The Knowledge Engineering Review, Vol. 7:4, 1992, 289–322

An overview of on-line assistance: from on-line documentation to intelligent help and training

BARBARA WASSON and SIGMUND AKSELSEN

Norwegian Telecom Research, PO Box 1156, N-9001, Tromsø, Norway

Abstract

An important aspect of any interactive computer system is its ease of use. On-line assistance is a major contributor to software usability. Conventional on-line assistance includes on-line documentation, or keyword access to textual help descriptions. More sophisticated on-line assistance includes intelligent help which provides immediate assistance for a specific problem, and intelligent training whose main goal is to support the user in learning about the system. These contemporary forms of intelligent assistance represent knowledge about the particular application, the user and the particular tasks the user is to perform. In addition, they possess inferencing capabilities to model the individual user's goals, plans, intentions and knowledge.

This paper provides an overview of on-line assistance research, from empirical studies that attempt to determine the nature of effective assistance, to the design of intelligent on-line assistance systems capable of tailoring the assistance to the individual user. For readers interested in more details, recommendations for further reading in the related areas of advice-giving and intelligent help, intelligent user interfaces, critiquing, intelligent tutoring systems, and on-line documentation and hypertext are given.

1 Introduction

In recent years, there has been a growing research field interested in providing on-line assistance for users of complex systems[1]. Key phrases such as "*empower the user*", "*user centred design*", "*advisory interfaces*", "*intelligent user interfaces*" and "*adaptive user interfaces*" have become the mottos of researchers who advocate that the quality of human-computer interaction depends on the sophistication of the interface and the on-line assistance provided to users performing their tasks. The trend is towards co-operation and cross-fertilization between researchers interested in human-computer interaction (e.g., interface design) and intelligent assistance[2] (e.g., help, advising, tutoring) systems. Advances in on-line assistance research should be of interest to those developing interactive computer systems such as information processing systems (e.g., word processors, data-base management systems), computer-mediated communication systems (e.g., mail, conferences, bulletin boards), and knowledge based systems such as decision support systems (e.g., anesthetic management, robot selection, speech recognition), or design systems (e.g., in architecture, engineering, software design).

Users of today's systems are demanding. They are interested in using systems to perform their required tasks and do not want to spend time searching for better ways to do things. This awareness of usability, coupled with the increasing complexity of interactive computer systems, places more responsibility on the designer to embue the system with the ability to provide sophisticated assistance. The novice user must be "guided" in how to interact with the application and perform required tasks. The more experienced user must be "supported" in exploiting the functionality of the product. Conventional on-line assistance includes on-line documentation, training manuals,

[1] Knowledge-based systems are to be included in the term *complex system*.

[2] The term *intelligent assistance* refers to approaches that utilize artificial intelligence techniques.

keyword access to help descriptions, as well as interface features. Two examples of such interface features are menus designed to intuitively provide information about the system, and scrollable windows that enable the user to "look back" at previous screens. These conventional forms of assistance must be accessed by the user. Recently, attention has turned to providing more sophisticated assistance that is instigated by the system. This requires the utilization of artificial intelligence techniques such as domain modelling, task modelling, user modelling, plan recognition and natural language understanding. Knowledge about the particular application, the user, and the particular tasks the user is to perform, is used to interrupt the user at such times as an error state is reached, a question can be answered, an opportunity for introducing a new concept is recognized, or sub-optimal behaviour is observed. This type of intelligent assistance is often in the form of training, tutoring, coaching, critiquing, or question answering. The goal of all types of on-line assistance, however, is to expand the knowledge and the skill of the user. By accomplishing this goal, the user will be able to make more efficient and effective use of the system's functionality.

This paper overviews the field of on-line assistance. Furthermore, it aims to provide a source from which those interested in incorporating on-line assistance into knowledge-based systems can draw. Section two focuses on the issues involved in designing effective on-line assistance systems with three categories of on-line assistance being identified: on-line documentation, on-line help and on-line training. The following three sections discuss and give examples of each of three categories, respectively. Section six discusses methodologies for including on-line assistance design in the development of software and shares experiences from an actual project. The paper is summarized, and the final conclusions are presented in section seven.

2 On-line assistance

Designing effective on-line assistance is a complex process where there are many alternatives to consider and many trade-off decisions to be made. A good understanding of the application itself (e.g., mail, medical consultation), the user population (e.g., novice, expert), the kinds of tasks that will be performed (e.g., sending and receiving mail), and the types of assistance appropriate to particular users and the task they are performing (e.g., giving definitions, advice, training) is necessary). Questions that address some of this knowledge include:

- What is on-line assistance? Why is it needed?
  —What kind of assistance should be provided?
  —Why does assistance often fail to be effective?
  —What advisory strategies are there? Under what conditions are different strategies effective?
  —How will the help be accessed? How will it be displayed?
  —Should both systems and user initiated help be provided?
- Who are the users of this system and what are their tasks?
  —What is the user's previous experience with the system, the task, and with computers?
  —Will the system infer the user's skill level from actions and responses, or by self-classification?
- Are different types of assistance needed for experts and novices?
  —What kind of questions do users ask? What kind of help do they really need?
  —How can a user's mental model of a task domain be incorporated into an advice-giving system?

This section discusses some of the issues and reports on empirical research that has attempted to answer the questions identified above. The first section introduces and defines the terminology that is commonly found in the literature pertaining to assistance systems. The following sections attempt to describe, in general, the underlying theories and motivations that drive this field of research. In particular, the problems in using complex systems, the users of such systems and the types of activities that users and advisors perform, are addressed. The section concludes with a summary, followed by a characterization of assistance that will be used to structure the remainder of the paper.

2.1 What is on-line assistance?

As in every area of research the terminology can be very confusing. Different terms are used to refer to the same entity, and the same term is used for a number of different items. For example, the terms *help systems*, *advice-giving systems*, or *advisory interface*, *intelligent support systems*, and *intelligent interfaces* are often used in the literature to refer to systems that provide some sort of *on-line assistance* for the user. Although each term is slightly different in its meaning, the systems built to support these ideas have the common goal of making the system more "useable", i.e., to make life easier for the user as they use the system to accomplish tasks. To clarify the terminology of the research communities which contribute to the area of on-line assistance, this section introduces the most common concepts found in the literature. This will lead to a better understanding of the goal of on-line assistance.

Kearsley (1988, p. 3) defines a help system as "one or more programs designed to provide user assistance embedded in a larger program or computer system". The help system is designed to provide *immediate* assistance for a *specific* problem the user has encountered. He distinguishes *help systems* from *on-line documentation*, which provides descriptive information about system functionality, and *on-line training systems*, which have the main purpose of instructing, and are more general in nature.

Carroll et al. (1988) use the term *advisory interface* to refer to the training, reference material, on-line help, and other advisory support available to users, and claim that it is one of the most important means for facilitating the useability of computer systems. They identify a phenomenon called the *production paradox* (Carroll et al., 1986), referring to a conflict for the user of a system between learning and working. The user wants to use the computer to accomplish a task, but does not want to invest time in learning and perfecting skills that would enhance their effectiveness and efficiency in using the system. Thus, the goal of such advice-giving systems is to "mitigate the learning versus working conflict by better integrating the time and effort spent on learning with actual use of the system" (Carroll et al., 1987, p. 14).

Fischer (1987) talks about *intelligent support systems* which use techniques from artificial intelligence to imbue the system with knowledge used to help the user learn about and use complex computer systems. Knowledge about the task, the user and about communication processes is represented. This knowledge enables the support system to "enhance incremental learning processes [learner is exposed to increasingly complex environments only when ready to move to the next skill level], to provide help and documentation, to support the understanding of existing programs and advice given, to assist in the construction of new systems" (Fischer, 1987, pp. 179–180).

The term *intelligent interface* is used to refer to an interface that provides services to its users in such a way that the interface and the user "co-operate" to accomplish the tasks the user must perform. According to Rissland (1984), the interface should be designed to provide the following services to its users:

- carry out menial tasks (e.g., set terminal characteristics)
- automate routine tasks (e.g., doing back up of active files, often)
- provide assistance with more complex tasks (e.g., file transfer protocols)
- provide easy access to tools (e.g., mail and conferencing systems)
- provide status information (e.g., inform when new mail has arrived)
- provide on-line assistance and documentation (e.g., help and manuals)
- allow multi tasking.

These services should be provided with the realization that different users have different needs, depending on the tasks they have to carry out, the tools they have available, their training and expertise, and even their personal tastes. Thus, intelligent interfaces support various forms of on-line assistance as well as providing additional functions (such as multi-tasking) to make life easier for the user.

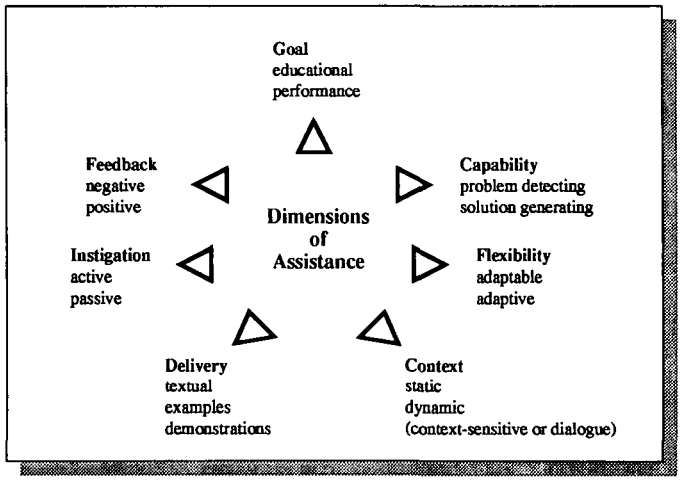


Figure 1 Characterization of assistance

2.2 *Characterization of assistance*

Figure 1 presents a characterization of assistance that can be used to facilitate discussion of the various forms of assistance.

Fischer et al. (1985) classify help according to whether the user or the system instigates the interaction. In *passive* systems, the user must explicitly request help (e.g., through a help menu), whereas in an *active* system the system interrupts the user when it determines that the user requires help (e.g., suboptimal behaviour is recognized). To distinguish between systems that take the context of the situation in which the interaction is taking place into account when providing help and those that do not, Kearsley (1988) categorizes helps as *static* or *dynamic*. Static helps are given independent of the context in which they are invoked. For example, the same help is provided whether the user is a novice or expert, or the user is trying to perform a legal or illegal move. Dynamic helps are dependent on the situation and are divided further into *context-sensitive* help and *dialogues*. In context-sensitive help, the help system has an understanding of the current context and this influences the help that will be provided. The current context is defined by such issues as what the user is trying to accomplish, what menus or options have recently been displayed, and how long it has been since the last help. In dialogue help, the system provides an answer to explicit user question. The help system engages the user in a dialogue to identify exactly what help the user needs. Often, this results in the help system guiding the user step-by-step through a problem (Kearsley, 1988).

Assistance systems can be analysed according to their goal. Fischer et al. (1990) describe *educational* systems where the main objective is to support learning (e.g., learn rules for designing a kitchen), and *performance* oriented systems where the focus is on helping the user produce better products (e.g., shorter time to create a document). They also differentiate systems based on their *problem detection* and *solution generation* capabilities. All systems can detect suboptimal behaviour and inform the user of this (e.g., display an error message to tell that a command has been used improperly), thus they are problem detecting. Some systems, however, have the capability to go beyond the mere detection of problems, and are able to suggest better alternatives, hence are solution generating (e.g., suggesting the appropriate command to use instead of just informing the user that a command is incorrect).

Assistance can also be described according to the type of feedback provided to the user. Again, Fischer et al (1990) talk about systems that provide both *negative* and *positive* feedback. During negative feedback, users are told about suboptimal aspects of their behaviour so that they can avoid it in the future. Positive feedback is given when good behaviour is recognized, reinforcing the desired behaviour. Finally, they talk about the flexibility of systems: is it *adaptable* and *adaptive*?

To accommodate to the various levels of users and individual user preferences, the help provided by the system must be adaptable—changeable by the user (e.g., the user can select the level of help to be received), or adaptive—the system changes its own behaviour based on observations of the user or inferences about the user (e.g., the system observes that the user is becoming more experienced and decides to provide less explicit help messages).

The delivery mechanisms of assistance are described by Tuck et al. (1990) as being *textual descriptions*, *examples* or *demonstrations*. Textual descriptions consist of written messages that must be read and understood by the user; examples are actual or simplified pieces of the application used to illustrate some condition; and demonstrations lead the user through steps that illustrate how to accomplish some task. During demonstrations the systems may either just animate the task, or may engage the user in interacting with the demonstration.

2.3 Empirical studies of users

This section discusses the users of complex systems. First, an argument that users only use a small percentage of a system's functionality is presented. Second, a review of empirical studies of the types of users is given. Third, investigations into the types of assistance that users require and the strategies employed by advisors are reported.

2.3.1 Using complex systems

In the preface to his book on on-line systems, Kearsley (1988, p. ix) states that "today's computer user expects to be able to use a computer system with minimal or no training and with no special technical expertise". Ironically, however, computer systems and the tasks to be performed are becoming increasingly complex. In addition, users are often unwilling to learn more than is necessary for the immediate solution of their current problem—they use the computer to accomplish tasks, and to satisfy their own goals and intentions. They tend to "ignore" or not "notice" whatever is irrelevant to the current situation. Complex systems are often designed so that the novice user can quickly learn a small set of basic commands which are sufficient for most tasks. Additional commands which will make more efficient and effective use of the system are there for the user to "grow into". It is not uncommon, however, for users "to become trapped by the simple complete set of basic commands and never progress to learning the full power [i.e., the functionality defined by the designer] of the system" (Finin, 1983, p. 176). In this case users are not even aware that assistance is needed.

As computer systems increase in complexity, the more important the on-line assistance becomes. Furthermore, there is general agreement that users only make use of a small percentage of a system's functionality resulting in inefficient use of these complex systems. In fact, Fischer et al. (1985) report that their empirical studies of users (using UNIX, EMACS, SCRIBE, and LISP) have shown that only about 40% of a system's functionality is exploited. They explain the levels of system usage using the diagram presented in Figure 2. D_4 represents the actual system and the set of concepts and associated commands available to the user. The subset of concepts and commands that are regularly used by the user, without problems, are contained within D_1. The concepts and commands represented by D_2 indicate the subset that is used only occasionally. The user typically does not know the details about them, or the effects (and side effects) of using them. D_3 reflects the user's *mental model* of the system. A mental model represents the user's internal representation of the system—that is, what the user believes (or thinks) the system provides. It includes not only correct beliefs, but incorrect beliefs as well. This is illustrated in Figure 2 by ellipse D_3 covering space outside of the actual system represented by rectangle D_4. Facilitation of the user's learning about, access to, and application of the capabilities provided by the system will increase the percentage of the system's functionality that is being used.

Careful examination of Figure 2 reveals that there are already some actual concepts that are in the user's mental model—those at the intersection of ellipse D_3 and rectangle D_4. These concepts

have not yet been used even occasionally by the user, but it should be relatively easy to support the user in acquiring this functionality. Those concepts that appear only in rectangle D_4, however, will be more difficult to add to the user's repertoire. The user does not know about the existence of these concepts so it is not likely that they will be discovered without some assistance. For these reasons Fischer (1988) argues that it is possible for the user to acquire the concepts already in their mental model by free exploration, but the acquisition of the unknown concepts will require some guiding and coaching. This suggests that various types of assistance are required.

Fischer (1987) cites preliminary empirical findings that identify several problems that prevent users from successfully exploiting the potential of high-functionality (i.e., complex) systems. The term *tool* is used to indicate features of the system that enable the user to perform efficiently and effectively. The problems are as follows:

- Users do not know about the existence of tools (and therefore they are not able to ask for them)
- Users do not know how to access tools
- Users do not know when to use the tools
- Users do not understand the results that tools produce for them, and
- Users cannot combine, adapt, and modify a tool to their *specific* needs.

From these findings it can be concluded that on-line assistance must cover not only the availability of system features (or tools) and how to access them, but also how and when to use the tools to accomplish the individual user's goals, plans, and tasks.

2.3.2 User types

Users of computer systems differ widely with regard to the amount of computer experience they have had, their assumptions and expectations about how a particular program will work, their understanding of the task they are to perform, and their experience in performing the task. Effective on-line assistance must be sensitive to these differences. Knowing who you are designing an application for is important in any software engineering endeavour, and it is no different with the design of on-line assistance (see also section 6). A common approach is to define categories of users which can be used to design appropriate on-line assistance (and be used for user modelling as described later). Three such categorizations are described below.

A cube, shown in Figure 3, with the three axis *computer experience*, *task experience* and *program experience*, is used by Kearsley (1988) to illustrate that users can lack skills or knowledge in any combination of these dimensions. Hence, there are various types of users and the design of-line assistance provided must take this into account. He argues that it is impractical to expect an assistance system to provide help for every distinct user type that could be represented by Figure 3.

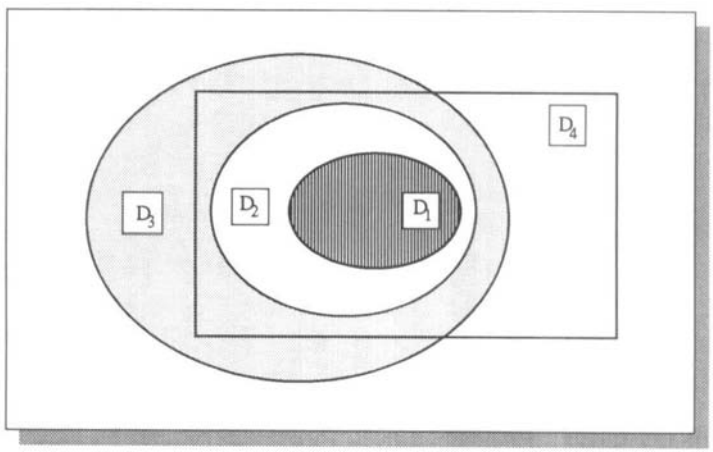


Figure 2 Levels of system usage (from Fisher, 1988, p. 139)

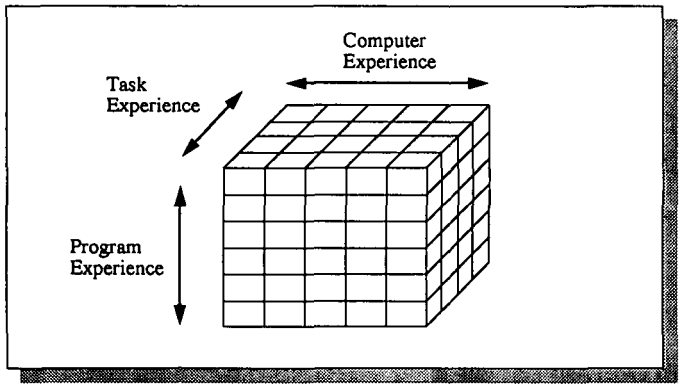


Figure 3 Three dimensions that helps should address (from Kearsley, 1988, p. 4)

Thus, data must be collected to identify typical users and typical user problems. He suggests three user groups and the type of assistance that they will require:

Novices: users who have little experience with computers will need help with basic concepts and operations. As a rule, novices want to see only information that is absolutely necessary.

Experts: experienced computer users want to know about limitations, shortcuts, and complex operations. They want to know anything that will allow them to do things more efficiently.

Casual users: people who only use a computer (or the current application) occasionally. They may be either novices or experts. Casual users need help remembering things they already know.

Rissland (1984) distinguishes between new users, occasional users, computer professionals, and users new to a tool. Each group may differ in the profession they represent (e.g., lawyers, doctors, chemists), and each of which will have their own idiosyncrasies (e.g., prefer mouse to keyboard commands). She argues that these experiences, goals and personal preference differences have implications for the type of explanations provided, the level of support given and the style of interaction. Fischer (1987) identifies intermediate users who want to learn something, and experienced users (or experts) who want critiques of their solutions to problems. Thus, there is agreement that the experience, goals and intentions of the user should influence the design of the type of assistance provided.

2.3.3 Activities of users and advisors

Carroll et al. (1987) report that although there is behavioural evidence that various forms of communication are used in advice giving, there has been no systematic investigation of the effectiveness of these styles. They suggest that the design of useable and effective systems will come from a general understanding of what advice is, how people generate and deliver advice, and how people make use of advice in the context of problems.

To identify the types of assistance that users require, two lines of investigation have been followed. The first is from the user's perspective: what activities do users engage in, what questions are asked, and what kind of assistance do users request? The second is from the advisor's view: what strategies are employed by advisors when interpreting user questions and providing advice? This section identifies studies that have addressed these issues.

Hartley et al. (1988) attempted to identify users' activities and the questions and explanations that arise as they learn about and use a software system. The users participating in this study were asked to articulate their thoughts and questions as they used the UNIX mail system. These protocols were then analysed to develop a computational model that describes the cognitive activities in which users engage and to classify user enquiries. Users may not necessarily be aware that they engage in these cognitive activities, but these computational models are useful to the system designer. They can be used, for example, in designing facilities for answering user questions.

Table 1 Activities and questions of users (adapted from Hartley et al., 1988, p. 346)

Activity	*Question type*	*Example*
Goal setting	Exploration	What can I do with Mail?
Planning	Enablement	Is the first step to specify who I want to send Mail to?
	Justification	Why use *next*?
Method Specification	Enablement	
Interpretation/ Evaluation	Interpretation/ Evaluation	Why did the last command fail?
Debugging	Interpretation/ Evaluation	
	Clarification	What is in my mailbox?
	Exploration	
	Orientation	What can I do next from here?
Exploration	Exploration	
Recapping	Clarification	
	Orientation	
	Elaboration	
Reorganization	Enablement	

Their initial experimental findings, summarized in Table 1, show that users engage in eight problem-solving activities with associated questions falling into one of seven categories. For example, when *setting a goal* such as "I will read all the messages" or "I need to find out how to send a mail message", the user often asks *exploration questions* that will further their knowledge about the system or verify their understanding of the system. Such a question might be "What can I do with Mail?".

Once a goal has been set, users tend to *outline a plan* using answers to *enablement questions* such as "Is the first thing I do to specify to whom I want to mail". This plan allows them to continue with their task. *Justification questions* like "Why use *next*?" aid the user in understanding why a decision implicit in the answer to an enablement question was made. Actual system commands are then used to *specify a method* to accomplish the goal step. Again, enablement questions such as "If I want to send a message to Thore, do I use the command mail thore.danielsen@fbt.tf.tele.no?" provide the user with the information required to proceed with the task at hand. If the user finds that the method they have chosen does not accomplish their task, the user will engage in an *interpretation/evaluation* activity using such questions that relate specifically to the understanding of the system's response. Using *interpretation/evaluation questions* such as "Why did that last command fail?", the user is involved in a *debugging* activity. This requires answers to *clarification questions* that give the user an indication of the system status (e.g., "What messages are left in my mailbox"), *orientation questions* that present alternative moves from the current position (e.g., "What can I do next?"), other exploration questions to set new goals, or interpretation/evaluation questions to understand what is going on. Hartley et al. (1988) claim that to be effective a system must be capable of providing assistance for both the formulation of goals, plans and methods, and the evaluation of the system's response to their actions. The assistance must be able to respond to user questions and engage in a development of explanations with the user.

Motivated by the lack of any well-articulated theory of diagnostic advice giving, Kidd (1985) was interested in determining the types of advice users actually require in a diagnostic task domain. Kidd studied naturally occurring consultations between a human expert and those seeking assistance, and came to the following general conclusions:

- Users were much more concerned with receiving advice about how to solve their problems than with detailed reasoning about the identification of the cause (i.e., they generally accepted the expert's diagnosis of the fault).
- Users took an active role in the problem solving process. They seemed to want to negotiate the solution with the expert.
- Users often had already thought out their own remedy for the problem and they only wanted the expert to evaluate and/or explain this for them.

These conclusions are supported by the findings of Aaronson et al. (1987) that users often ask advisors to verify hypotheses about possible solutions. This approach is often called critiquing (see section 4.3), and appears to be applicable for a wide range of users (Kidd, 1985).

The literature also contains reports from studies that address advising from the view of the advisor. Pollack (1985) reports that advisors most often answered questions or requests about how to do some action. In some situations, however, the advisor responded to an inferred goal and not the direct question. This suggests that an assistance system that is only capable of responding to direct user questions will not be sufficient. McKendree et al. (1986) categorize the roles of advising as *informing* users about available information, *defining* terms, *indexing* the user into appropriate solution sources for complex problems, and *structuring* poorly-understood problems. Furthermore, they suggest that the roles of informing and defining could be provided in an assistance system with an intelligent interface (recall section 2.1) to on-line documentation. In such a solution, a context-sensitive interpretation of a user's query could be used to narrow the responses to only those most likely to give the desired information. It is more complicated to provide the indexing and structuring roles. These will require an environment where mutual problem solving can take place.

2.4 Summary

Computer systems are becoming increasingly complex and more powerful, hence offering more functionality to the user. As Fischer et al. (1985) and others have pointed out, however, only a small percentage of the functionality of these systems is exploited. The reasons for this range from the user not knowing about additional features or how to find out about them, to having become too comfortable using the small subset of basic commands that are known. Users do not want to spend time learning about the system. They want to use the computer to perform tasks. The user must be provided assistance in the forms of support and training to become a more effective and efficient consumer. Thus, on-line assistance becomes an extremely important aspect of the computer system.

Furthermore, there is a wide variety of users differing in experience in using computers, using the particular application and performing particular tasks. Each individual user also has/his own goals, intentions and personal preferences for using a system. He/she performs a variety of activities from setting specific goals, to exploring the system for functionality. The user requires assistance from the system in carrying out these activities.

There are various lines of research addressing the design of on-line assistance. Empirical studies of the human–human advising and instructional processes are useful for identifying the types of assistance that users require and the roles of the advisor. Research reports indicate that the advising process is most often a co-operative venture. The user asks questions, proposes solutions and asks for advice. The role of the advisor is to answer specific user questions, verify hypotheses, critique proposed solutions, or introduce new information. These findings are useful to those actually designating and building systems that provide on-line assistance. The actual engineering of such systems is instructive in identifying what is successful and what is not, as well as providing insight into what is difficult to implement. An existing system with on-line assistance offers a vehicle for further empirical studies with real users using actual systems—human–machine interaction. Thus, providing on-line assistance is an interdisciplinary research paradigm for those

interested in human–computer interaction and those interested in providing on-line assistance of various forms.

The remainder of this paper will concentrate on systems that have been built to provide one or more of the most common forms of on-line assistance. First, the most traditional type of assistance, *on-line documentation* is discussed. Second, the progression of *on-line help* from simple static presentation of text to attempts to use artificial intelligence techniques to infer the user's goals, plans, and intentions is presented. Third, a description is given of two *on-line training* systems which have the main goal of aiding the user in learning to use the system to perform tasks.

3 On-line documentation

One of the earliest and most common forms of assistance is *on-line documentation* or manuals. These are essentially electronic versions of the printed reference material, or off-line documents, that describe the system and its functionality. These *keyword based systems* are often used to explain terms, describe commands and function keys, and search for related topics (Fischer et al., 1985). Kearsley (1988) describes such information as static fixed text entities that serve as an on-line glossary. The documentation is often retrieved through stylised keywords or command names. The UNIX *man* (UNIX, 1979) command is an example of this type of access to on-line documentation. The user types man word and all the pages of the on-line manual that match the word exactly are printed. One limitation of such systems, however, is that the system requires exact citations of the entries for which help is being sought. In addition, different systems use different vocabularies for the same functionality. Table 2 gives an example of the different command names used in various operating systems for removing a file.

More recently, access to the UNIX command manual has been improved by providing a graphical interface (implemented with Xwindows) as illustrated in Figure 4. All the available UNIX commands can be alphabetically displayed in one window. The user can retrieve a manual page for any of the commands by pointing to the command with the mouse. The manual page can then be displayed in other window. There is an option to view both the list of commands and a manual page in a split window format as shown in Figure 4.

Another passive access method is to use a *hierarchial menu* of topics and subtopics for which there is documentation. The DEC VMS help (DEC, 1978) is an example of this approach. The user selects a topic through a help menu which then shows the subtopics for that topic. This process continues until a subtopic that has on-line description has been selected. Navigation of these hierarchical menus requires some intuitive understanding of what the keywords mean, thus requiring the user to have some knowledge of what concept might give the desired functionality. Another disadvantage is that the descriptions are often verbose and the same response is given to all users for the same request (Jones et al., 1991).

The effect of on-line documentation on user performance was studied by Houghton (1984). The analysis found that the performance of experienced users was enhanced, but that of the inexperienced user was adversely affected. Jones et al. (1991) explain that experienced users are able to utilize the help to remind themselves of the syntax or available options of a command. They

Table 2 Alternative commands for getting rid of files (from Gwei et al., 1990, p. 365)

Operating system	*Command name*	*Root of name*
UNIX	rm	remove
VMS, VME, OS/2	del	delete
HARRIS	el	eliminate
CP/M, VM/CMS	era	erase
OS/360	decatalog	decatalog
OS/400	dltf	delete file

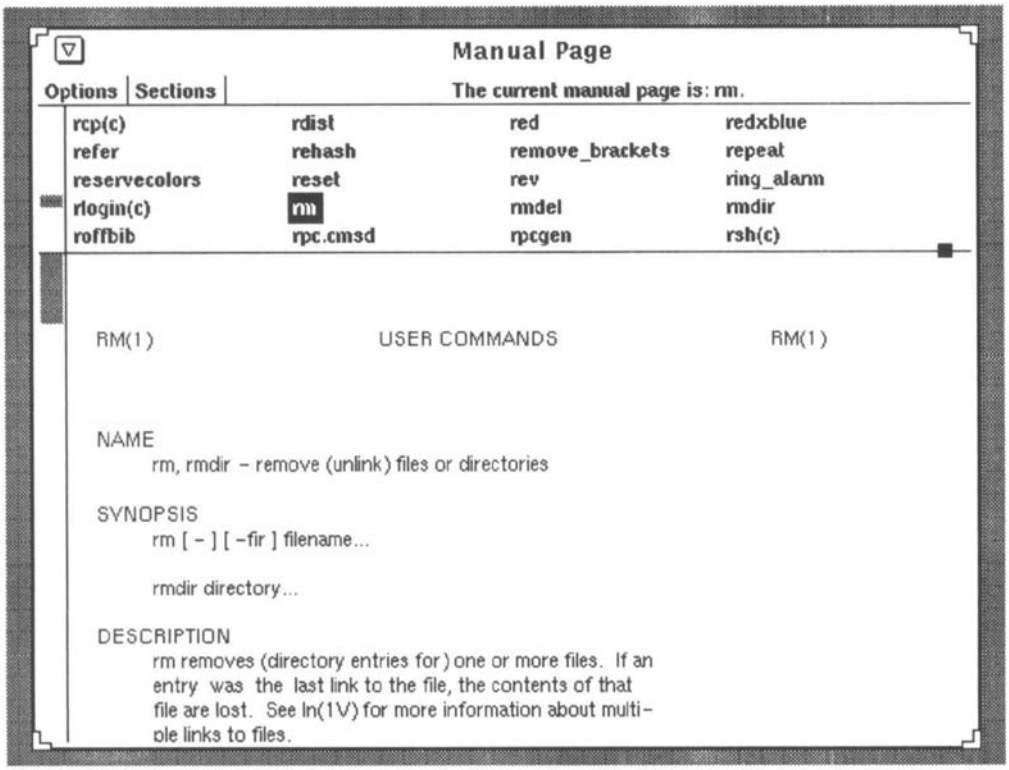


Figure 4 Sample UNIX xman screen for the *rm* command

have a framework from which to exploit the system help. This is not the case with inexperienced users. Their lack of a good framework prevents effective exploitation of the on-line documentation. These disadvantages do not mean, however, that there is no use for on-line documentation. Fischer et al., like Jones and colleagues, recognize that if the user knows the name of a command, but not its details, then keyword based help systems are a "quick, easy to implement and sufficiently reliable choice" (Fischer et al., 1985, p. 161). In particular, such systems can be used for explanations of terms, descriptions of commands and function keys, and searches for related concepts.

Current research into the use of *hypertext* systems for the representation and management of text offers a new approach to on-line documentation. A non-linear organization of text is facilitated by a database of text nodes and a network of links forming associations between the text components. Navigation of the text/network combination is supported by a *browser* that graphically displays the underlying network (i.e., the organization of the database) on the computer screen using windows and link icons. In his comprehensive introduction and survey to hypertext, Conklin (1988) cites the Emacs INFO Subsystem, an on-line help system for the text editor Emacs, and the Symbolics Document Examiner (Walker, 1985), the on-line help system for Symbolics machine documentation, as two examples of structured browsing systems. The focus in the development of structured browsing systems is ease of learning and use, with special importance placed on the interface. Another example is the FrameMaker[3] publishing software that offers the user a hypertext help system browser, shown in Figure 5, and the ability to create hypertext documents.

Hypertext is not without its problems. Two commonly cited disadvantages are *disorientation* and *cognitive overhead* (Conklin, 1988). Disorientation stems from the ability of hypertext systems to offer more degrees of freedom in movement around the text, thus raising the possibility of getting lost in the tangled web of links—i.e., losing one's location, often referred to as "being lost in

[3]FrameMaker® is a product of Frame Technology Corporation, 1010 Rincon Circle, San Jose, CA 95131, USA.

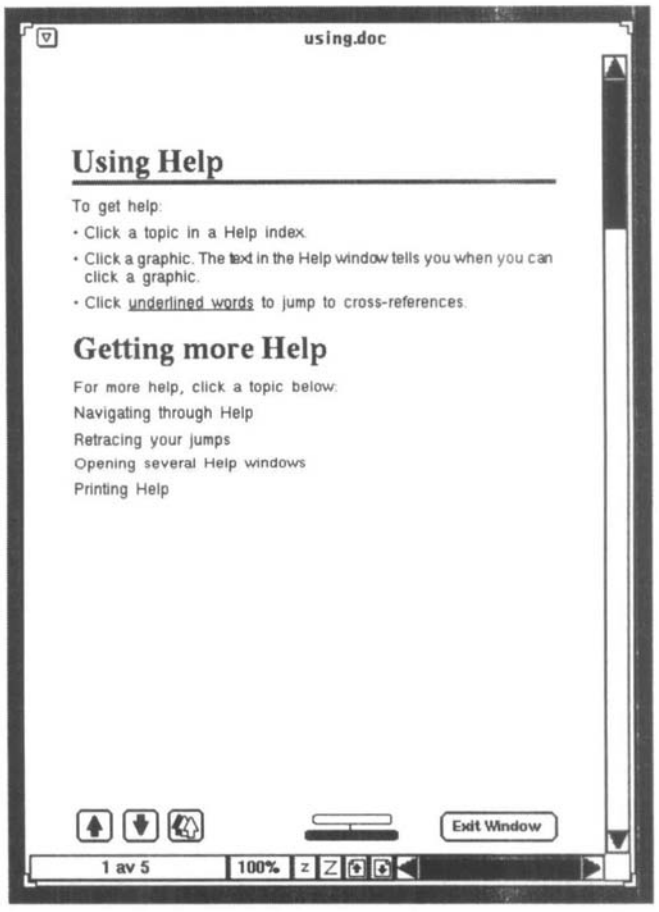


Figure 5 Sample screen from FrameMaker's Hypertext On-line Help

hyperspace". Cognitive overhead is a result of the need to create, name and keep track of links when creating your own hypertext documents—i.e., the user needs to maintain several trails of thought at one time. Research is currently addressing these problems. A promising approach to dealing with navigation difficulties is the combination of expert systems and hypertext technologies, referred to as *expertext* (Rada et al., 1988)[4] .

4 On-line help

Research and development of on-line help systems addresses some of the limitations of conventional on-line documentation systems. One goal is to move beyond pure browsing of commands and their functions provided by on-line documentation systems. The forms of help that have been developed range from the earliest static text-based systems that deliver the same message to each user, in all situations, to sophisticated dynamic systems that tailor the message to the individual user's experience level and the current situation. Currently, the common view that useability no longer refers only to an "easy to use" interface, requires the help system to be able to infer the user's goals, plans and intentions, and to be able to adapt to the idiosyncrasies of the individual user[5]. Finin (1983, p. 176) claims that "the ultimate success of an intelligent help systems will depend of the depth of its understanding of the task domain, the system it provides information about, and the user's knowledge of the system".

[4] A detailed discussion on the merging of the two technologies can be found in Rada et al. (1988, 1990).
[5] Although it is extremely difficult to adapt a system to the individual user—the user is a moving target as he/she adapts to the interface and help system (as pointed out by one of our reviewers)—this is the focus of user modelling research (see Kok, 1991, for a comprehensive review).

Three systems are chosen as illustrations of various types of on-line help developed to date. The first two sections provide an overview of the WIZARD and ACTIVIST systems, two early attempts at actively providing static help by watching for suboptimal user behaviour. This is followed by a section that describes a set of knowledge-based help systems based on the *critiquing* model (Miller, 1983). A critiquing system offers criticism on a user action, or user generated solution to a user specified task. For example, Silverman (1992) identified word processing critiques such as spelling or grammer checkers, and style replicators, and engineering critiques for areas such as VLSI design, and mechanical parts design. In each of these applications, the user chooses a task (e.g., write a particular type of document, or designing a particular chip) and receives criticism when desired. Thus, in the critiquing paradigm, passive or active dynamic help is provided as the user pursues his/her own goals and is illustrative of the many issues that contemporary help systems must address.

4.1 WIZARD: Recognizing the user's need

Finin (1983) described WIZARD, an experimental help system for users of a subset of the VAX/VMS operating system. The development focus of WIZARD was the problem of identifying when the user required help by recognizing correct, yet inefficient command sequences. By actively volunteering advice on how to perform tasks more efficiently, WIZARD helps the novice user become more proficient in using the operating system. The approach used to recognize opportunities for such advice is by matching the user's command sequence to "bad plans" represented in a catalogue. When a "less efficient" sequence is recognized, WIZARD constructs a help message that gives immediate advice (i.e., an alternative, more efficient action or action sequence), and/or points to an on-line documentation entry that will provide relevant information. If no on-line documentation is available, the help message may suggest off-line documentation to consult.

WIZARD uses a knowledge base of concepts that represent the objects and processes that define the task domain (e.g., deleting a file), the generic and specific commands available in the system to carry out the task concepts (e.g., the command DELETE) and a history of the interaction of the user and the system (e.g., user 243 used DELETE at 15:34 on 12.10.91). A "bad plan" is represented as a sequence of events where an event is related to a concept or concepts from the knowledge base. Each plan has a goal associated with it (e.g., rename a file) and when the plan is recognized, the goal is taken as the goal of the user and used to generate advice for more efficiently realizing the goal. Figure 6 illustrates a "bad plan" and the advice given by WIZARD. The user's goal is to rename file TEST1 to TEST2. This is successfully carried out by the "bad plan": COPY TEST1 TEST2; DELETE TEST1.

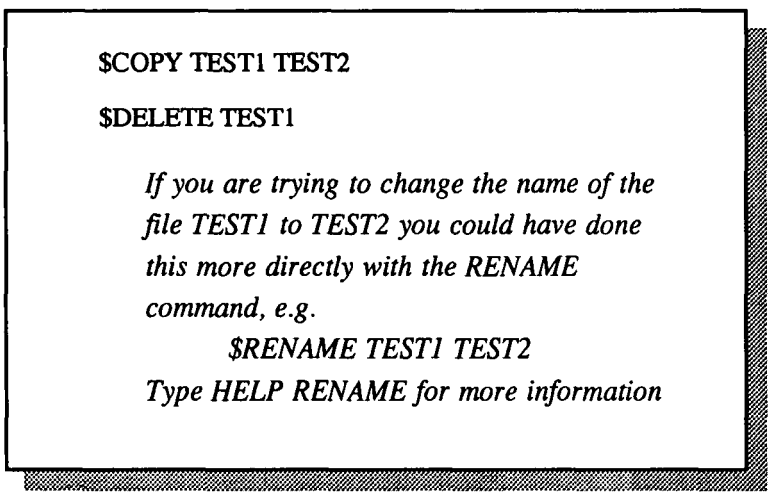


Figure 6 Example of WIZARD advice (from Finin, 1983, p. 177)

This is one of the early attempts at providing dynamic context-sensitive help tailored to an individual user's goal. WIZARD, however, only deals with situations where inefficient sequences of actions have *achieved* the user's goal (i.e., bad *but* successful plans). More sophisticated inferring of user intentions, goals and plans will have to be done to recognize situations where users *cannot achieve* their goal.

4.2 ACTIVIST

Fischer et al. (1985) describe ACTIVIST, an active help system that watches the user for suboptimal behaviour while using an editor. ACTIVIST, like WIZARD, recognizes when a complex command can be used in place of suboptimal commands. In addition, ACTIVIST can identify situations where the user knows the complex command, but does not use the minimal key sequence to issue the command. For example, the user types a command name instead of using the corresponding function key.

ACTIVIST uses *plan specialists* to recognize and evaluate the plans of the domain. Examples of typical plans are "delete the next word", or "insert a word". Each plan specialist uses: (i) a *transition network*, which maps all the different ways the functionality of the editor can be exploited to accomplish the plan, and (ii) an *expert*, which knows the optimal way to carry out the plan. The optimal plan path includes both the best editor commands and the minimal key sequence for executing these commands. When a plan specialist recognizes that the user has carried out their plan, the sequence of commands used is compared to the "best" solution. If it is not the best solutions the user may receive advice. If the recognized commands are the "best plan" commands, the actions are then compared to the minimal key sequence and if they do not match, then the user may receive advice.

ACTIVIST goes a step further than WIZARD in that a simple user model is maintained to vary the help strategy of ACTIVIST with the behaviour of the user. Figure 7 shows the user model (the top box) for a user working in the editor window (the bottom box). The user model records, for each plan:

- how often the user does the plan
- how often the user does the plan with the optimal commands
- the wrong commands used
- the wrong keys used
- how often this plan's message should be given to the user.

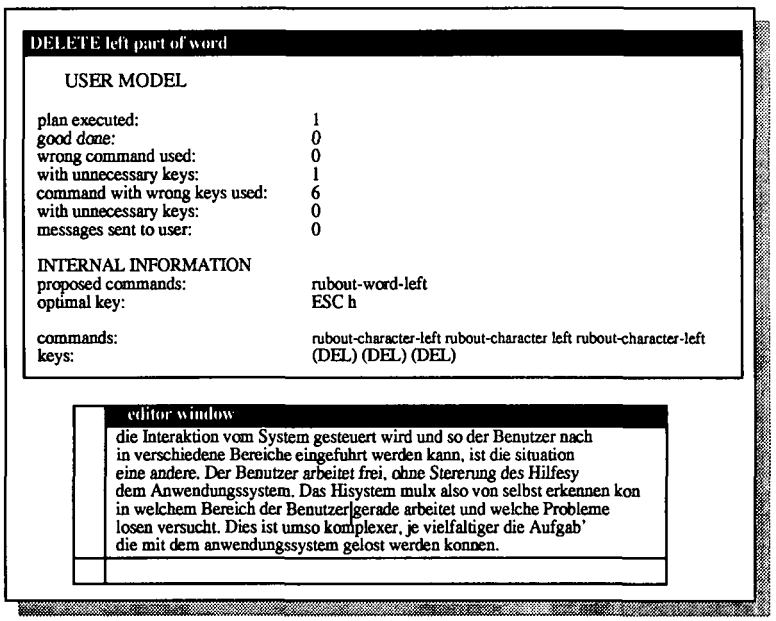


Figure 7 Monitoring the recognition task (from Fischer et al., 1985, p. 166)

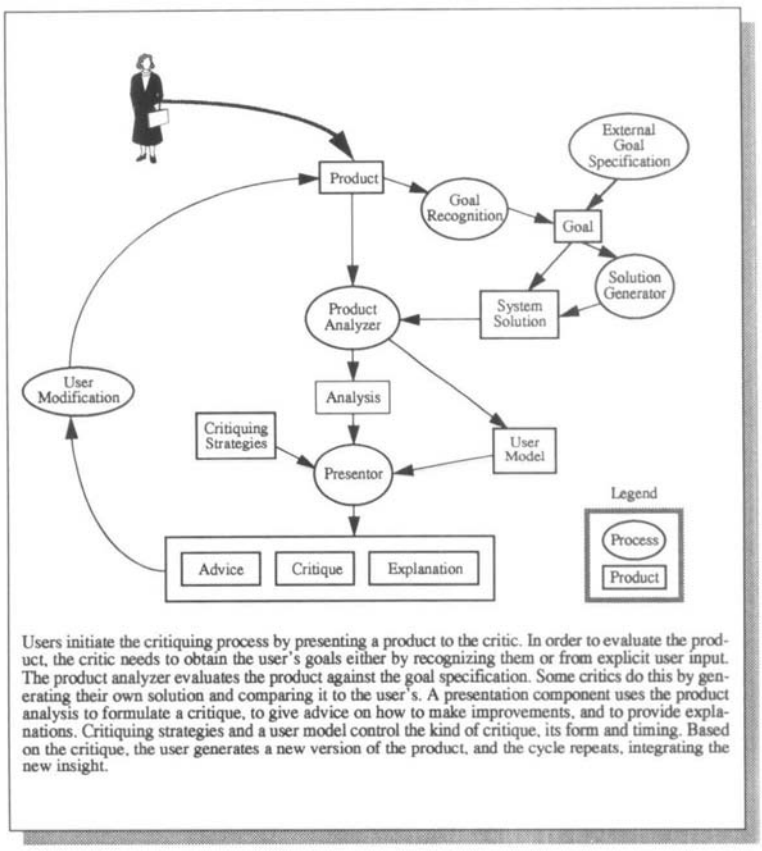
The help strategy depends on the time the last message was given and the number of times this particular error has resulted in an error message being given. Thus, ACTIVIST includes heuristics about when to give advice. For example, if a help message has been given to the user a set amount of times, the system will no longer give that message if the suboptical plan that triggers it is recognized again. This allows the user to decide to ignore the advice and proceed in their own way. Also, advice about a plan is given only immediately after the plan was done wrong—never at a later time. This is illustrative of the issue of intrusiveness. It is difficult for the system to infer if the user has not understood an error message or has chosen to ignore it. One possible solution is to provide a means for the user to turn the advising off so the same help message is not given over and over again.

The approach used in ACTIVIST of watching for user intentions and actions by having one plan per possible goal is not sufficient for realistic applications (Fischer et al., 1985). There will be too many possible plans. Thus criteria such as "very complex plans are not relevant for novice users" might be used to prune and focus the plan monitoring. Plan recognition is an active research topic as evidenced, for example, by a special issue on plan recognition in the *Journal of User Modelling and User-Adapted Interaction* (1991). Fischer also argues that the approach of having the help system as an addon to an existing system instead of being an integral part of the initial design, is not ideal.

4.3 Critiquing

Motivated by a goal of increasing the useability of high functionality computer systems, Fischer and his colleagues have been developing knowledge-based help systems based on the critiquing paradigm. Useability is increased by facilitating learning about, access to, and application of the knowledge a system contains. This critiquing process, illustrated and described in Figure 8, enables the user to pursue his/her own goals with the program interrupting only if his/her behaviour is judged to be significantly inferior to what the program would have done. The critic and the user work cooperatively to solve the user's problems. The critic "looks over the shoulder" of the user for errors and suboptimal situations for which it can make improvement suggestions. The user can then use the information to fix the problems, seek additional advice or explanations, or ignore the information altogether. Thus, control is in the hand of the user.

Critics are distinguished from advisors. Advisors are the *primary source* for a solution and *do not require* the user to submit either a partial or proposed solution to a problem. Critics, on the other hand, *require* a partial or full solution in which they will look for deficiencies. To date, Fischer and his colleagues have developed three systems based on various aspects of the critiquing approach. LISP-CRITIC (Fischer, 1987; Fischer et al., 1991) is a system used by LISP programmers both to improve the code they are writing, and to receive programming knowledge on demand. The critic can have as a goal to either provide feedback on how to make the code more cognitively efficient (i.e., easier to read and maintain), or more machine efficient (i.e., faster or requiring less memory). These two goals are distinguished as *educational* and *performance* critics, respectively. FRAMER (Lemke, 1989) is a design environment in which designers can receive critiques on program frameworks—the components of window-based user interfaces on Symbolics LISP machines. The user is provided with a palette of items such as tile-panes, display-panes and menu-panes, which are to be put together into a framework where programming can take place. The first implementation employed a *passive* strategy. Empirical evidence revealed, however, that users often invoked the critic too late—after a major design decision had been made. Thus, a second version implements an *active* strategy where a continuous display of help messages is available for the user. JANUS (Fischer et al., 1990) is a design environment for constructing residential kitchens. Like the LISP-CRITIC, JANUS is both a learning environment for design students and a tool for skilled designers.



Users initiate the critiquing process by presenting a product to the critic. In order to evaluate the product, the critic needs to obtain the user's goals either by recognizing them or from explicit user input. The product analyzer evaluates the product against the goal specification. Some critics do this by generating their own solution and comparing it to the user's. A presentation component uses the product analysis to formulate a critique, to give advice on how to make improvements, and to provide explanations. Critiquing strategies and a user model control the kind of critique, its form and timing. Based on the critique, the user generates a new version of the product, and the cycle repeats, integrating the new insight.

Figure 8 The critiquing process (from Fischer et al., 1990, p. 342)

To highlight some of the key issues of the critiquing process, the JANUS system is described in more detail. JANUS is composed of two subsystems: JANUS-CRACK and JANUS-VIEWPOINTS[6]. JANUS-CRACK, shown in Figure 9, is the knowledge-based design environment in which the user constructs a kitchen and receives critique messages on the design. The user selects components of a kitchen from the Palette to construct a kitchen in the Work area. The critiquing messages appear in the Message window. The Design state area of the screen lists the components used to date. The Commands window remind the user of what steps need to be carried out. The Catalog is available for selecting predesigned kitchens (either good or bad designs) that can be either critiqued or modified.

JANUS-VIEWPOINTS is an issue-based explanation system that the user can access to display an argumentation for a particular message (i.e., the reasons the particular critiquing message was given). This allows the user to assess the critique and decide whether to accept or reject it, as well as learn the underlying principles and avoid similar design problems in the future. Explanations can be based on differences between the user's and system's solutions, referred to as *differential* critiquing, or on violations of general design principles called *analytical* critiquing. JANUS uses an analytical critque where a set of rules define building costs, safety standards and functional preferences that the user's design must meet. For example, the message "DOUBLE-BOWL-SINK-1 is not in front of a window", given in the Messages area of Figure 9, is a violation of the rule "sinks should be in front of a window". The messages provided by a critiquing system can either be *problem detecting*, as illustrated above, or *solution generating* where the critic goes a step further than just detecting a problem and suggests an alternative solution.

[6]JANUS is illustrative of the integration of knowledge-based and hypertext technologies. A description of this integration can be found in Fischer et al. (1989).

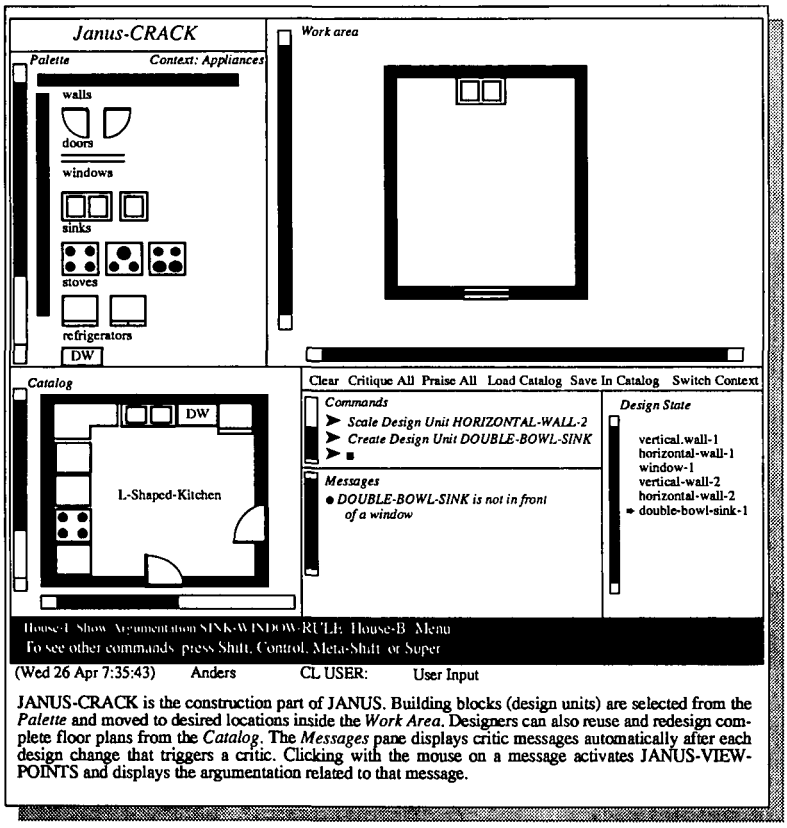


Figure 9 JANUS-CRACK: the SINK-CRITIC (from Fischer et al., 1990, p. 340)

The user of JANUS has two options as illustrated in Figure 9. To design by composition the user *builds* a design in the Work area using design units (e.g., doors, sinks, stoves) from the Palette, or to design by *modification* the user selects a prestored design from the Catalog. The Catalog includes both good and bad designs which enables the user to decide whether to learn about the good features—*positive* critiquing via the "Praise All" button, or the bad features—*negative* critiquing via the "Critique All" button. Users can also add their own designs to the catalogue.

The critiquing paradigm addresses some of the major challenges facing developers of intelligent on-line help. For example, critiquing provides both static and dynamic help. The user can request critiquing via special buttons. The system provides continuous appraisal of the design as the user is working. Thus, issues such as what to monitor and when to interrupt arise. Intervention strategies are needed that determine when and how a critic should interrupt. Another questions is "What should be critiqued?". Whether a critic is an educational critic where learning is the main focus, or a performance critic where learning is a by-product, has implications for the critiquing strategies. This draws on education, cognitive science and psychology research to define effective learning situations and advising strategies.

Providing an effective interface where the user is able to easily use the interface, as well as infer information about the system from the design, requires the skill of those interested in human–computer interaction. Tailoring help to the individual user requires artificial intelligence techniques such as knowledge representation, plan recognition, natural language understanding and user modelling.

4.4 Summary

The current trend is to provide intelligence in on-line help that increases the usability of complex systems. There is a drive to infer what the user is doing. The earliest systems like WIZARD and ACTIVIST began by recognising known sequences of commands. Recently, paradigms like

critiquing, have taken this reasoning further by attempting to infer the user's goals, plans and intentions. Contemporary help systems take a knowledge-based approach that uses techniques and methodologies from artificial intelligence to give the ability to "understand" the system, the user and the advising process. To meet this requirement, the system must represent large amounts of knowledge about specific domains[7]. Fischer (1988) identifies five domains of knowledge required to enable critiquing to take place:

- knowledge of the problem domain
- knowledge about communication processes
- knowledge about the communication partner
- knowledge about the most common problems that users have in using a system
- knowledge about instructional strategies.

The domain knowledge is the representation of the application or program the help is designed for. It is used to infer how the program works, how concepts relate to one another, why an error has occurred, and how the tasks can be performed. Knowledge about the communication process includes how windows, menus or pointing devices are used, and how information is exchanged implicitly between the user and the system. This is often referred to as the "interface". Since *the user* of a system does not exist, knowledge about the communication partner enables the system to be sensitive to the individual user. A user model represents the system's beliefs about the user's conceptual understanding of a system. In addition, it keeps track of: the set of tasks performed by the individual user, the user's way of accomplishing domain-specific tasks, the pieces of advice given, whether the user remembered and accepted the advice, and the situations in which the user asked for help. Providing relevant advice and support requires knowledge about the most common problems that users have in using a system and in instruction/advising/tutorial strategies that can be used to assist the user. The most effective on-line help systems will base this knowledge on pedagogical theories. These issues parallel those that arise in the development of artificial intelligence-based educational systems often referred to as intelligent tutoring systems (ITSs), intelligent computer-aided instructional systems (ICAIs), or intelligent learning environments (ILEs). As the development of such systems is a research field in its own right, the interested reader is directed to the list of recommended readings[8].

5 Training systems

The distinction between help systems and training systems is not sharp. When does help become training? For example, the critiquing approach discussed in the previous section, is suitable for educational purposes. The user can use the critics to learn the rules for design. The guided task paradigm described in this section could be categorized as a help system. It has been classified as a training system in this paper, however, because it has the goal of training the user to use the Mickey User Interface Management System (Olsen, 1986) to build graphical user interfaces. The EUROHELP project, also presented in this section, aims to provide both help and learning functionality for users of information processing systems (e.g., Unix-Mail). It has been included in this category because it is presented as an example of an intelligent tutoring system, in particular a coaching system, that deals with issues such as teaching stategies and didactic discourse.

Kearsley distinguishes between help and training only in their goal. Help systems have the goal of supporting the user in performing a task. Training systems have the explicit goal of supporting the user in learning. Those interested in these two applications of the technology have much to offer one another. As Sullivan (1991, p. x) points out "the combination of these two areas offers a

[7] A detailed discussion of these sources of knowledge is beyond the scope of this paper.

[8] In addition to the general recommendations, more specialized discussion can be found in Wasson (1990) and Akselsen (1991). Wasson proposes a framework for planning the content of an instructional interaction, and Akselsen presents a general architecture for an ILE in the domain of fish disease diagnosis.

powerful capability in which one system could both support the use of a complex system as well as provide training to new users and retraining to experienced users when upgrades to the system are made".

5.1 Guided tasks

Tuck et al. (1990) describe a *guided task* paradigm where the user is guided by the system, step-by-step, through a demonstration. This paradigm is different from other forms of help in that instead of giving help on individual commands, it combines commands into tasks which address application domain problems. It also provides an interactive demonstration of the task in the actual interface in which the user will be performing the task. Thus, the user *actively* participates in the acquisition of new knowledge. Although empirical testing of the effectiveness of this approach has not been carried out yet, it is felt that "involving the user in the demonstration is much more effective than just animating the task" (Tuck et al., 1990, p. 74).

The user selects a task from a library of task scripts (i.e., sequences of statements) which describe the various tasks the user can receive guidance on. For example, the user might want guidance on how to size windows on a screen, draw a line, or place text in an active window. Once the task has been selected, the system will guide the user through the action sequence required to perform the task, giving pointers and descriptions of how to carry out the actions. These actions are as basic as how to select an item from a menu, where to click on the screen, or how to stylize (i.e., choose the font, size, and emphasis) text. Figure 10 illustrates the pointer guidance provided to draw a line. First the user is shown the menu in which the "Line..." command is found (see Figure 10(a)), and is told to select this menu. Once the menu is open (see Figure 10(b)), the command "Line..." is pointed out and the user is told to select this option. Once this command has been selected, the user is requested to draw the line as directed in Figure 10(c).

Tuck et al. (1990) report that running the first subjects through sample task demonstrations revealed that although they ran quickly through the demonstrations, and could do what they were told, the users did not have a clear *understanding* of what they were doing. This revelation lead to including dialogue boxes that give textual directions to augment the pointers, and an explanation box that gives the semantics of the entire task being performed. Figure 11 shows a sample screen layout that illustrates these various help components. The three dialogue boxes under the "Guided Tasks" header are displayed as needed. The top box is the script title box, which is displayed throughout the session. This box gives the name of the task and provides a "quit" button that allows the user to terminate the demonstration at any point. The second box gives the current statement

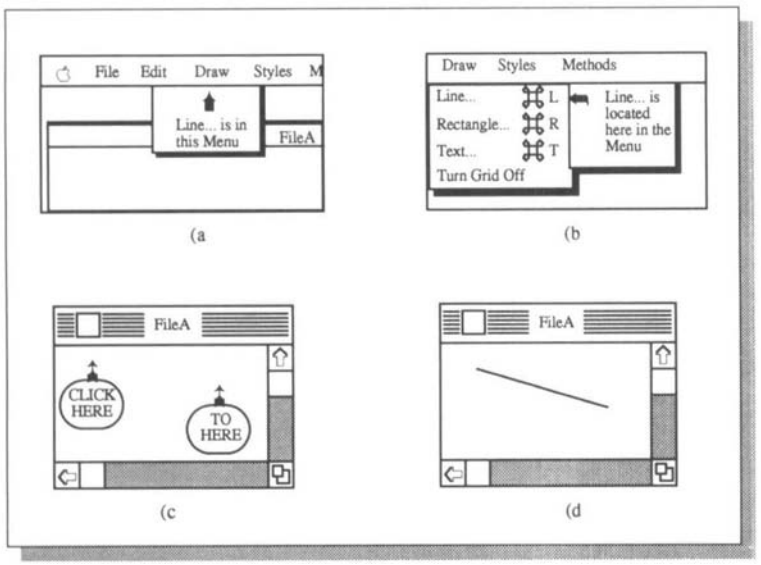


Figure 10 Guidance on drawing a line (from Tuck et al., 1990, p. 73)

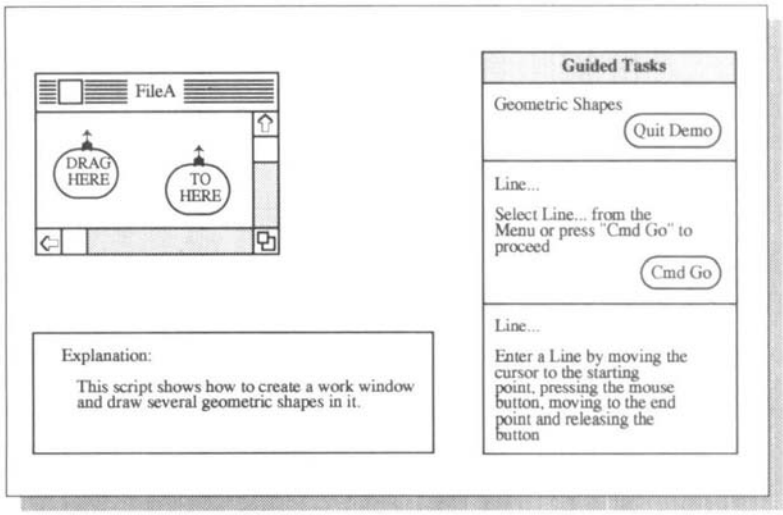


Figure 11 Sample guided task screen (from Tuck et al., 1990, p. 77)

name with instructions of how to carry out the action. The "Cmd Go" button provides a means for the user to direct the system to carry out the action without the user participating. This feature is included to cover the situation where the user feels familiar with the concept and wishes to pass over particular statements. The argument box, the third box, is shown only if the user is going to participate in this part of the demonstration (i.e., has not pushed the "Cmd Go" button). The purpose of the box is to explain how to perform the required actions. The explanation window at the bottom of the screen provides additional reasons why the user is being asked to perform certain actions. For example, the explanation given in Figure 11 gives the semantics of the task script as a whole: *to create a work window and draw several geometric shapes in it.*

Tuck et al. (1990) refer to guided task help as a static help demonstration. The state of the user does not influence the running of the demonstration. The paradigm is a good design for the novice user who needs to be introduced to a system. The user chooses what task to have demonstrated and the parts of the demonstration in which to participate. The task demonstrations can be run whenever the user wants, thus do not interrupt their line of concentration.

Although the guided task paradigm described in this section is currently a research project, there are a few products on the market that use part of the same principle of interactively engaging the user in demonstrations and providing a means through which the user can be guided through the activity. One of these is the Apple System 7 *Networking Basics Tour*[9]. In this tour, the user is guided in providing network access to a file on their own hard disk, and guidance on how to access a remote file that has been shared. The user must carry out the actions described by the "tour guide" and is given the option of being shown exactly what to do. Opportunity for practicing a unit that has just been guided is also provided. Reports on the effectiveness of this approach have not been seen, but the authors have tried the demonstration and feel that there are some aspects of the design that could be improved upon. Further discussions on this project appear in Section 6.

5.2 EUROHELP

The EUROHELP project (Breuker et al., 1987; Breuker, 1988; Hartley et al., 1988), is one of the most ambitious intelligent on-line help systems project to be undertaken. Collaboration among five institutions[10] has lead to a detailed investigation into many of the methodologies and issues involved in building intelligent on-line help systems. This research has contributed many ideas and

[9]Developed by MacroMind Inc., 1028 W. Wolfram, Chicago, IL 60657, USA.

identified many issues involved in developing on-line assistance. In addition, a prototype help system for the UNIX-Mail utility[11], has been developed.

The primary goal of EUROHELP is to provide information about the use of information processing systems (IPS) *when needed* by *any type* of user. The need can be identified by the user (passive) or inferred by the system (active). The advice is tailored to the individual user. The advising strategy is that of a human coach who (i) watches over the shoulder of the user to interpret the performance when things go wrong or when there is an opportunity to extend the repertoire of the user, and (ii) is able to answer questions in the context of current use of the IPS. According to Hartley et al. (1988), the EUROHELP coach can:

- engage in a dialogue with the user when asked questions on:
  —plans
  —particular commands
  —system effects
  —errors.
- interrupt the user:
  —when feedback from the IPS is judged to be inadequate for the current state of user knowledge
  —when the opportunity to extend the repertoire of the user arises
  —when the learner is about to make a catastrophic error
  —to provide material to correct misconceptions or errors
  —to comment on plans which seem inefficient.

The EUROHELP architecture, shown in Figure 12, consists of three main modules, the QUESTION INTERPRETER, the PERFORMANCE INTERPRETER, and the COACH. Several knowledge bases are used by the various components to execute their functions. A hierarchical knowledge base of all the known tasks and plans for carrying them out is represented in the applications model. Thus, the applications model represents the help system's expertise in the application. Another knowledge base is the user model which represents the plans and goals the system believes the user if familiar with, and the frequency of use of each. The user model is an overlay of the application model[12] .

The QUESTION INTERPRETER is responsible for answering user questions. To avoid dealing with the difficulties of interacting in natural language, the user is restricted to selecting questions from a specially generated menu of questions. This menu of questions is generated by the QUESTION INTERPRETER based on the user's current task and current knowledge state (obtained from the user model). The types of questions that are generated for the question menu are based on the empirical work of Hartley and his colleagues as discussed in section 2.4. These include elaboration, enablement, interpretation, orientation, justification, clarification and evaluation questions. Details of question answering issues for on-line help systems and a report of the empirical study can be found in Hartley et al. (1988).

The PERFORMANCE INTERPRETER is concerned with recognizing the user's current plan, and diagnosing inefficient or incorrect plans. This is highly dependent on the ability of the system to infer the user's intentions. Responsibility for inferring the user's intentions lies with three co-operating components: the PLAN RECOGNIZER, the PLANNER, and the DIAGNOSER. The PLAN RECOGNIZER tries to "explain" (i.e., recognize) the user's behaviour by matching it to one of its known plans. If the plan is recognized, the PLANNER generates an alternative plan for the same task. If the generated plan is different than the recognized plan, the DIAGNOSER attempts to explain the differences. The results of the PERFORMANCE INTERPRETER,

[10]The participating institutes include the universities of Leeds and Amsterdam, ICL (Manchester, UK), Courseware Europe (Zaandam, the Netherlands), Axion (Birkerød, Denmark).

[11] A prototype that is implemented in Interlisp and runs on a Xerox 1186. Information about the prototype is available from Axion, Bregnerodvej 144, DK-3460 Birkerød, Denmark.

[12]The overlay model (Carr et al., 1977) approach to user modelling assumes that the user's knowledge is a subset of the expert's knowledge which is represented in the domain/application model.

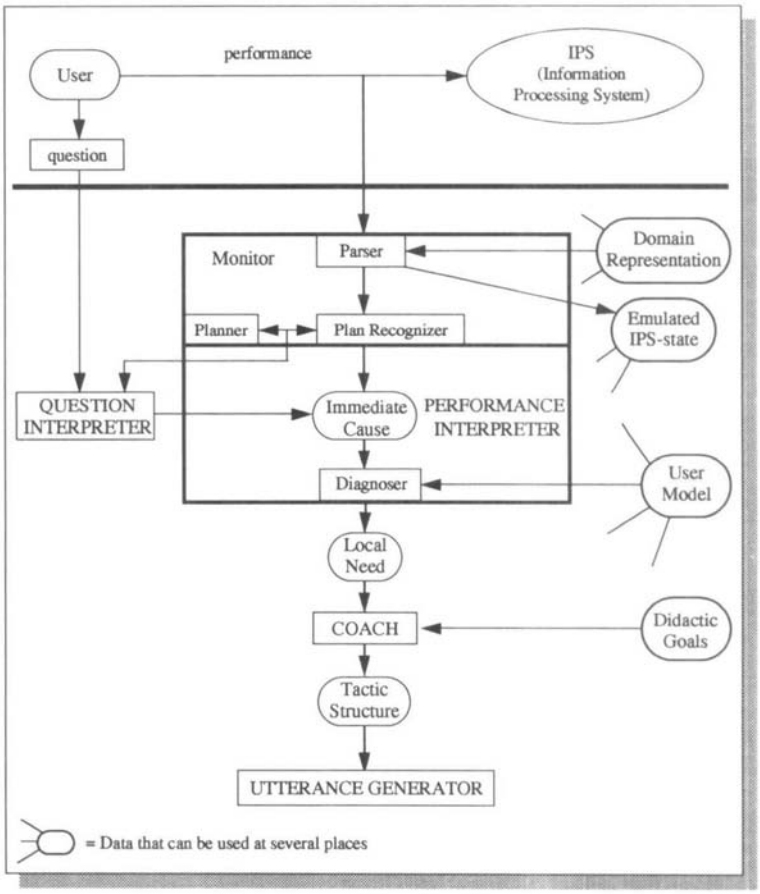


Figure 12 Architecture of EUROHELP (from Breuker et al., 1987,p. 168)

referred to as needs, are passed to the COACH. Details of performance interpretation in EUROHELP can be found in Stehouwer et al. (1989).

The COACH has two responsibilities. First, to help the user with a current problem, referred to as a local need. Second, to teach the user about the IPS, called a global need. Local needs require immediate attention. Global needs can be thought of as learning goals, hence they are long term goals which do not necessarily need immediate attention. The COACH uses the local and global needs to determine whether to provide help, or to teach. If the local need can be related to a global need, then the COACH teaches. Otherwise it simply presents the required information. When the COACH receives a need, it first decides whether to provide pure help or to coach. Once this decision has been made, the COACH tries to find a strategy (represented as a skeleton plan) from a library of stereotypical plans, to meet the need. If a match is found, the skeleton plan is instantiated to the current situation and planning is complete. The COACH has several coaching strategies available which depend on the user need. If the user is missing a basic concept, then the strategy of *initial instruction* is used. In EUROHELP there are five basic concepts that enable the novice user to perform almost all of the tasks—but often not in the most efficient way. If the user lacks potentially relevant concepts, then the *expansion* strategy is used to enlarge the user's repertoire by instructing about new concepts. This strategy is used when a precursory concept has been mastered and there is an opportunity for immediate application of the new concept. *Feedback* is given if there is a lack of information about the correct action—the system describes, to the user, the effects and side effects of correctly used commands. If an error or misconception has been identified, then a *remedial* strategy is used.

The EUROHELP prototype provides both passive and active help, satisfies both educational and performance goals, is problem detecting and solution generating, and dynamically provides

context-sensitive and dialogue feedback. It is representative, like the critiquing approach presented in section 4, of contemporary intelligent help systems—it is a knowledge-based approach that uses AI-techniques such as plan recognition, plan generation, and user modelling. Questions about how the represent the application, how to represent the user's current knowledge state and, how to recognize the user's goals, plans and intentions, must be addressed. The focus of providing explicit instructional functionality and the ability to answer user's questions, however, gives additional research questions. Questions about the instructional process include: when to interrupt, what to say, and how to say it. Research into question answering deals with recognizing the user's intention, and generating an answer tailored to the individual user.

5.3 Summary

On-line training can refer to both training the user in the use of the application and how to perform tasks, and to providing on-line instruction to aid the user in learning new concepts. The Guided Tasks paradigm and the EUROHELP project are illustrative, respectively, of such systems. More emphasis is placed on having the user understand the actual tasks that they are performing. The interaction with the user is to satisfy the explicit goal of helping the user learn some knowledge that will aid their performance—it is important that the user knows, at a conceptual level, *why* the assistance has been given. This is in contrast to on-line help where the emphasis is on helping the user get over an immediate hurdle and on with the task at hand—the user may not even be informed of what the problem is, or why this particular solution will work. The difference is, of course, subtle. All the systems reviewed in the last three sections could be thought of as having the goal of helping the user learn something. The distinction between on-line documentation, on-line help and on-line training is really an abstraction that is made based on the main research emphasis.

6 Design and development of on-line assistance

This final discussion describes our attempts in designing on-line assistance for a complex interactive computer system. In the Norwegian Research Department's Telemedicine project Ratatosk, we were challenged to address usability through on-line assistance. After a brief description of the Ratatosk project, three methodologies for usability engineering, help system implementation, and knowledge engineering are given. These methodologies provided guidelines and support for our activities in the Ratatosk project. In the concluding section we share our experiences in the predesign phase, where we first identified the characteristics of the various types of on-line assistance, and then, using a description of our target user population, selected one approach for initial focus—the guided task paradigm. It is hoped that these experiences in the predesign phase are of use to others wishing to incorporate on-line assistance into their knowledge based systems.

6.1 Ratatosk project

The Ratatosk project is concerned with adaptative user interface research for computer-mediated communication (CMC). CMC systems provide user access to services such as email and conferencing systems. Often the services offered have differences in functionality, and in interpretation and usage of the most basic concepts of communication (Danielsen et al., 1990b). A solution for avoiding the confusion caused to the user by these differences is to provide a common interface that integrates a number of communication services, thus hiding the complexities and heterogeneity of the underlying systems. Such a system is referred to as an integrated CMC system. The user is given uniform access to the services and is then faced with using one set of commands for accessing the services, and the unique commands for employing each individual service.

Ratatosk (Danielsen et al., 1990a,b), is an example of an integrated interface that provides access to existing computer-mediated communications systems. The current prototype[13], shown in

[13]The prototype is implemented in C and runs on Sun and HP workstations.

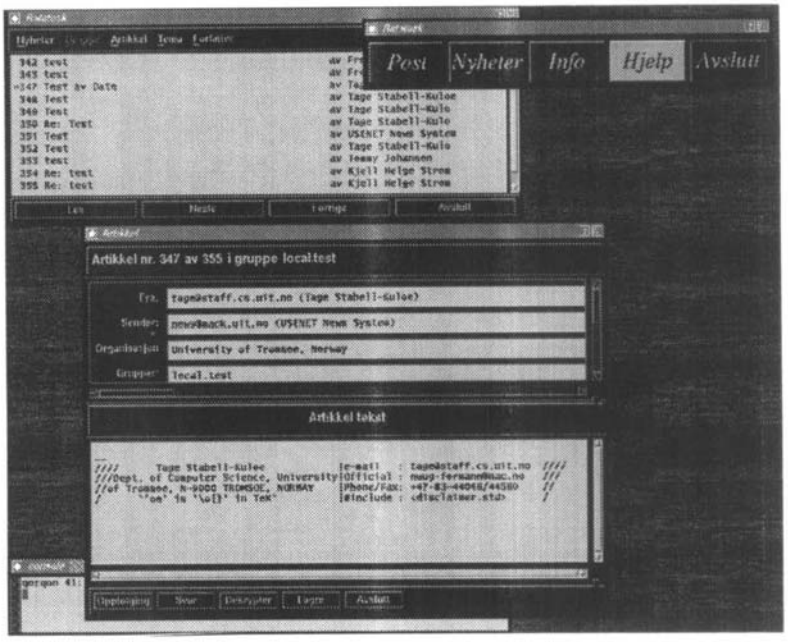


Figure 13 Snapshot of Ratatosk main menu and conference service

Figure 13, supports access to the electronic mail services mailx and EAN, and the News and PortaCom conference systems. That is, the underlying communication services are standard systems that are available individually. Ratatosk aspires to (i) have the ability to adapt itself to the level of the user, (ii) provide assistance when appropriate, and (iii) be sensitive to the different expectations, preferences and needs of individual users. It is this second aspiration that is of interest here.

The target user population, for our concerns, is the health care worker including doctors, nurses, nursing assistants, administrators, medical clerks, and office staff. This represents a rather diverse group of users, from the points of view of background, job requirements, and computer experience.

6.2 On-line assistance implementation cycle

The design and development of any software product, be it knowledge-based or not, is facilitated by tools (e.g., for prototyping), methodologies (e.g., for knowledge acquisition), and life cycles (e.g., from conception to life end of a product). This section identifies three such facilities, from which the design and development of on-line assistance for interactive computer systems can benefit. A detailed discussion is beyond the scope of this paper, but the interested reader is directed to the given references.

6.2.1 Usability engineering

Since on-line assistance is a vital aspect of the user interface, *usability engineering models* which aim to ensure good user interfaces, should be incorporated into the development process of interactive computer systems. A usability engineering process, presented by Nielsen (1992), includes elements to be considered during the *predesign, design,* and *postdesign* stages of software development. It focuses on users, user participation in the design, and empirical user testing and prototyping combined with iterative design. The predesign stage—understanding the target user population and tasks—involves: *knowing the user* (via, among other things, visiting the work environment, performing task and functional analyses, and being aware of the evolution of the user), *performing competitive analysis* (using existing products as prototypes), and *setting usability goods* (with respect to learnability, efficiency of use, user satisfaction, frequency of errors, etc.). During the

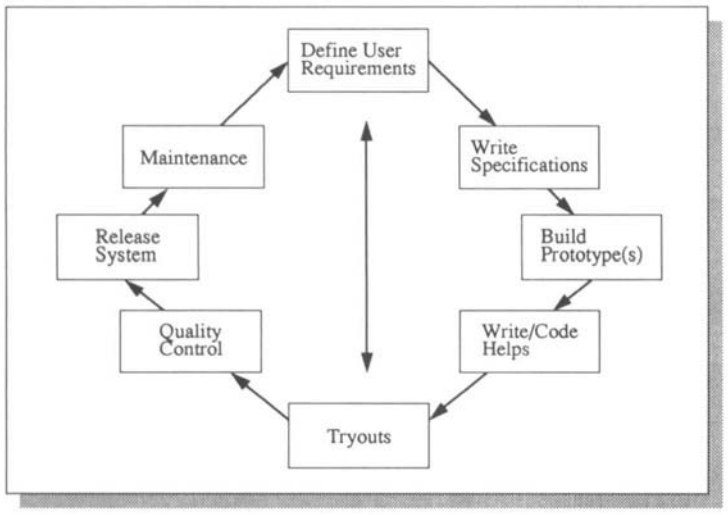


Figure 14 Help system implementation cycle (Kearsley, 1988)

design stage, the goal of which is to arrive at a usable implementation that can be released, several elements are identified including: *participatory design* (having a pool of users available from the beginning), *coordinated design* (to ensure consistency between documentation, on-line help, on-line training, application screens, etc.), *guidelines and heuristics analysis* (following well-known principles for user interface design, such as be consistent, prevent errors, provide shortcuts, etc.), *prototyping* (as early and often as possible), *empirical testing* (for *usability* via think aloud, constructive interaction, attitude questionnaires, etc., and *formative evaluation* for finding out which aspects of the user interface work), *iterative design* (use the input from the previous step to improve the design). The postdesign stage involves gathering data for the next version and for new future products.

6.2.2 Help system implementation cycle

One type of on-line assistance is on-line help, thus Kearsley's (1988) *help system implementation cycle* is of interest. Figure 14 outlines the major steps in an "idealised" implementation cycle which correspond with the development steps for any software system. Each step may comprise substeps, and many not necessarily by carried out in the order specified. In addition, each step feeds the others. Several of the steps correspond with those identified in the usability engineering process discussed above.

The first step in the implementation cycle is the *analysis of user requirements*. During this phase the needs of the user are identified. In particular, it is important to identify where and why users are going to have problems in using the system. Several data collection techniques can be used including: *task analysis* (identification of all tasks required to successfully complete a job or activity), *critical incidents* (identification of successful and unsuccessful incidents associated with a specified activity), *focus groups* (small groups of prospective users are used to give reactions to a description of the system or a prototype), *protocol analysis* (individual users are asked to describe aloud what they are doing and thinking when using a program; these are transcribed and used for analysis).

The requirements identified in the first phase are used to write a set of assistance *specifications*. These specifications include the characteristics (such as static/dynamic, educational/performance etc.) of the intended help, and where necessary, the algorithms for identifying the appropriate assistance message. Also included in this phase are specifications about issues such as where the assistance messages will appear, and how the help messages are stored internally.

During the *prototyping* phase, a mock-up of the real system is developed. The prototype does not have the full functionality of the eventual system, and does not have to look like the final version. In fact, a good point to remember is not to let the prototype be the final design. The

prototype is useful to collect preliminary feedback about a screen layout, the ease of a particular method for carrying out a task, the effectiveness of the assistance messages or other aspects of the specifications that can be tested. The data from this phase is useful in rewriting the specifications to deal with limitations identified by having real users interact with an actual system.

Once the specifications have been finalized, the actual *coding* of the assistance system can be carried out. Issues such as what programming language to use, or how to guarantee maintenance of the system are addressed. Once the actual system is developed, *tryouts* of the system with prospective users takes place. The analysis techniques used in phase one are also applicable here to identify whether the assistance is effective or not and can be used to identify missing or existing types of assistance that must be added. *Quality control* concerns a careful review of the system in use to ensure robustness and correctness (i.e., accuracy) of the assistance. It is a rare software that is ever "complete"—there are always improvements that can be made. Therefore, *releases*, versions with an agreed-upon set of features, are used. This enables further tests of the system with feedback coming from those actually using the system. The problems encountered by these actual users can be addressed in the next release. It is also important to keep the assistance up to date—to plan for some way to provide system *maintenance*. This is an on-going phase that must take place after the development is complete.

6.2.3 Knowledge engineering

As identified in earlier sections, the development of contemporary on-line assistance requires the modelling of several domains (e.g., the application, the tasks), thus *knowledge engineering tools* (e.g., the HSDS-tools (Breuker, 1990) for constructing application models) and *methodologies* (e.g., the KADS knowledge acquisition methodology (Breuker et al., 1989)) which facilitate the design and development of knowledge-based systems, can be applied to the design and development of on-line assistance. Breuker (1990) shares the lessons learned from EUROHELP's *life cycle* for help system development. He describes the life cycle, after an initial stage of prototyping, reading literature, and performing experiments, as following the traditional software engineering stages of analysis, design, implementation, and evaluation. One departure from traditional engineering approaches is the separation of the application model (AM) from the intelligent help system shell (IHSS). The AM is the representation of the domain knowledge (see the domain representation in Figure 12) as *operational knowledge* that describes how to use the application, and *support knowledge* that describes the underlying concepts. Thus, a major contribution of EUROHELP is a methodology that provides guidelines for the developers of AMs for existing information processing system (IPS) applications. There are three phases: *user and task analyses* (identification and characterization of the target user population), the *development of the conceptual map* (list of names, groupings, relationships, etc., to be manipulated by the IPS, and the operations to be performed), and the *construction of the application model*. Furthermore, a set of help system development tools (HSDS-tools), such as hierarchic browsers, frame editors, storage and access mechanisms, have been developed and specifically tailored to the construction of AMs.

6.2.4 Main points for consideration

The main point of interest gleaned from the three cited methodologies is that each begins with a detailed look at the user and the tasks the user is to perform, and each makes extensive use of prototypes. A useful technique in understanding the user's tasks is to use a task analysis approach such as the one proposed by Johnson et al. (1988). A task analysis provides a behavioural description of the activities in which users engage. They propose the following as having potential roles in a task analysis:

- analysis of texts of manuals
- direct observation of task performer
- analyst performing task
- structured interview with task performer

- questionnaires
- concurrent or retrospective protocol
- frequency counting
- rating scales
- sorting
- pilot study
- tutorial sessions.

Developers of on-line assistance systems need to incorporate these into their development processes as Kearsley (1990, pp. 72–73) so aptly argues, "making design decisions about help systems involves balancing the anticipated effectiveness of the helps against the labour costs of implementing the design . . . . The only way to determine the right level is to build prototypes, run tryouts, and collect effectiveness data . . . . Designers of help systems should be prepared to spend a magnificent amount of time observing and collecting data on users if they want to do their job well!".

6.3 Designing assistance for Ratatosk

Providing on-line assistance for users of an integrated CMC system such as Ratatosk raises special challenges. In addition to giving the assistance required to support the use of each individual service, there is a need to supply assistance for selecting and accessing the correct service to carry out the desired task. For example, what types of communication are best for email, for a conferencing system or a bulletin board? The novice user, especially, will need training in identifying tasks they can carry out using each service, guidance in selecting the most appropriate service, and help in exploiting the functionality of the selected system. Expert users need facilities to remind themselves of things they have forgotten and help to expand their repertoire to include more effective and efficient skills. This challenge can be summarized as *to provide the appropriate type and level of assistance to aid each user in meeting their individual goals, plans and intentions*.

To meet these challenges, we needed a clear understanding of what features contemporary on-line assistance systems had to offer. Table 3 summarizes the characteristics of assistance that can be used to describe each of the systems reviewed in the previous three sections. Although only one system, EUROHELP, provides assistance for a (single) communication service (UNIX mail), the techniques and issues identified by the other systems can provide insight into the different types of assistance required for assorted user groups and diverse activities. For example, if the goal is to provide educational assistance as well as performance assistance, then the approaches used in the critiquing, guided task and EUROHELP paradigms should be examined. Similarly, if the system to be built is to provide context-sensitive assistance, then the design and implementation of the critiquing and EUROHELP systems should be studied. It has been argued that *on-line documentation* is best for expert users who wish to have a means of reminding themselves of how particular commands work or to search for commands corresponding to a function they want to perform. A *training* system like the guided task approach, on the other hand, seems best suited to the novice user who needs to be trained in using a new application, performing new tasks, and learning about available functionality. *Help* systems can be useful to all user groups, especially if the ability to tailor the advice to the individual user and particular task has been included. By watching the user perform their task, and inferring the goals and plans of the user, the help system can adjust the level and type of help provided.

Therefore, for a system such as Ratatosk it is necessary to provide a number of different forms of assistance. For instance, on-line documentation is useful to describe the underlying services provided and the syntax and format of commands used to assess the services. On-line help can be used to help the user make more efficient use of the individual services. Guidance on how to perform particular tasks using the best service available can be provided by an on-line training system.

Table 3 Characteristics of reviewed assistance systems

	Dimension							
	Goal		Instigation		Capability		Feedback	
System	Performance	Educational	Active	Passive	Problem detecting	Solution generating	Negative	Positive
UNIX man	X			X	X		X	
DEC VM Shelp	X			X	X		X	
WIZARD	X		X		X		X	
ACTIVIST	X		X		X		X	
Critiquing	X	X	X	X	X	X	X	X
Guided Task	X	X		X	X	X	X	
EUROHELP	X	X	X	X	X	X	X	X

	Dimension							
	Context			Flexibility		Delivery		
		Dynamic						
System	Static	Context-sensitive	Dialogue	Adaptable	Adaptive	Textual	Examples	Demonstrations
UNIX man	X					X		
DEC VMS help	X					X		
WIZARD	X					X		
ACTIVIST	X					X		
Critiquing	X	X			X	X	X	X
Guided Task	X				X	X		X
EUROHELP	X	X	X		X	X		

As the previous subsection highlighted, to identify the on-line assistance characteristics required and the approach to be taken, it is necessary to understand the targeted user population. A detailed investigation into our potential user group—health care workers—must be carried out to identify their background, experience, expectations, the services they will be given, the tasks they will be required to carry out and the potential problems they will encounter. Once the user population is understood, it is then possible to be more specific about the type(s) of assistance required by the users. At this point in the predesign, however, we have preliminary report (Engum, 1991; Eidsvik, 1991) that our user population will consist of many novice users for whom access to electronic communication will be a novelty. For this reason, a prototype of a *guided task-like training system* that will assist the user in accessing and using the services provided by Ratatosk is being designed and implemented[14].

Using basic ideas from the guided task paradigm described by Tuck et al. (1990) and the Apple System 7 *Networking Basics Tour*, we envisage the novice user being guided through such tasks as gaining access to a conferencing system, choosing a news group and selecting a particular article within that group to read. Enhancing the training system to include a curriculum for learning about the system and a planner (e.g., the PEPE methodology from Wasson (1990)) to help the user determine how to go through the curriculum, is a research possibility to explore. The approach we have chosen corresponds to the early prototyping and experimentation approach used in the

[14]The prototype implementation is using KnowledgePro Windows (Knowledge Garden Inc.) on top of WINIX (1991) which provides mail and conferencing functionalities.

EUROHELP life cycle, and the recommendation for early and often prototyping given in Nielsen's usability engineering model.

By using the description of our user population and beginning to test approaches to on-line assistance, insight into future designs of Ratatosk on-line assistance will be provided at an early stage. The prototype will also enable us to be more specific in our understanding of our user population, and the tasks they will perform. Our situation is rather unique in that we are going to be offering services to a user population, the majority of which, does not currently have access to such facilities. Therefore, a detailed task analysis is not possible at this point. We need to envisage uses of Ratatosk in the healthcare sector.

Parallel to the development of a guided-task like prototype, we have proposed several informative studies of the conceptual design of a guided task like paradigm, that can quickly be designed and then tested using a representative sample of potential Ratatosk users. These include:

- Using the System 7 *Networking Basics Tour*, run sample users through the demonstration getting their feedback on issues such as: ease/difficulty of use, system response to incorrect user behaviour, and content of the messages to the user.
- Using the System 7 *MacIntosh Basics*[15] demonstration to compare some aspects of its design of assistance to that in the System 7 *Networking Basics Tour* (e.g., the way in which what to do next is demonstrated).
- Using *MacroMindDirector*[16] for the MacIntosh, rapidly prototype some aspects of guided-task assistance for Ratatosk to test with potential users.

Tyler Blake, in a workshop on *Advanced Methods for User Interface Design* at the Conference on Human Factors in Computing Systems (CHI90), pointed out that rapid prototyping of a system is for testing user experience (i.e., their behaviour and responses) and *not* programs (i.e., the computer code). He recommends prototyping at the conceptual level before the actual system is delivered. The prototype should present representative scenarios of (in our case) guidance on a particular task to a representative population. Furthermore, he suggests prototyping aspects of the final system on a platform which is different from the platform on which the final deliverable system will be implemented. By using a different platform, the users (and the designers) are prevented from developing preconceived notions of what the final system will look like. This will enable them to focus on the conceptual design—for example, what is liked and disliked about the particular design of the guidance or whether one behaviour is preferred over another, or is more effective—and allowing redesign to reflect the feedback.

By engaging in this second activity, we satisfy both Blake's recommendation for prototyping on a different platform (our target population will be using PCs), and the usability engineering method of competitive analysis (see previous subsection).

7 Summary and conclusions

The increasing complexity of interactive computer systems requires sophisticated on-line assistance that will empower the user to exploit the system functionality to the fullest. Users of today's systems are demanding. They are interested in using systems to perform their required tasks and do not want to spend time searching for better ways to do things. There is a wide variety of users differing in their use of computers, specific applications and particular tasks. Several lines of research address the issues involved in providing effective on-line assistance. These range from empirical studies of the human–human advising process to the engineering of complex systems that include some form of on-line assistance. This report has abstracted three categories of assistance to organize the discussion of issues: *on-line documentation*, *on-line help*, and *on-line training*.

[15]This is an interactive demonstration of the basics of the MacIntosh interface (e.g., trash can, scrollbars) developed by MacroMind Inc.

[16]This is a product of MacroMind Inc.

On-line documentation is one of the earliest and most common forms of assistance. The user is given access to textual descriptions of system features and commands. Empirical research (cited in section 2) reports that such assistance is useful for experienced users who need explanations of terms, descriptions of commands and function keys, or are searching for related concepts. Novice users, however, do not possess a good framework for effective exploitation of this type of assistance—they often do not recognize that they need help and even if they do, they do not know where to begin to search for help.

On-line help attempts to recognize when a user is performing ineffectively and provide help tailored to the particular situation and the individual user. Providing such assistance requires more computational effort than providing static on-line text, but results in more efficient and effective use of system resources. To help the user in exploiting the functionality of a system, it is necessary to recognize the user's goals, plans and intentions. This requires representation of large amounts of knowledge about specific domains, the user, communication, common problems and instructional strategies. Novice users, who do not always recognize when they need help, or what they need help about, are assisted in becoming more effective users of the system. Expert users can use the help to become more efficient.

The goal of on-line training systems is educational—the user is trained to use a system, to perform particular tasks, or is supported in learning about system concepts. The focus is on assisting the user in understanding, at a conceptual level, why the assistance has been given. Novice users will benefit from on-line training when interfacing with a new application or performing a new task. Expert users can also use the training to familiarize themselves with a new application or task. It is the length of time used for training and the amount and type of feedback that needs to be tailored to the user's experience level.

The on-line documentation systems presented in section 3 are standard programs released with the operating systems for which they were developed, thus are in wide-spread use. The on-line help and training systems reviewed in sections 4 and 5 respectively, were developed as research vehicles for investigations of particular aspects of the advising process[17]. Thus they are not being used except under experimental conditions.

The usability of complex systems is enhanced by the provision of on-line assistance. In light of the "production paradox" (i.e., that users want to spent their time using, not learning about systems (Carroll et al., 1986)), providing tailored support should make the use of these complex systems more attractive. The literature also supports the claim that assisting users as they perform their tasks will increase their effectiveness and efficiency. However, what must be kept in mind when giving on-line assistance, is that the user is to be "satisfied" with using the system to perform their tasks. This implies that the user not be "over burdened" or "pushed" to conform to predefined approaches (by the designers) of using the system.

Although these benefits can be used to argue for the inclusion of on-line assistance in complex systems, the cost of providing such support should be considered. In particular, the amount and type of assistance must be weighed against the effort required to provide it. For example, providing passive, static, text-based assistance such as on-line documentation is relatively quick, easy and reliable (Fischer et al., 1985). The move to giving active, context-sensitive help or training is a big step. It requires the use of techniques already employed in the development of knowledge-based systems such as knowledge representation, user modelling, and instructional planning to provide assistance that is tailored to the individual user.

There are still many empirical research questions about assistance that need to be investigated. For example, what types of assistance are best for what type of user? What kind of advisory strategies are there? Under what conditions are different strategies effective?. How should the assistance be accessed and displayed? Thus, there is a need for research focused both on empirical studies of the nature of assistance and how to build systems based on these results. It is because of these research questions that the development of on-line assistance is still in its infancy.

[17] Apple's System 7 *Networking Basics Tour*, mentioned in section 5, is an exception.

Acknowledgments

The authors would like to extend thanks to Thore Danielsen, Svein-Ivar Lillehaug and Trine Folkow for their useful comments on earlier drafts of this paper, and to Roar Steen for the snapshot of the Ratatosk screen shown in section 6. The authors also wish to thank *The Knowledge Engineering Review* referees for their very helpful comments and suggestions.

References

Akselsen, S, 1991. "Mediating knowledge. Case study: A computer-based learning environment for education and training of fish-health personnel" *Dr. Scient. Thesis,* University of Tromsø, Norway.

Aaronson, AP and Carroll JM, 1987. "The answer is in the question: a protocol study of intelligent help" *J. Behaviour and Information Technology* **6** 393–402.

Breuker, J (ed), 1990. "EUROHELP: Developing intelligent help systems" *Final report on the P280 ESPRIT Project EUROHELP*, Kopenhagen, Amsterdam/Manchester, Leeds, EC.

Breuker, J, 1988. "Coaching in help systems" In: J Self (ed) *Artificial Intelligence and Human Learning* 310–337, Chapman and Hall.

Breuker, J, Winkels, R and Sandberg, J, 1987. "A shell for intelligent help systems" *Proc. 10th Int. Joint Conference on Artificial Intelligence* **1** 167–173.

Carr, B and Goldstein, IP, 1977. "Overlays: a theory of modelling for computer-aided instruction" *AI Lab Memo 406* MIT.

Carroll, JM and Aaronson, AP, 1988. "Learning by doing with simulated intelligent help" *Communications of the ACM* **31** 1064–1079.

Carroll, JM and McKendree, J, 1987. "Interface design issues for advice-giving expert systems" *Communications of the ACM* **30** 14–31.

Carroll, JM and Rosson, MB, 1986. "Paradox of the active user" In: JM Carroll (ed) *Interfacing Thought: Cognitive aspects of human computer interaction* 80–111, Bradford/MIT Press.

Conklin, J, 1988. "Hypertext: an introduction and survey" In: I Grief (ed) *Computer-supported cooperative work: a book of readings* 423–475, Morgan Kaufmann.

Danielsen, T, Finseth, W, Flægstad, F, Hartvigsen, G and Steen, R, 1990a. "Ratatosk—et adaptivt brukergrensesnitt for datamaskinbasert kommuinikasjon" *TF-report R35/90*, Teledirektoratets forskningsavdeling, Kjeller, Norway (in Norwegian).

Danielsen, T, Finseth, W, Flægstad, F, Hartvigsen, G and Steen, R, 1990b. "Ratatosk—an adaptive user interface to computer-mediated communication" In: *Message Handling Systems and Application Layer Communication Protocols Proceedings* 345–354. (Also available as TF-lecture F16/90.)

DEC, 1978. *Command Language User's Guide*, Digital Equipment Corporation.

Eidsvik, AK, 1991. "Medisinske informasjonsdatabaser: et forsøk med søking i Medline" *TF-report R30/91*, Teledirektoratets forskningsavdeling (Norwegian Telecom Research), Kjeller, Norway (in Norwegian).

Engum, B, 1991. "MEDIS—medisinsk informasjonsservice" *TF-report R22/91*, Teledirektoratets forskningsavdeling (Norwegian Telecom Research), Kjeller, Norway (in Norwegian).

Finin, TW, 1983. "Providing help and advice in task oriented systems" In: *Proc. 8th Int. Joint Conference on Artificial Intelligence* 176–178.

Fischer, G, 1988. "Enhancing incremental learning processes with knowledge-based systems" In: H Mandl and A Lesgold (eds) *Learning Issues for Intelligent Tutoring Systems* 138–163, Springer-Verlag.

Fischer, G, 1987. "A critic for LISP" In: *Proc. 10th Int. Joint Conference on Artificial Intelligence* 177–184.

Fischer, G, Lemke, AC, Mastaglio, T and Morch AI, 1991. "The role of critiquing in cooperative problem solving" *ACM Transactions on Information Systems* **19** 123–151.

Fischer, G, Lemke, AC and Mastaglio, T, 1990. "Using critics to empower users" In: *CHI'90 Conference Proceedings* 337–347, ACM.

Fischer, G, Lemke, A and Schwab, T, 1985. "Knowledge-based help systems" In: *CHI'85 Conference Proceedings* 161–167, ACM.

Fischer, G, McCall, R and Morch, A, 1989. "JANUS: integrating hypertext with a knowledge-based design environment" In: *Hypertext '89 Proceedings* 105–117.

Gwei, GM and Foxley, E, 1990. "Towards a consultative on-line help system" *Int. J. Man-Machine Studies* **32** 363–383.

Harley, JR and Smith MJ, 1988. "Question answering and explanation giving in on-line help systems" In: J Self (ed) *Artificial Intelligence and Human Learning* 338–360, Chapman and Hall.

Houghton, RC, 1984. "Online help systems: a conspectus" *Communications of the ACM* **27** 126–133.

Johnson, P, Diaper, D and Long, J, 1984. "Task, skills and knowledge: task analysis for knowledge based descriptions" In: B Shackel (ed) *Interact84* North Holland.

Jones, J and Virvou, M, 1991. "User modelling and advice giving in intelligent help systems for UNIX" *Information and Software Technology* **33** 121–133.
Journal of User Modelling and User-Adapted Interaction, 1991, (1, 2).
Kearsley, G, 1988. *Online Help Systems: Design and Implementation* Ablex.
Kidd, AL, 1985. "What do users ask?—some thoughts on diagnostic advice" In: *Proc. 5th Technical Conference of the British Computer Society Specialist Group on Expert Systems* 9-19, CUP.
Kok, AJ, 1991. "A review and synthesis of user modelling in intelligent systems" *The Knowledge Engineering Review* **6**(1) 21–47.
Lemke, AC, 1989. "Design environments for high-functionality computer systems" Ph.D thesis, University of Colorado.
McKendree, J and Carroll, JM, 1986. "Advising roles of a computer consultant" In: *CHI'86 Conference Proceedings* 35–40, ACM.
Nielsen, J, 1992. "The usability engineering life cycle" *Computer* **25**(3) 12–22.
Olsen, DR, 1986. "MIKE: the menu interaction control environment" *ACM Transactions on Graphics* **5**(4).
Pollack, ME, 1985. "Information sought and information provided: an empirical study of user/expert dialogues" In: *CHI'85 Conference Proceedings* 155–159, ACM.
Rada, R and Barlow, J, 1988. "Expert systems and hypertext" *The Knowledge Engineering Review* 285–301.
Rissland, EL, 1984. "Ingredients of intelligent user interfaces" *Int. J. Man-Machine Studies* **21** 377–388.
Silverman, BG, 1992. "Survey of expert critiquing systems: practical and theoretical frontiers" *Communications of the ACM* **35**(4) 106–127.
Stehouwer, M and van Bruggen, J, 1986. "Performance interpretation in an intelligent help system" In: *Proc. 6th Annual ESPRIT Conference* 248–257.
Sullivan, JW and Tyler, SW, 1991. "Foreword" In: JW Sullivan and WT Sherman (eds) *Intelligent User Interfaces* vii–viii, Addison-Wesley.
Tuck, R and Olsen, DR, 1990. "Help by guided tasks; utilizing UIMS knowledge" In: *CHI'90 Conference Proceedings* 71–78, ACM.
UNIX, 1987. *Programmer's Manual,* Bell Laboratories.
Wasson, BJ 1990. "Determining the focus of instruction: content planning for intelligent tutoring systems" *Ph.D thesis*, University of Saskatchewan. (Also available as ARIES Research Report 90-5.)
WINIX, 1991. *System description version 1.10a for MS-DOS and UNIX system V operating systems* The Royal Norwegian Ministry of Education, Research and Church Affairs, Oslo, Norway.

Further reading

Advice-giving and Intelligent Help

- Carroll, JM and McKendree, J, 1987. "Interface design issues for advice-giving expert systems" *Communications of the ACM* **30** 14–31.

This foundational paper discusses the knowledge requirements (e.g., general skills, domain knowledge, user models and knowledge bounds) for advice-giving systems and elaborates on advice contingencies and content (e.g., initiative, consequences and scope). Further, it taxonomizes some of the key issues in the design of advice-giving systems. The authors conclude the paper by giving some guidelines for managing the research agenda in this field. They claim that two items need to be promoted: a psychological theory of advice and advising; and a behavioural methodology to assess the actual success of demonstration advice-giving systems.

- Hartley, JR and Smith MJ, 1988. "Question answering and explanation giving in on-line help systems" In: J Self (ed) *Artificial Intelligence and Human Learning* 338–360, Chapman and Hall.

This chapter first reviews previous work in the area of question and explanation giving as relevant for intelligent on-line help programs. Then a methodology for question interpretation and question answering which takes account of the user's intentions, working context, and levels of knowledge is considered. The proposed techniques are illustrated with examples from the EUROHELP (see next reference) prototype intelligent help system for UNIX-Mail.

- Breuker, J (ed), 1990. *EUROHELP: Developing Intelligent Help Systems* Kopenhagen, Amsterdam/Manchester, Leeds, EC.

This is the concluding report from EUROHELP, one of the largest undertakings in intelligent help system research. The project included five institutions and 115 manyears, and lasted from 1984 to 1990. This report provides a comprehensive reference for research in problems of user applications, and in building adequate help facilities. One of the main contributions of the project is the well documented development methodology used for designing, and developing tools and methods for building Intelligent Help Systems.

Critiquing

- Silverman, BG, 1992. "Survey of expert critiquing systems: practical and theoretical frontiers" *Communications of the ACM* **35** 106–127.

A "critic" refers to a computer program that criticizes user-generated solutions. This paper overviews a proposed model of the critiquing process and describes how critiquing works. Further, it surveys five categories of critiquing system usage and gives an illustrative example of the use of critics. It concludes with some reflections on future work.

- Fischer, G, Lemke, AC Mastaglio, T. and Morch, AI, 1991. "The role of critiquing in cooperative problem solving" *ACM Transactions on Information Systems* **19** 123–151.

This paper presents critiquing as a major activity of a cooperative problem-solving system. The critiquing approach is described in detail, and descriptions of some of the *Colorado critics* are presented. This paper is a good overview of the research being carried out at the University of Colorado.

Intelligent user interfaces

- Rissland, EL, 1984. "Ingredients of intelligent user interfaces" *Int. J. Man-Machine Studies* **21** 377–388.

This is one of the first papers that discusses general features of intelligent interfaces. These features include the sources of knowledge needed by an interface to be considered intelligent, and the characteristics that are desirable in an interface. The ideas put forward in the paper are illustrated by an examination of two examples of interfacing between a user and a system: on-line help and tutoring. The paper concludes with a brief survey of some of the challenges to designers of interfaces.

- Sullivan, JW, and Tyler, SW (eds), 1991. *Intelligent User Interfaces* Addison-Wesley.

This book presents an excellent collection of chapters that explore the state of the art in intelligent user interfaces research. The chapters are a subset of papers presented at a 1988 Monterey, California workshop on *Architectures for Intelligent Interfaces: Elements and Prototypes*. The chapters, which have been peer reviewed, incorporate the authors' latest results and provide a more detailed description of their work. Nineteen chapters, divided into four sections entitled (I) Multimodal Communication, (II) Models, Plans, and Goals, (III) Dynamic Presentation Design, and (IV) Knowledge-Based Tools for Interface Design, cover topics from user and discourse models, general user modelling, architectures for knowledge-based graphical interfaces, blackboard implementations, and intelligent user interface design environments.

Intelligent tutoring systems

- Wenger, E, 1987. *Artificial intelligence and tutoring systems: Computational and cognitive approaches to the communication of knowledge* Morgan Kaufmann.

This comprehensive textbook is essential reference and introduction for anyone involved in the design or development of intelligent tutoring systems. The author has firmly grounded the subject within its background of educational theory and the philosophy of knowledge, as well as providing numerous supporting examples for his theoretical opinions drawn from the field of ITS practise. There is a comprehensive bibliography, a well as author and subject indexes. (Review by Wood and Holt (1990)—see bibliographies below.)

- Polson, MC and Richardson, JJ (eds), 1988. *Foundations of Intelligent Tutoring Systems* Lawrence Erlbaum.

This book presents a synthesis of the field of Intelligent Tutoring Systems through a coordinated set of essays, written by leading researchers in the field. Through the nine chapters, the anatomy of ITSs is examinated and two things identified: the achievable ITS capabilities for the near term; and, the fundamental research questions that must be answered along the way towards more robust and effective, knowledge-based educational systems. The chapters address each of the ITS anatomy components as well as implementation and evaluation issues.

On-line documentation and hypertext

- Halasz, FG, 1987. "Reflections on NoteCards: Seven issues for the next generation of hypermedia systems" *Communications of the ACM* **31** 836–852.

This paper presents NoteCards (developed at Xerox PARC) as a foil against which to explore some of the major limitations of the late 1980's generation of hypermedia systems, and characterizes the issues that must be addressed in designing the next generation systems. These issues include: search and query in a hypermedia network, composites—augmenting the basic node and link model, virtual structures for dealing with changing information, computation in (over) hypermedia networks, versioning, support for collaborative work and extendibility and tailorability. The integration of hypermedia and artificial intelligence techniques are also discussed.

- Conklin, J, 1988. "Hypertext: An introduction and survey" In: I Grief (ed) *Computer-Supported Cooperative Work: A book of readings* 423–475, Morgan Kaufmann.

This paper gives a thorough definition of the various elements included in a hypertext system and discusses hypertext implementation. Four broad application areas for which hypertext systems have been developed are identified. Further, an extensive survey of various systems, beginning with Bush's Memex system (put forward in 1945) through Nelson's Xanadu project, the Emacs INFO Subsystem, to Brown University's Intermedia. This paper is a must read for those wanting an introduction to the area of hypertext.

Bibliographies

- Wood, PH and Holt, PD, 1990. "Intelligent Tutoring Systems: An Annotated Bibliography" *SIGART Bulletin* **1**(1) 21–42.
- Woods, DD, Johannesen, L and Potter, SS, 1991. "Human Interaction with Intelligent Systems: An Overview and Bibliography" *SIGART Bulletin* 215) 39–50.