus or Minus Two: Some Limits on our Capacity for Processing Information. *Psychological Review*, 63:81–97.

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

Nielsen, J. (2000). Web Resarch: Believe the Data. *Alertbox*, (11 July 1999; www.useit.com/alertbox/990711.html).

Norman, D. (1986). Cognitive Engineering. In Norman, D. and Draper, S., editors, *User Centered System Design*, pages 31–61. Lawrence Erlbaum Associates.

Norman, D. (1988). *The Psychology of Everyday Things.* BasicBooks.

Overmyer, S. (2000). What's Different about Requirements Engineering for Web Sites? *Requirements Engineering*, 5(1):62–65.

Pfaff, G., editor (1985). *User Interface Management Systems.* Springer Verlag.

Raaijmakers, J. and Shiffrin, R. (1992). Models for recall and recognition. *Annual review of psychology*, 43:205–234.

Sebillotte, S. (1988). Hierarchical Planning as a Method for Task-Analysis: The Example of Office Task Analysis. *Behaviour and Information Technology*, 7(3):275–293.

SIGCHI92 (1992). ACM SIGCHI Curricula for Human-Computer Interaction. Technical report.

Sobel, A., editor (August 28, 2002). *Computing Curricula – Software Engineering Volume, First Draft.* CCSE Steering Committee, `http://sites.computer.org/ccse/`.

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

van der Veer, G. and van Vliet, H. (2001). The Human-Computer Interface *is* the System: A Plea for a Poor Man's HCI Component in Software Engineering Curricula. In Ramsey, D., Bourque, P., and Dupuis, R., editors, *Proceedings 14th Conference on Software Engineering Education & Training, February 19-21, 2001, Charlotte, USA*, pages 276–286. IEEE.

van Duyne, D., Landay, J., and Hong, J. (2002). *The Design of Sites*. Addison-Wesley.

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

van Welie, M., van der Veer, G., and Eliëns, A. (2000). Patterns as Tools for User Interface Design. In Vanderdonckt, J. and Farenc, C., editors, *International Workshop on Tools for Working with Guidelines*, pages 313–324. Springer Verlag.

Vinter, O., Poulsen, P., and Lauesen, S. (1996). Experience Driven Software Process Improvement. In *Software Process Improvement*, Brighton.

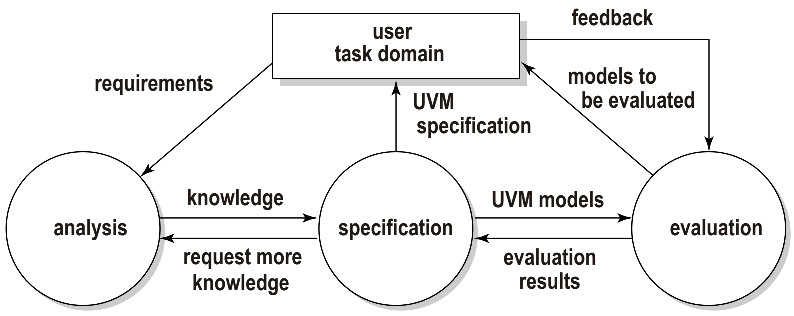

Figure 1: A process model for user interface design