Mental Models and Human-Computer Interaction 2
M.J. Tauber and D. Ackermann

THE IMPORTANCE OF MODELS IN MAKING COMPLEX SYSTEMS COMPREHENSIBLE

Gerhard Fischer

Department of Computer Science and Institute of Cognitive Science,
University of Colorado, Boulder, USA

ABSTRACT

Reality is not user-friendly. To cope with, model, and comprehend a complex reality requires complex systems. Complex systems offer power, but they are not without problems. In our research, high functionality computer systems serve as prototypical examples for complex systems and are used to instantiate our framework of cooperative problem solving in joint human-computer systems.

Models play a crucial role in the creation and use of these systems and help to increase their comprehensibility. Three different types of models are discussed in this paper: the designers' models of users, the users' models of systems, and the systems' models of users.

Innovative system designs supporting human problem-domain communication and providing design environments illustrate the relevance of these models in making complex systems comprehensible.

Introduction

We are interested in the design and understanding of *high-functionality computer systems*. These systems contain thousands of objects, and they are not completely mastered even by expert users. An important question to ask is: What kind of models are needed for the design, comprehension, and use of such systems?

High-functionality computer systems are *designed* systems. To comprehend designed systems, we have to understand the goals, functions, and adaptive capabilities for which they are used. The models associated with these systems are part of the design (i.e., they must be designed too), and they can and should provide important requirements for the design.

Many high functionality computer systems (e.g., powerful programming environments, knowledge-based systems) are used to support *cooperative problem solving* in joint human-computer systems. Models are of greater importance in cooperative problem solving systems than in autonomous systems because the problem solving activity is shared by the cooperating agents. In order to increase the comprehensibility of high-functionality computer systems, we are developing methodologies and systems which break the *conservation law of complexity* (Simon, 1981). This law states that the description of a complex system does not need to be equally complex.

This paper discusses and describes three types of models (extending a classification scheme presented in Fischer (1984)):

- M_1 : the designers' models (the models that designers have of users, tasks, and technologies relevant to the design of a system),
- M_2 : the users' models (the models that users have of systems and tasks), and
- M_3 : the systems' models (the models that systems have of users and tasks).

The relevance of these models is demonstrated in the context of a number of system building efforts oriented towards making complex systems comprehensible.

High-functionality computer systems - examples of complex systems

Our main research interest is how people understand and successfully use high-functionality computer systems that contain substrates of components used in the design of artifacts in a variety of domains. Symbolics Lisp machines and UNIX systems are examples of high-functionality computer systems. Symbolics Lisp machines, for example, offer approximately 30,000 functions and 3,000 flavors (object-oriented classes) documented on 4,500 pages of manuals. They contain software objects which form substrates for many kinds of tasks. For example, the user interface substrate provides classes of windows, support for inter-referential input/output, and screen layout design tools. Systems with such a rich functionality offer power as well as problems for designers and users. Even experts are unable to master all facilities of high-

functionality computer systems (Draper, 1984). Designers using these systems can no longer be experts with respect to all existing tools, especially in a dynamic environment where new tools are being continuously added. High-functionality computer systems create a *tool-mastery* burden (Brooks, 1987) that can outweigh the advantage of the broad functionality offered.

The study and the understanding of complex objects and systems has been the major research goal for the *Sciences of the Artificial* (Simon, 1981). Dawkins (1987) claims that one major area for studying complex systems is biology, because biological objects give the appearance of having been designed for a purpose. This is in contrast to physics which he sees as the study of simple things that do not tempt us to invoke design. From a design point of view, human-made artifacts (computers, airplanes, etc.) should be treated as biological objects despite the fact that they are not alive.

Complex objects and systems in Simon's, Dawkin's, and our own thinking can be characterized by three major properties:

(1) They have a heterogeneous structure,

(2) their constituent parts are arranged in a way that it is unlikely to have arisen from chance alone, and

(3) they have some quality which was specified in advance.

There is a crucial difference between models for the *Natural Sciences* (dealing with natural systems) and models for the *Sciences of the Artificial* (dealing with designed systems). In the *Natural Sciences*, models have to describe existing phenomena. We cannot change physics to make the construction of models easier for users. Models have to be based on reality. However, simplification may play a crucial role in achieving a high degree of *cognitive comprehensibility* (e.g., worlds without air resistance and without friction are widely used simplifications).

In the *Sciences of the Artificial*, models are an integral part of the design itself, and they can and should serve as important design criteria for systems. Previous research about models has been oriented mostly towards rather simple systems i.e., systems requiring operation only or systems which could be learned in a few hours or less (Kieras and Bovair, 1984). Norman (1986) claims that constructing models is easy for single tasks, used by one set of users (e.g., specialized tools such as

spreadsheets) whereas constructing models is difficult (perhaps impossible) for general purpose systems with an open-ended set of users and power. High-functionality computer systems pose a dilemma: On the one hand, these are systems where good models are most urgently needed - but on the other hand, it is unclear how these systems can be designed so users may successfully build models for them. Models for high-functionality computer systems cannot be deduced merely from experience because there are too many experiences to go through and they cannot be complete. Therefore future research efforts should be oriented not towards perfect models, but towards models which are "good enough". Learning complex systems is an incremental, indefinite process requiring an understanding of how models evolve in naturalistic settings over long periods of time.

To develop better design requirements for complex systems, we have studied *usage patterns of complex systems* as shown in figure 1. This qualitative analysis of users' knowledge about complex systems reveals two interesting findings:

- The users' model of the system contains concepts which do not belong to the system (the part of D_3 which is not part of D_4).
- There are system parts of which users are unaware (the part of D_4 which is not part of D_3).

The former issue requires facilities assisting users in incrementally bringing their $\mathbf{M}_2$-type models closer in accordance with the actual system. To address the latter issue, intelligent support systems (e.g., active help systems (Fischer, Lemke and Schwab, 1984) and critics (Fischer, 1987)) are needed which rely on $\mathbf{M}_3$-type models pointing out to users existing functionality that may be useful for their tasks.

Cooperative problem solving systems

The power of many high-functionality computer systems can best be exploited by users if the systems are designed to facilitate cooperation between a human and a computer. Cooperation requires more from the system than having a nice user interface or supporting natural language dialogs. One needs a richer theory of problem solving, one

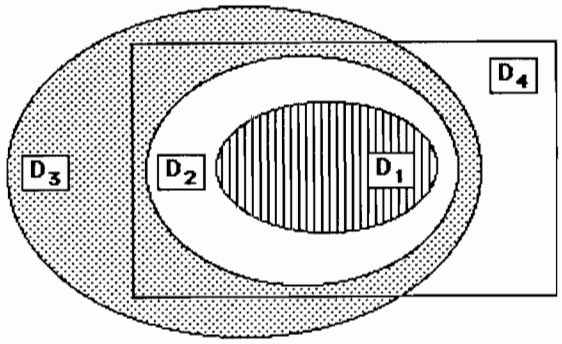


Figure 1. Levels of system usage.
The different domains correspond to the following:

- D_1: the subset of concepts (and their associated commands) that users know and use without any problems.
- D_2: the subset of concepts that users use only occasionally. They do not know details about the concepts contained in D_2 and they are not too sure about their effects.
- D_3: the users' model of the system (i.e., $\mathbf{M}_2$, the set of concepts which they think exist in the system).
- D_4: the actual system.

that analyzes the functions of shared representations (i.e., models of the communication partner and models of the task), mixed-initiative dialogs, argumentation, and management of trouble.

In a cooperative problem solving system the users and the system share responsibility for the problem solving and decision making. Different role distributions may be chosen depending on the users' knowledge, the users' goals, and the task domain. A cooperative system requires richer communication facilities than the ones that were offered by traditional expert systems. Figure 2 shows the architecture for such a system.

Cooperative problem solving requires that users have $\mathbf{M}_2$-type models of the systems with which they interact (see section "The users' model of systems") and that systems have $\mathbf{M}_3$-type models of their users (see section "The systems' models of users"). Being unable to completely understand the actual system (see figure 1), users have to interact with a system based on their $\mathbf{M}_2$-type models. $\mathbf{M}_3$-type models are needed to provide explanations, e.g., by using *differential*

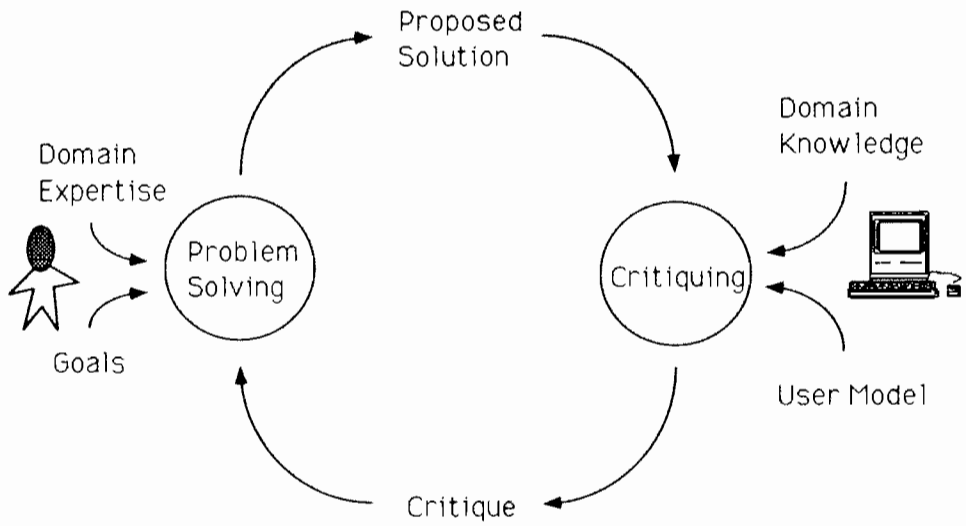
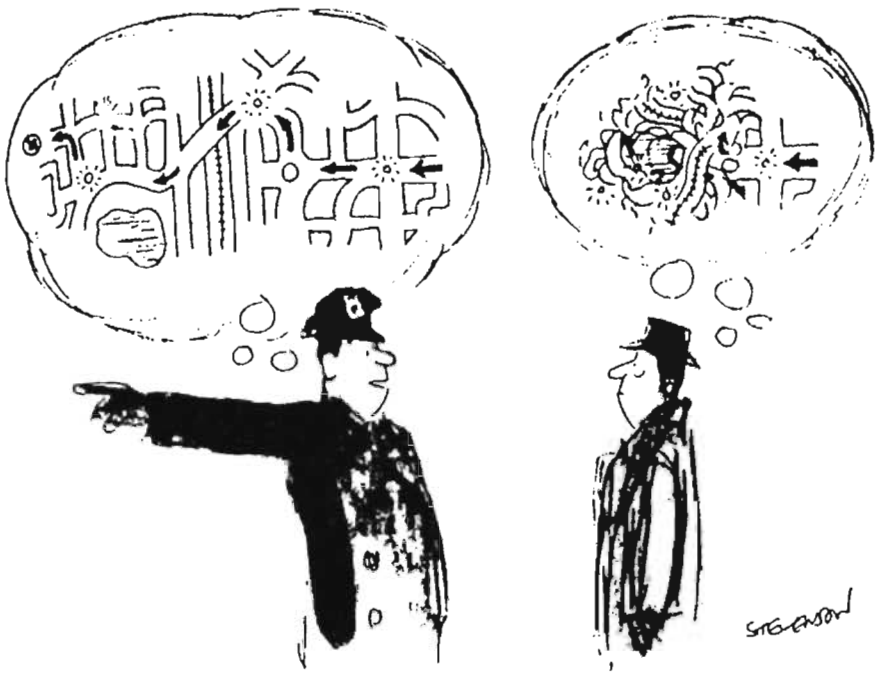


Figure 2. Architecture for a cooperative problem solving system.
This diagram shows an architecture for cooperative problem solving. It is significant that the human user and the computer each possess domain knowledge that is brought to bear on the problem.

descriptions. Explanations need to be given at the right level of detail and with the right level of assumed shared understanding about the other agent's knowledge. Differential descriptions are used to describe something new using terms that an individual user already understands. Assuming too much knowledge on the listener's part leads to information overflow problems as illustrated in figure 3.

Cooperative problem solving systems cannot be restricted to "one-shot" affairs. One cannot always be right the first time, and one cannot guarantee that advice or criticism is understood. In order to obtain a deeper understanding of these issues, we conducted an empirical study of cooperative problem solving between customers and sales agents (including their use of models) in a very large hardware store (offering more than 300,000 items). Details of this study are contained in Reeves (1989). The most important findings are summarized here briefly:

- *Incremental query specification*: Frequently customers did not know what they really needed, and did not know how their problems



Drawing by Stevenson; © 1976 The New Yorker Magazine

Figure 3. Mental models and mutual understanding.

The picture characterizes a number of issues. The visitor cannot build up a coherent model. More of the information provided should have been put in the world (e.g., by drawing a map). The structural model provided by the policeman is too detailed; it may be possible to avoid this by tailoring the explanations more to the goals and objectives of the visitor. The policeman could have reduced the complexity of his description using layers of abstractions.

could be mapped onto the items which the store offers. Their queries were constructed incrementally through a cooperative problem solving process between customers and sales agents.

- *From natural language to natural communication*: People rarely spoke in complete, grammatical sentences, yet managed to communicate in a natural way. This observation indicates that the support of natural communication (which allows for breakdowns, follow-up questions, clarifying dialogs, explanations, etc.) is much more important than being able to parse complex syntactic sentences.

- *Mixed-initiative dialogs*: People are flexible in the roles they play during a problem solving episode. They easily switch from asking to explaining, from learning to teaching. The structure of these dialogs were neither determined by the customer nor by the sales agent, but indicated mixed initiatives (Carbonell, 1970) determined by the specifics of the joint problem solving effort.
- *Multiple specification techniques*: A variety of different specification techniques were observed ranging from bringing in a broken part to very general request such as "I need a lock for my doors that will reduce my insurance rate."
- *Management of trouble*: Many breakdowns and misunderstandings occurred during the observed problem solving episodes. But in almost all cases, clarifying dialogs led to a recovery, illustrating the feature that problem solving among humans cannot be characterized by the absence of trouble, but by the identification and repair of breakdowns (Suchman, 1987).
- *User Modeling*: The study made it evident that user modeling plays a crucial role in identifying the right level of shared understanding and providing the appropriate level of explanation and help.

A conclusion from this study was that without similar support structures as observed in the interaction between the sales agents and the customers, high-functionality computer systems will be underutilized and are more of an obstacle than a source of power in solving problems.

Comprehensible systems

Informational systems pose specific problems with respect to comprehensibility. Comprehensibility cannot be defined abstractly without reference to users' knowledge and goals. In this section, a number of aspects are described that make systems difficult to understand, requirements are enumerated that potentially increase comprehensibility, and a theoretical framework is presented that contributes to the characterization of comprehensibility. Comprehension in the context of high-functionality computer systems is relevant because these systems change the nature of many tasks from problem solving to comprehension. Instead of constructing a solution to a problem from scratch, many

problems can be solved by a selection process followed by a customization process.

Difficulties in comprehending complex systems

Unlike other engineered systems (e.g., mechanical systems), computer systems are largely *opaque*, that is, their function cannot be perceived from their structure (Brown, 1986). Beyond being opaque, the *sheer size* prohibits a complete understanding. Many of the situations that a user encounters in a high-functionality computer system are new and unfamiliar. The *absence of a user-perceivable structure* (such as a layered architecture or an "increasingly complex microworld" structure (Burton, Brown and Fischer, 1984)) often makes it impossible for users to acquaint themselves with closed subparts of the system.

The *vocabulary problem* (Furnas et al., 1987) limits comprehension, because people use a great variety of words in referring to the same thing. Empirical data show that no single access word, however carefully chosen, will cover more than a small portion of users' attempts. Even more serious than the vocabulary problem is the fact that people do not only choose different names for the same underlying concepts, but they *decompose a complex system into different sets of concepts* (Standish, 1984). Especially in new domains (e.g., modern user interfaces, object-oriented programming, mental models research), where the understanding of the field is in an early stage, different conceptualizations compete with each other and limit the comprehensibility.

System designers acknowledge the difficulty in comprehending complex computational systems by developing associated help systems. Many of those are limited in their usefulness because they are oriented toward the system rather than toward the user. That is, information is structured around a system description, not around an analysis of the problems users address when using the system. A shortcoming of many existing information stores is that access is by *implementation unit* (e.g., LISP function, UNIX command) rather than by *application goal* on the task level.

A last issue to be mentioned in this short and incomplete enumeration of factors that limit the comprehensibility of complex systems is the poor understanding of *consistency*. Grudin (1989) has shown that the role of consistency is much more problematic than most people assume and that predictability may be more important than consistency.

He identified a number of situations in which violations of principles caused no problems in practice, and he pointed out that there is no unique best strategy to achieve consistency. The dimension along which a system should behave consistently is dependent on the goals one wants to achieve.

Techniques for making systems comprehensible

There is no "silver bullet" (Brooks, 1987) for making systems comprehensible, but there are a number of techniques that can contribute to a higher degree of comprehension of a complex system. We briefly describe the ones that we have found most relevant in our own research.

Cognitive fidelity versus physical fidelity

Brown (Brown, 1986) argues for the separation of *physical fidelity* from *cognitive fidelity*, in order to recognize that an "accurate rendition of the system's inner workings does not necessarily provide the best resource for constructing a clear mental picture of its central abstractions." In the Natural Sciences, designers and teachers can only change the cognitive fidelity of a system. That is, incorrect models based on intentional simplifications may be more useful than technically correct ones. Simon (Simon, 1981) argues that in many situations, it is not the best or the correct model which is the desirable one, but the one which is understood by everyone and supports rather than paralyzes action. In the Sciences of the Artificial, designers have the freedom to shape the *physical fidelity* (as a product of our design), allowing users to construct efficient M_2-type models.

Human problem-domain communication and construction kits

Many systems use knowledge representations at a too low level of abstraction. This makes both system design and explanation difficult, since system designers must transform the problem into a low-level implementation language, and explanation requires translating back to the problem level. When creating powerful tools for domain experts, we must teach the computer the languages of application domains. Systems with abstract operations and objects of a domain built into them give the impression of *human problem-domain communication* (Fischer and Lemke, 1988) rather than human-computer communication. Human

problem-domain communication reduces the cognitive transformation distance between problem-oriented and system-oriented descriptions (Hutchins, Hollan and Norman, 1986).

Construction kits are tools that foster human problem-domain communication by providing a set of building blocks that model a problem domain. The building blocks and the operations associated with them define a design space (the set of all possible designs that can be created by combining these blocks) and a design vocabulary. A construction kit makes the elements of a domain-oriented substrate readily available by displaying them in a menu or graphical palette (see figure 13). This kind of system eliminates the need for prerequisite, low-level skills such as knowing the names of software components and the formal syntax for combining them.

Human problem-domain communication redraws the borderline between the amount of knowledge coming from computer systems versus coming from the application domain. Systems supporting human problem-domain communication eliminate some of the opaqueness of computational systems by restricting them to specific application domains facilitating the model building and understanding process for all the models discussed in this paper.

Layered architectures

Layered architectures can enhance comprehensibility by hiding the lower levels of a complex system from the user. Dawkins (1987) describes this idea in connection with biological objects:

> I am a biologist. I take the facts of physics, the facts of the world of simplicity, for granted. If physicists still don't agree over whether those simple facts are yet understood, that is not my problem. My task is to explain elephants, and the world of complex things, in terms of the simple things that physicists either understand, or are working on. The physicist's problem is the problem of ultimate origins and ultimate natural laws, the biologist's problem is the problem of complexity.

In our own research (Lemke, 1989), we have built software environments with layered architectures providing tools at multiple levels ranging from domain-independent to domain-specific (see figure 4). A hierarchy of increasingly powerful tools is based on low-level primitives as offered by standard programming languages. Domain-oriented substrates are built on top of these primitives. Construction kits make

the elements of substrates readily available. Design environments use knowledge of the problem domain to judge the quality of a design.

Knowledge-Based Design Environments (e.g., JANUS with its critics and the catalog of examples)
Construction Kits (e.g., the palette in JANUS)
High-Functionality Computer Systems with many Substrates (e.g., bitmap editors to generate icons)
Low-level Primitives (e.g., read-character, write-character)

Figure 4. From low-level primitives to knowledge-based design environments. This figure shows a progression of increasingly higher levels of support tools beginning with low-level primitives up to knowledge-based design environments.

Layered architecture plays an important role for explanations. If there is a complex thing that we do not yet understand, we can come to understand it in terms of simpler parts that we already do understand. But these simpler parts should be as close as possible to the object under investigation. Complex organization should be constructed such that satisfying explanations may normally be attained if we peel the hierarchy down one or two more layers from the starting layer. This observation provides evidence for the success of information processing psychology in explaining intelligent behavior (Newell and Simon, 1976).

Main streets versus side streets

An interesting analogy can be constructed by comparing the use of high-functionality computer systems with learning how to get around in a large city (Ehrlich and Walker, 1987) (similar approaches are "training wheels" (Carroll and Carrithers, 1984) and "increasing complex microworlds" (Burton, Brown and Fischer, 1984)). Getting around in the city as well as doing something in a high-functionality computer system provides options: users can do things in more than one way, and they can express their tasks in the manner most comfortable to them. Main streets may not provide the shortest distance, but they require a smaller mental effort and they reduce the probability of getting lost (see the illustration in figure 3).

"How-to-do-it knowledge" versus "how-it-works knowledge"

The relevance of these two types of knowledge is determined by the users' goals and other contextual factors (e.g., if a device breaks, will there be someone around to help out?). Many aspects of the internal mechanism of a piece of equipment are irrelevant to most user tasks. What happens if a breakdown occurs? If our goals include the ability to fix something, then "how-it-works knowledge" is definitely important. It remains an open issue how many levels one has to descend in a layered architecture (see figure 4) to comprehend the "how-it-works knowledge" for a system. For a more detailed discussion see the contributions by Polson and Kieras (Turner, 1988).

"Knowledge in the world" versus "knowledge in the head"

Norman (1988) distinguishes between "knowledge in the world" and "knowledge in the head". One of the challenges for innovative design of computer systems is redrawing the borderline between the two. Techniques should be developed which put more "knowledge in the world" so we have to keep less "knowledge in our head". As mentioned before, informational systems are often opaque. But computer systems offer substantial power for making the *invisible visible.* In doing so, the real challenge is not making everything visible, but to make the information visible which is of importance for understanding a system and for the goals of a specific user (the latter requiring that the system has a model of a user; see section "The systems' models of users"). Visualization techniques (such as our "software oscilloscope" (Böcker, Fischer and Nieper, 1986)) provide automatically generated graphical representations which assist in understanding data and control structures by generating more "information in the world".

Applicability conditions

Many complex systems are difficult to understand, because they represent building blocks, tools, and examples without applicability conditions describing the situations in which these objects can or should be used. The objects in the **Palette** or the designs in the **Catalog** (see figure 13) have no knowledge associated with them that would inform users about situations in which they can be used.

Differential descriptions

An economic description for many objects is an explanation of how they differ from some already known object. Object-oriented inheritance networks and case-based reasoning methods achieve economy in their representation by exploiting this principle. This approach requires systems having models of users and users having models of systems. Descriptions of concepts new to a particular user are generated by relating them to concepts already known to a user. These previously understood concepts are contained in the system's model of the user.

Learning on demand

The fact that high functionality computer systems are never completely mastered implies that support for learning on demand is not a luxury but a necessity. The support of learning on demand sets computer-based systems apart from other media such as paper. Paper is passive and can only serve as a repository for information, whereas computer systems can be active and assist us in searching, understanding, and creating knowledge in the course of cooperative problem solving processes. Users are often unwilling to learn more about a system or a tool than is necessary for the immediate solution of their current problem (i.e., they remain in D_1 in figure 1). To be able to successfully cope with new problems as they arise, learning on demand is necessary. It provides new information in a relevant context and it eliminates the burden of learning things in neutral settings when users do not know whether the information will ever be used and when it is difficult for them to imagine an application.

Learning on demand can be differentiated according to whether the user or the system initiates the demand. Demands originating from the user can be triggered either by a discrepancy between an intended product and the actual product produced or by experimentation with a system turning up interesting phenomena that users find worth further exploration. Demands to learn cannot originate from users if users are unaware that additional functionality exists in the system (e.g., the functionality contained in D_4 but not in D_3 in figure 1). In this situation, the system must take the initiative. To prevent active systems from becom-

ing too intrusive, a metric is required for judging the adequacy of a user's action, and the advice must be based on information structures accumulated in the system's model of the user (Fischer, Lemke and Schwab, 1984). *Breaking the "Conservation Law of Complexity"*

Complex systems do not necessarily require equally complex descriptions. There is no "conservation law of complexity" (Simon, 1981), and the overall goal of making systems more comprehensible is the development of representations, techniques, and tools which allow us to break this conservation law. The following list summarizes some of the major approaches achieving this goal:

- exploit what people already know (e.g., support "human problem-domain communication"),
- use familiar representations, based on previous knowledge and analogous to known situations (e.g., take advantage of differential descriptions),
- exploit the strengths of human information processing (e.g., the power of our visual system) by putting more information into the world using technology which makes visible what would otherwise be invisible,
- segment information into microworlds (e.g., allow beginners to stay on "main streets"),
- enhance learning by supporting learning on demand.

Narrowing the gap between situation models and system model

Many systems fail because their designs and descriptions do not relate to users' problems. The discrepancy between a user's and a system's view of the world and approaches to overcoming this gap are briefly described in this section.

The *situation model* (Dijk and Kintsch, 1983; Fischer, Kintsch et al., 1989) is a mental representation of the situation as the user sees it, including the problems motivating a task, general ideas for finding a solution, and a characterization of the desired goal state. The *system model* consists of a set of operations that, when invoked, would result in the desired solution. These operations must all be within the repertory of the system; that is, for each operation there must exist one or more commands, depending upon context, which executes. At the level of the situation model, goals refer to actions and states in the users'

problem space and are articulated in terms of what they want. Goals may be precise or imprecise, but the important point is that they are not necessarily structured or named according to the system. They are subjective and vary among individuals.

Figure 5 outlines several different approaches to bridging the gap between the situation and system model. Which of the five approaches illustrated in this diagram is the most promising one depends on the specific situation and the ability of designers to construct the system components that make a specific approach possible.

Models

Models are ubiquitous. In this section, we will take a closer look at the role of the three models $\mathbf{M_1}$, $\mathbf{M_2}$, and $\mathbf{M_3}$ (as introduced in the first section).

$\mathbf{M_1}$: The designers' models of users

Design for users

The times when designers of computer systems were their only users belong to the past. "Arm-chair" design, in which designers choose themselves as models for the user population of their systems must be replaced by models of the intended user community of the system. Figure 6 illustrates this situation. These models can be formed by studying task domains, (e.g., office work (Malone, 1983), cooperative problem solving (Reeves, 1989)), interviewing users (Lewis, 1982), and evaluating users working with similar systems. It remains to be seen for specific designs, which parts of the designers' models are descriptive (based on the mentioned information sources) and which parts are prescriptive (based on their ideas of how the system should operate).

Design with users

The Scandinavian approach to system design (Bodker et al., 1988) is based on an even more cooperative approach between designers and prospective users. Instead of "design for users", one should "design *with* users" as illustrated in figure 7. Design with users supports cooperative design as a process of mutual learning; it allows users to participate in the design process as lay designers. Users probably have no particular

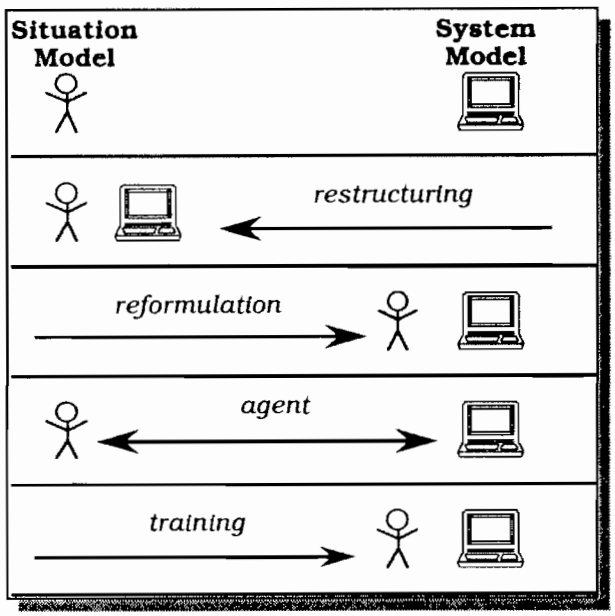


Figure 5. Different approaches in relating situation and system models.

- The first row illustrates the normal situation, where there is no support for bridging the gap. In this case, people frequently have difficulties solving problems or finding information, because they are unable to generate an effective system model, even if they have a clear understanding of the situation involved.
- The second row shows the approach in which a new system model is constructed that is closer to an individual's situation model and hence easier to understand.
- The third row illustrates the possibility of making the system model more transparent, allowing users to express their situation model incrementally within the system model (an approach supported by HELGON, an information manipulation system based on query by reformulation (Fischer and Nieper-Lemke, 1989)). HELGON supports multiple specification techniques allowing users to access complex information spaces from different starting points.
- The fourth row shows how an agent can help in translating a query from the situation into the system model (e.g., a role played by sales agents, travel agents; see the description of the hardware store study in section "Cooperative problem solving systems").
- The last row illustrates the training approach, where users are trained to express themselves in the system model.

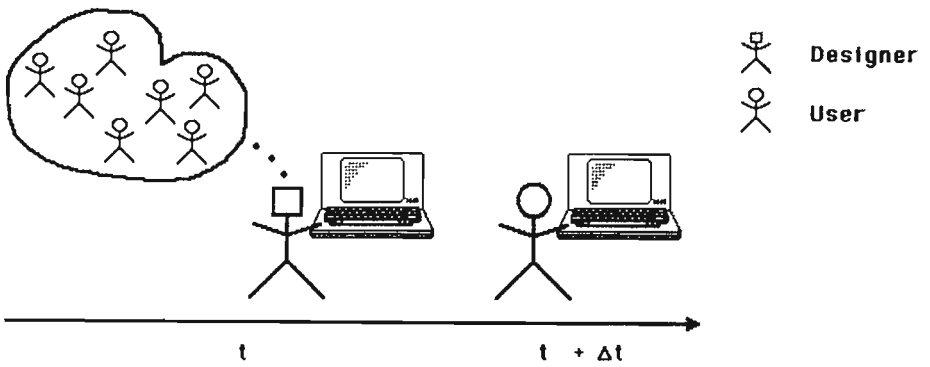


Figure 6. Design for users.
The designer uses models of users and tasks in constructing systems for an anticipated user community.

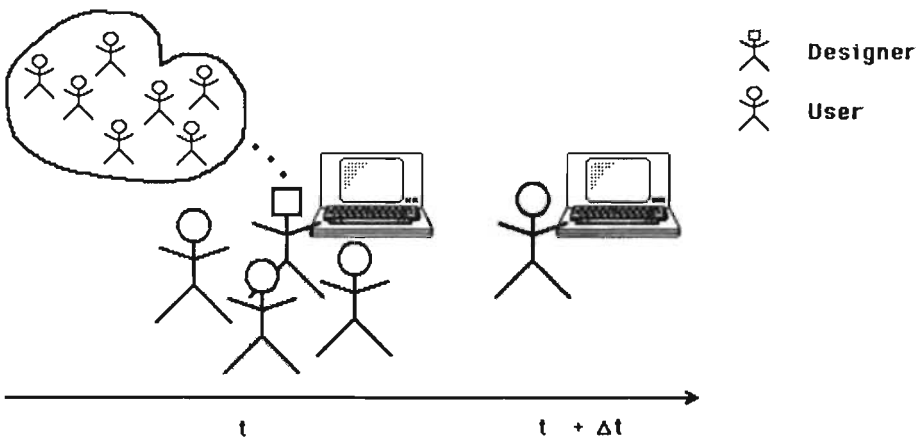


Figure 7. Design with users.
In "design with users", designers do not only use models of the potential user community, but collaborate with users in the design.

expertise as designers, but they have expertise within the work domain. This cooperation can potentially eliminate one of the major problems of software development: the thin spread of application knowledge (Curtis, Krasner and Iscoe, 1988).

Design with users requires having prototypes and supporting human problem-domain communication. Through the utilization of

prototypes users experience the future operation of a system within a simulated application. The needs and demands of the prospective users are essential to good design, but are hard to express (even by the users) before the future situation has been experienced. With prototypes, designers can share early design ideas with the target users. Prototyping is most beneficial if the design model used is articulated at the problem-domain level. A description at this level will be comprehensible to users familiar only with the problem domain.

Design for redesign

Pre-designed systems are too encapsulated for problems whose nature and specifications change and evolve. A useful system must accommodate changing needs. Domain experts must have some control over the system because they understand the semantics of the problem domain best. End-user modifiability (Fischer and Girgensohn, 1990) is not a luxury but a necessity in cases where the systems do not fit a particular task, a particular style of working, or a personal sense of aesthetics. Lack of modifiability creates systems whose *software is not soft*, prohibits evolution of systems over time, and makes users dependent on specialists for help.

M_2: The users' models of systems

A user's model of a complex system is a cognitive construct that describes a user's understanding of a particular content domain in the world. These models are formed by experience, self-exploration, training, instruction, observation, and accidental encounters. In systems that operate at the "human computer-communication" level, the model will be centered around the properties of a computer system (see figure 8). An advantage of this type of model (representing a general computational environment) is that it is uniform across domains.

In systems that operate at the "human problem-domain communication" level (such as the JANUS system; see section "Innovative system design"), users are able to form models using concepts much more closely related to an application domain (see figure 9). One basic working assumption of our research is that it is easier for users to develop a model at this level because it can be articulated in their domain of expertise. Another advantage of these types of models is that they reduce the gap between system and task domain.

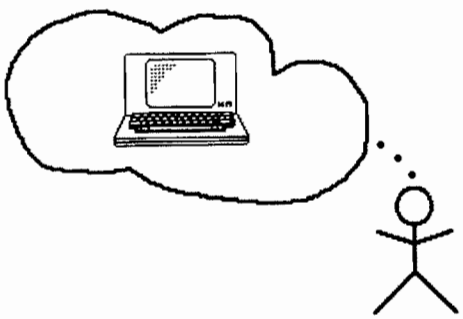


Figure 8. A user's model of a system at the "human-computer communication" level.

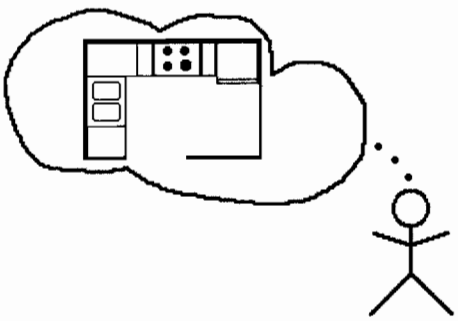


Figure 9. A user's model of system at the "human problem-domain communication" level.

Criteria for evaluating the usefulness of users' models of systems are (for a more complete list see Turner (1988)):

- *familiarity* (do they draw upon familiar domains?),
- *scope* (how much of the total system does the model cover?),
- *derivational* length (how long is the derivational length associated with any particular explanation),
- *accuracy and consistency* (see previous remarks in section "Comprehensive systems"),

- *extensibility, generalizability, task-orientation,*
- *predictive and explanatory power.*

Systems that operate at the "human problem-domain communication" level have many advantages with respect to these evaluation criteria, e.g., they achieve short derivational paths by taking advantage of higher level of abstraction.

Two other aspects of complex systems are important with respect to the users' models: *evolution of a system* and *tailorability*. Users' knowledge of systems and their perspective of what they can and want to do with systems changes over time. What happens to their models of the system in such a process? Individuals change their understanding of a system as they continue working with it (see figure 10).

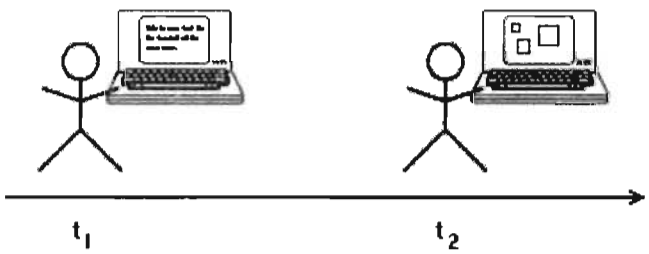


Figure 10. Evolution of a system over time for the same user.

Evolution can occur in *adaptable systems* (the users are changing the system) as well as in *adaptive systems* (where the system changes its appearance by itself).

Users want to tailor a system to their specific needs (see figure 11). Sometimes the need for a different model of the system may be the driving force behind these changes.

M_3: The systems' models of users

There are a number of efforts to incorporate models of users into knowledge-based systems (Rich, 1983; Clancey, 1986; Kass and Finin, 1987; Fain-Lehman and Carbonell, 1987; Reiser, Anderson and Farrell, 1985; Wahlster and Kobsa, 1988; VanLehn, 1988). In our own

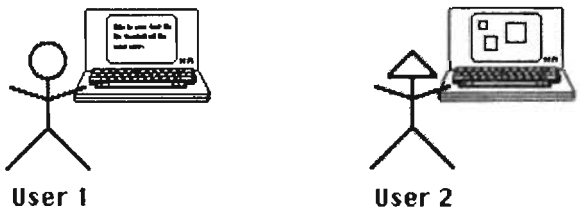


Figure 11. Tailorability of a system to the needs of different users.

research, we have investigated systems' models of users (see figure 12) in connection with *active help systems* (Fischer, Lemke and Schwab, 1984) and *critics* (Fischer, 1987; Fischer and Mastaglio, 1989).

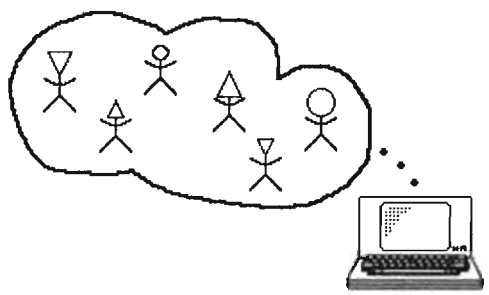


Figure 12. A system's model of users.

M_3-type models for critic systems pose specific demands. Unlike tutorial systems, which can track a user's expertise over a path of instruction, computer-based critics must work with users having a variety of background experiences. To operate effectively, critics must acquire an individual, persistent model of each user.

There are a number of different goals for which systems' models of users are important: customizing *explanations* so they cover exactly what users need to know; providing *differential descriptions* of new concepts in relationship to known concepts as indicated by an M_3 model of an individual user; presenting *information through user-specific filters* focusing on the parts which seem to be most relevant for a user, and keeping *active systems quiet most of the time*.

A major problem in putting M_3-type models to work is the knowledge acquisition process for these models. In our work, we have used four techniques in acquiring knowledge about users: explicit questions, testing, tracking tutorial episodes, and implicit approaches. In implicit approaches, the system observes users and makes inferences regarding their expertise. The methods incorporated into our acquisition methodology are the result of an empirical study of how human experts accomplish the same task (Fischer, Mastaglio and Rieman, 1989).

Making M_3-type models a reality for widely used systems is a challenging research goal. Having persistent models that undergo evolutionary changes and that can deal with conflicting information (i.e., new information inferred about a user contradicts information contained in the model of the user) are research issues that need further explanation. Another challenge is making M_3-type models inspectable and modifiable by users.

Innovative system design efforts in making complex systems comprehensible

Design environments give support that is not offered by simple construction kits. In addition to presenting the designer with the available parts and the operations for putting them together, they

- incorporate knowledge about which components fit together and how they do so,
- contain cooperative critics that recognize suboptimal design choices and inefficient or useless structures,
- allow multiple specification techniques in creating a program and understanding a system,
- link internal objects with their external behavior and appearance,
- provide animated examples and guided tours (techniques supporting the incremental development of models in a high-functionality computer system), and
- support end-user modifiability.

JANUS: An example of an integrated, knowledge-based design environment

JANUS (Fischer, McCall and Morch, 1989) allows designers to construct artifacts in the domain of architectural design and at the same time informs them about principles of design and their underlying rationale by integrating two design activities: construction and argumentation. *Construction* is supported by a knowledge-based graphical design environment (see figure 13), and *argumentation* is supported by a hypertext system (see figure 14).

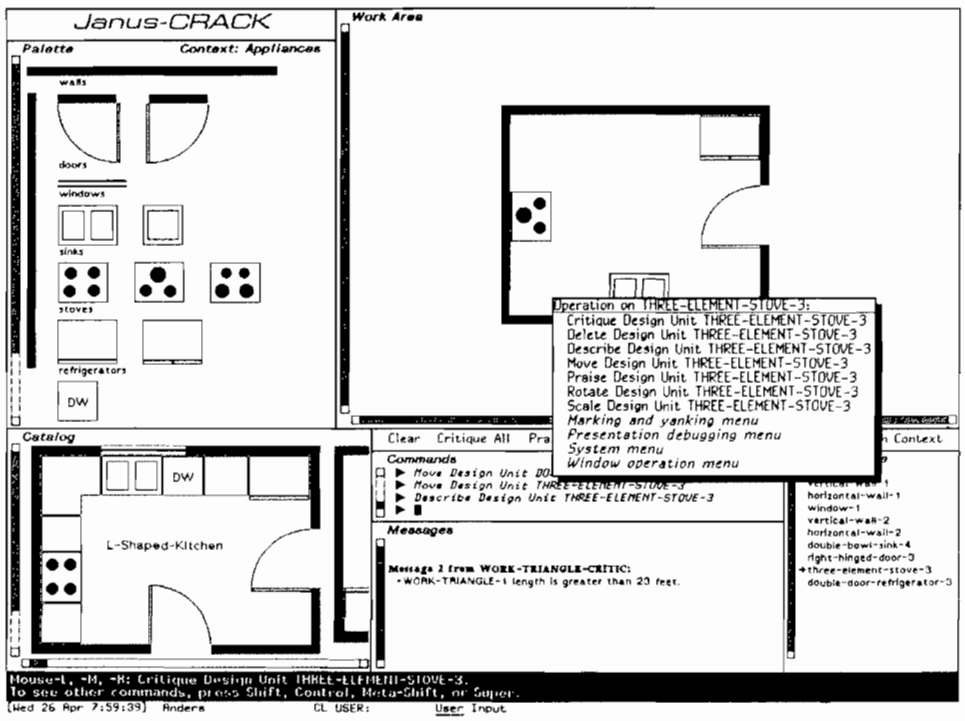


Figure 13. JANUS construction interface.

The interface of JANUS' construction component is based on the world model. Design units are selected from the **Palette**, and moved into the **Work Area**. Operations on design units are available through menus. The screen image shown displays a message from the **Work-Triangle-Critic**.

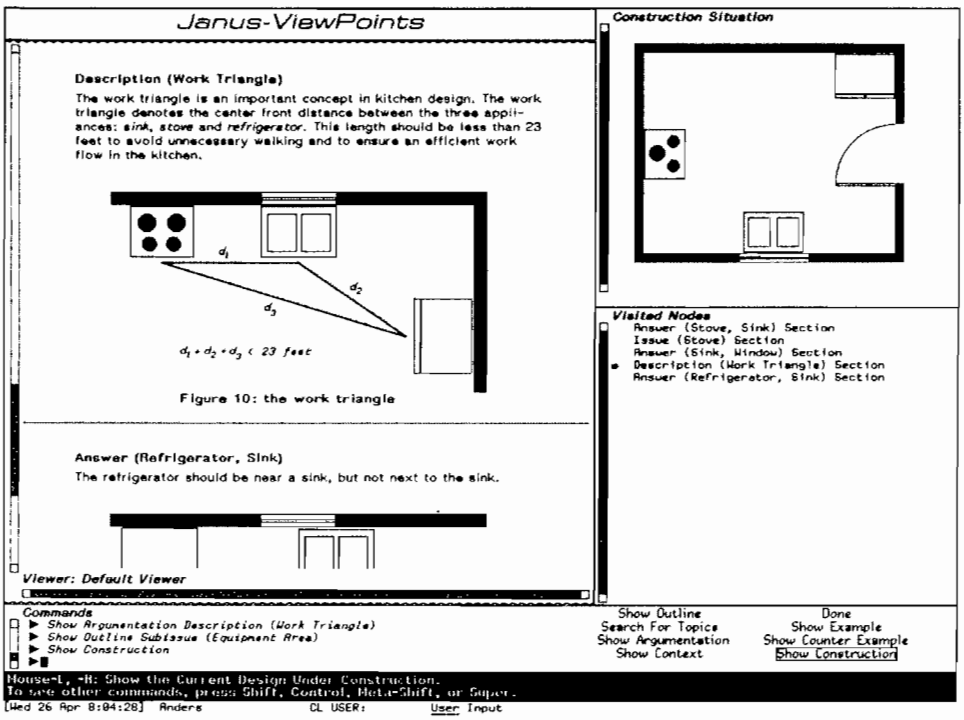


Figure 14. JANUS argumentation interface

JANUS' argumentation component uses the Symbolics Document Examiner as a delivery interface. The construction situation can be displayed in one of the panes to allow users to inspect the constructive and argumentative context simultaneously.

JANUS provides a set of domain-specific building blocks and has knowledge about combining them into useful designs. With this knowledge it "looks over the shoulder" of users carrying out a specific design. If it discovers a shortcoming in the users' designs, it provides a critique, suggestions, and explanations, and assists users in improving their designs. JANUS is not an expert system that dominates the process by generating new designs from high-level goals or resolving design conflicts automatically. Users control the behavior of the system at all times (e.g., the critiquing can be "turned on and off"), and if users disagree with JANUS, they can modify its knowledge base.

Critics in JANUS are procedures for detecting non-satisficing partial designs. The knowledge-based critiquing mechanism in JANUS bridges the gap between construction and argumentation. This means that critiquing and argumentation can be coupled by using JANUS' critics to provide the designer with immediate entry into the place in a hypermedia network containing the argumentation relevant to the current construction task. Such a combined system provides argumentative information for construction effectively and efficiently. Designers are not forced to realize beforehand that information will be required, anticipate what information is in the system, or know how to retrieve it.

JANUS' construction component

The constructive part of JANUS supports building an artifact either from scratch or by modifying an existing design. To construct from scratch, the designer chooses building blocks from the design units **Palette** and positions them in the **Work-Area** (see figure 13).

In construction by modification of an existing design, the designer uses the **Catalog** (lower left in figure 13), which contains several example designs. The designer can browse through this catalog of examples until an interesting one is found. This design can then be selected and brought into the **Work-Area**, where it can be modified. The **Catalog** contains both good designs and poor designs. The former satisfy all the rules of kitchen design and will not generate a critique. People, who want to design without having to bother with knowing the underlying principles, might want to select one of these, since minor modifications of them will probably result in few or no suggestions from the critics. The *poor designs* in the **Catalog** support learning the design principles. By bringing these into the **Work-Area**, users can subject them to critiquing and thereby illustrate those principles of kitchen design that are known to the system.

The *good designs* in the **Catalog** can also be used to learn design principles and exploring their argumentative background. This is done by bringing them into the **Work-Area** then using the "Praise all" command. This command causes the system to generate positive feedback by displaying messages from all of the rules that the selected example satisfies. The messages also provide entry points into the hypertext argumentation.

JANUS' argumentation component

JANUS' knowledge-based critics serve as the mechanism for linking construction with argumentation. They "watch over the shoulders" of designers, displaying their critique in the **Messages** pane (center bottom in figure 13) when design principles are violated. In doing so they also identify the argumentative context which is appropriate to the current construction situation. For example, when a designer has designed the kitchen shown in figure 13, the **Work-Triangle-Critic** fires and detects that the work triangle is too large. To see the arguments surrounding this issue, the designer clicks the mouse on the text of this criticism with the mouse. The argumentative context shown in figure 14 is then displayed. The argumentation component is implemented as a hypermedia system (Fischer, McCall and Morch, 1989).

M_1-type models in JANUS

In the context of JANUS, the designer is creating the design environment, whereas the user is working with the design environment to design kitchens. Within the JANUS project, all methodologies relevant for M_1-type models (as described in section "The designers' models of users") were employed:

- *Design for users*: Domain-specific design knowledge represented in JANUS has been acquired from kitchen design books. The architecture for a knowledge-based design environment was developed from the evaluation of earlier systems (Fischer and Lemke, 1988), indicating that construction kits were insufficient.
- *Design with users*: In the first phase, we worked together with kitchen designers using protocol analysis and questionnaires to capture the professional knowledge. The designers were given typical scenarios that included a sample floor plan and a hypothetical client with certain needs and desires. They were asked to plan a kitchen for this client in the space provided. In order to capture all the steps involved, including the ones which designers normally do not communicate, they were asked to think aloud during the design process. Among many other things, the interviews showed that professional kitchen designers design at different levels of abstraction. To support this requirement, we introduced intermediate abstractions into JANUS called "work centers". For example, a sink and a dishwasher can be combined into a cleanup center. Designers start

designing a kitchen with work centers and replace them later with their components (as illustrated in figure 15).

- *Design for redesign*: Experimental use of JANUS by professional and amateur kitchen designers indicated that situations arise that require the modification of the design environment itself. To support end-user modifiability in JANUS (Fischer and Girgensohn, 1990), the system was extended with knowledge-based components to support the several types of modifications by the user (e.g., introducing new classes of objects into the palette (e.g., a "microwave"), adding new critic rules to the system, and supporting the creation of composite objects (e.g., a "cleanup center").

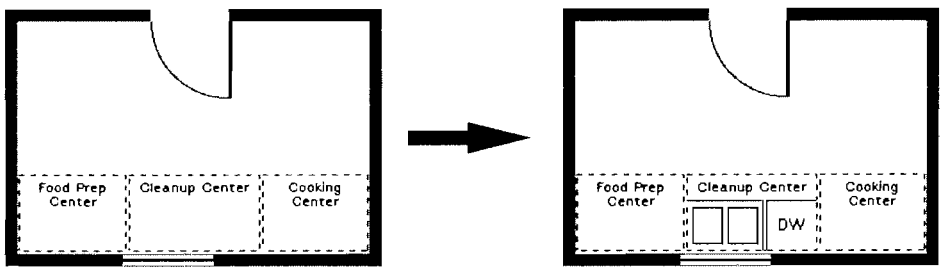


Figure 15. Work centers.

Composite objects allow designers to design at different levels of abstractions. After having completed the design of a kitchen at the work center level, users can proceed to the detailed design of the centers. The figure shows how the cleanup center is expanded to its components sink and dishwasher.

M_2-type models in JANUS

JANUS as a knowledge-based design environment supports human-problem communication and therefore the development of M_2-type models at the level of the problem domain (see figure 9) eliminating the need to understand the system at a programming language level (see figure 8). Designs can be represented at a level of abstraction with which the user is familiar. The development of an operational M_2-type model is enhanced further by:

- the critics giving users an opportunity for learning on demand and allowing "the situation to talk back" (Schoen, 1983), which supports the incremental construction of a model,
- the availability of the **Catalog** allowing users to get an overview of existing designs and giving them a feeling of what kinds of artifacts can be constructed,
- the existence of the argumentation component allowing users to inspect the design rationale behind the critics and making the underlying model open, discussable, and defendable,
- the support of multiple interaction techniques in creating a design, i.e., using construction, specification, argumentation, and the **Catalog** synergistically to create an understanding of the possibilities and the limitations of the system.

M_3-type models in JANUS

JANUS is a system supporting different classes of users ranging from neophyte kitchen designers to experts. JANUS also supports different kinds of tasks, e.g. learning tasks (such as improving a "bad" kitchen example from the **Catalog**) and design tasks (such as designing a "dream" kitchen given a set of constraints). M_3-type models are needed to support these different users and tasks. The information contained in these models is used for

- activating and deactivating sets of critics depending on the specific user and the specific task,
- supporting the level and detail provided by the argumentation,
- supporting stereotypes of users (e.g., users with a large family or users who only want to prepare TV-dinners), influencing the behavior of the system to present itself in a more cooperate fashion.

Conclusions

Reality is not user-friendly. To cope, model, and comprehend a complex reality requires complex systems. Complex systems offer power, but they are not without problems. Part of this power is the rich functionality which these systems offer in principle. But without innovative system designs (including methodologies and techniques to support all three types of models discussed in this paper), these systems

will not live up to their potential. Integrated knowledge-based design environments are promising systems which lead to the design of more comprehensible complex systems.

Acknowledgements:

The author would like to thank his colleagues and students at the University of Colorado, Boulder, especially Walter Kintsch, Clayton Lewis, Raymond McCall, Peter Polson, Andreas Lemke, Thomas Mastaglio, Anders Morch, Helga Nieper-Lemke, Brent Reeves, and Curt Stevens who have contributed to the ideas and systems described in this paper.

The research was partially supported by grant No. IRI-8722792 from the National Science Foundation, grant No. MDA903-86-C0143 from the Army Research Institute, and grants from the Intelligent Systems Group at NYNEX and from Software Research Associates (SRA), Tokyo.

References

Bodker, S., Knudsen, J.L., Kyng, M., Ehn, P. and Madsen, K.H. (1988). Computer support for cooperative design. *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'88),* ACM, New York, September 1988, pp. 377-394.

Böcker, H.-D., Fischer, G. and Nieper, H. (1986). The enhancement of understanding through visual representations. *Human Factors in Computing Systems. CHI'86 Conference Proceedings* (Boston), ACM, New York, April 1986, pp. 44-50.

Brooks Jr., F.P. (1987). No Silver Bullet. Essence and Accidents of Software Engineering. *IEEE Computer,* Vol. 20, No. 4, pp. 10-19.

Brown, J.S. (1986). From cognitive to social ergonomics and beyond. In D.A. Norman and S.W. Draper (Eds.), *User Centered System Design. New Perspectives on Human-Computer Interaction.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Burton, R.R., Brown, J.S. and Fischer, G. (1984). Analysis of skiing as a success model of instruction: manipulating the learning environment to enhance skill acquisition. In B. Rogoff and J. Lave (Eds.), *Everyday Cognition: Its Development in Social Context.* Cambridge, MA: Harvard University Press.

Carbonell, J.R. (1970). AI in CAI: an artificial-intelligence approach to computer-assisted instruction. *IEEE Transactions on Man-Machine Systems,* MMS-11, No. 4.

Carroll, J.M. and Carrithers, C. (1984). Training wheels in a user interface. *Communications of the ACM,* Vol. 27, No. 8, pp. 800-806.

Clancey, W.J. (1986). Qualitative student models. *Annual Review of Computing Science,* Vol. 1, pp. 381-450.

Curtis, B., Krasner, H. and Iscoe, N. (1988). A field study of the software design process for large systems. *Communications of ACM,* Vol. 31, No. 11, pp. 1268-1287.

Dawkins, R. (1987). *The Blind Watchmaker,* New York - London: W.W. Norton and Company.

van Dijk, T.A. and Kintsch, W. (1983). *Strategies of Discourse Comprehension.* New York: Academic Press.

Draper, S.W. (1984). The Nature of Expertise in UNIX. *Proceedings of INTERACT'84,* IFIP Conference on Human-Computer Interaction, Elsevier Science Publishers, Amsterdam, September 1984, pp. 182-186.

Ehrlich, K. and Walker, J.H. (1987). High functionality, information retrieval, and the document examiner. In G. Fischer and H. Nieper (Eds.), *Personalized Intelligent Information Systems. Workshop Report (Breckenridge, CO).* Institute of Cognitive Science, University of Colorado, Boulder, CO, Technical Report No. 87-9, 1987.

Fain-Lehman, J. and Carbonell, J.G. (1987). *Learning the User's Language: A Step Toward Automated Creation of User Models.* Technical Report, Carnegie-Mellon University, March 1987.

Fischer, G. (1984). Formen und Funktionen von Modellen in der Mensch-Computer Kommunikation. In H. Schauer and M.J. Tauber (Eds.), *Psychologie der Computerbenutzung.* Schriftenreihe der Österreichischen Computer Gesellschaft, Vol. 22. Wien - München: R. Oldenbourg Verlag.

Fischer, G. (1987). A critic for LISP. In J. McDermott (Ed.), *Proceedings of the 10th International Joint Conference on Artificial Intelligence* (Milan, Italy). Los Altos, CA: Morgan Kaufmann Publishers.

Fischer, G. and Girgensohn, A. (1990). End-user modifiability in design environments, human factors in computing systems. *CHI'90 Conference Proceedings*. Seattle, WA, ACM, New York, April 1990.

Fischer, G., Kintsch, W., Foltz, P.W., Mannes, S.M., Nieper-Lemke, H. and Stevens, C. (1989). *Theories, Methods and Tools for the Design of User-Centered Systems (Interim Project Report, September 1986 - February 1989),* Technical Report, Department of Computer Science, University of Colorado, Boulder, CO, March 1989.

Fischer, G. and Lemke, A.C. (1988). Construction kits and design environments: steps toward human problem-domain communication. *Human-Computer Interaction,* Vol. 3, No. 3, pp. 179-222.

Fischer, G., Lemke, A.C. and Schwab, T. (1984). Active help systems. In G.C. van der Veer, M.J. Tauber, T.R.G. Green and P. Gorny (Eds.), *Readings on Cognitive Ergonomics - Mind and Computers.* Lecture Notes in Computer Science, Vol. 178, Berlin - Heidelberg - New York: Springer Verlag.

Fischer, G. and Mastaglio, T. (1989). Computer-based critics. *Proceedings of the 22nd Annual Hawaii Conference on System Sciences,* Vol. III: Decision support and knowledge based systems track. IEEE Computer Society, January 1989, pp. 427-436.

Fischer, G., Mastaglio, T. and Rieman, J. (1989). User modeling in critics based on a study of human experts. *Proceedings of the Fourth Annual Rocky Mountain Conference on Artificial Intelligence,* Denver, CO, June 1989, pp. 217-225.

Fischer, G., McCall, R. and Morch, A. (1989). JANUS: integrating hypertext with a knowledge-based design. *Proceedings of Hypertext'89,* ACM, November 1989, pp. 105-117.

Fischer, G. and Nieper-Lemke, H. (1989). HELGON: extending the retrieval by reformulation paradigm. Human factors in computing systems, *CHI'89 Conference Proceedings (Austin, TX),* ACM, New York, May 1989, pp. 357-362.

Furnas, G.W., Landauer, T.K., Gomez, L.M. and Dumais, S.T. (1987). The vocabulary problem in human-system communication. *Communications of the ACM,* Vol. 30, No. 11, November 1987, pp. 964-971.

Grudin, J. (1989). The case against user interface consistency. *Communications of the ACM,* Vol. 32, No. 10, October 1989, pp. 1164-1173.

Hutchins, E.L., Hollan, J.D. and Norman, D.A. (1986). Direct manipulation interfaces. In D.A. Norman and S.W. Draper (Eds.), *User Centered System Design. New Perspectives on Human-Computer Interaction.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Kass, R. and Finin, T. (1987). Modeling the user in natural language systems. *Computational Linguistics,* Special issue on user modeling, Vol. 14, 1987, pp. 5-22.

Kieras, D.E. and Bovair, S. (1984). The role of a mental model in learning to operate a device. *Cognitive Science,* Vol. 8, 1984, pp. 255-273.

Lemke, A.C. (1989). *Design Environments for High-Functionality Computer Systems.* Unpublished Ph.D. Dissertation, Department of Computer Science, University of Colorado.

Lewis, C.H. (1982). *Using the 'Thinking-Aloud' Method in Cognitive Interface Design.* Technical Report RC 9265, IBM, Yorktown Heights, NY.

Malone, T.W. (1983). How do people organize their desks? Implications for the design of office information systems. *ACM Transactions on Office Information Systems,* Vol. 1, No. 1, pp. 99-112.

Newell, A. and Simon, H.A. (1976). Computer science as an empirical inquiry: symbols and search. *Communications of the ACM,* Vol. 19, No. 3, pp. 113-136.

Norman, D.A. (1986). Cognitive engineering. In D.A. Norman and S.W. Draper (Eds.), *User Centered System Design. New Perspectives on Human-Computer Interaction.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Norman, D.A. (1988). *The Psychology of Everyday Things.* New York: Basic Books.

Reeves, B. (1989). *Finding and Choosing the Right Object in a Large Hardware Store - An Empirical Study of Cooperative Problem Solving among Humans.* Technical Report, Department of Computer Science, University of Colorado, Boulder, CO.

Reiser, B.J., Anderson, J.R. and Farrell, R.G. (1985). Dynamic student modeling in an intelligent tutor for LISP programming. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence, AAAI,* 1985, pp. 8-14.

Rich, E. (1983). Users are individuals: individualizing user models. *International Journal of Man-Machine Studies,* Vol. 18, pp. 199-214.

Schoen, D.A. (1983). *The Reflective Practitioner: How Professionals Think in Action.* New York: Basic Books.

Simon, H.A. (1981). *The Sciences of the Artificial.* Cambridge, MA: The MIT Press.

Standish, T.A. (1984). An essay on software reuse. *IEEE Transactions on Software Engineering,* Vol. SE-10, No. 5, pp. 494-497.

Suchman, L.A. (1987). *Plans and Situated Actions.* New York: Cambridge University Press.

Turner, A.A. (1988). *Mental Models and User-Centered Design.* Workshop Report (Breckenridge, CO), Institute of Cognitive Science, University of Colorado, Boulder, CO, Technical Report, No. 88-9.

VanLehn, K. (1988). Toward a theory of impasse-driven learning. In H. Mandl and A. Lesgold (Eds.), *Learning Issues for Intelligent Tutoring Systems.* New York: Springer-Verlag.

Wahlster, W. and Kobsa, A. (1988). *User Models in Dialog Systems.* Technical Report 28, Universität des Saarlandes, FB 10 Informatik IV, Sonderforschungsbereich 314, Saarbrücken, FRG.