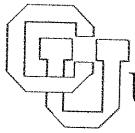
Making Computers More Useful and More Usable *

Gerard Fischer

CU-CS-364-87



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

* This research was supported by: Grant No. N00014-85-K-0842 from the Office of Naval Research and Grant No. MDA903-86-C0143 from the Army Research Institute.

Making Computers More Useful and More Usable

Gerhard Fischer

CS-CU-364-87 June 1987

Department of Computer Science
Campus Box 430
University of Colorado,
Boulder, Colorado, 80309

This research was supported by: Grant No. N00014-85-K-0842 from the Office of Naval Research and Grant No. MDA903-86-C0143 from the Army Research Institute.

MAKING COMPUTERS MORE USEFUL AND MORE USABLE

GERHARD FISCHER
Department of Computer Science and Institute of Cognitive Science
University of Colorado, Boulder

Abstract

Useful computers which are not usable are of little help; but so are usable computers which are not useful. One of the major goals of our research is to achieve these two goals simultaneously. We claim that useful computers have to be complex systems with a rich functionality. To make these systems usable and to exploit their power, we have constructed a variety of computer-based intelligent support systems which take advantage of the interactive and dynamic nature of computer systems.

Information and knowledge embedded in computers can be represented and used in qualitatively different ways than on paper. Paper is passive and can only serve as a repository for information, whereas computer systems can be active and assist us in searching, understanding and creating knowledge in the course of cooperative problem solving processes.

1. Introduction

For system to be *useful* (i.e. applicable to a wide range of problems by providing suitable abstractions), they have to offer a rich functionality (examples are systems like UNIX and LISP machines; see Figure 2-1). But systems with a rich functionality are in danger of becoming unusable. On the other hand, *usable* systems (e.g. like the MACINTOSH or the PINBALL CONSTRUCTION KIT [Fischer, Lemke 87]) are limited in their usefulness by their limited applicability and extensibility. To resolve this design trade-off is one of the major goals of our research.

Just as we need large libraries, we need complex computer systems. Although there is much leverage to be gained of thinking of online information systems as "books" or "libraries", this metaphor can overshadow the dynamic nature of the information and its interactive uses. In this paper, we will first illustrate our belief why there will always be a need for complex systems. Our answer towards mastering these complex systems is the use of *computer-based intelligent support systems*, which help the user asking the right questions, finding information, using it, understanding it, modifying it and personalizing it.

2. Reality is not User-Friendly -- or: There will always be a Need for Complex Systems

Modern computer systems are best understood not in their capacity to compute but to serve as *knowledge stores*. And because there is lots of knowledge around, we should not expect that the systems will be small and simple, but hey will be large and complex. The dimensions of some prototypical systems are given in Figure 2-1.

Systems offering such a rich functionality are a mixed blessing: in a very large knowledge store it is much more likely to find something related to what we need, but it is also much more difficult to find something specific. Our empirical investigations indicate that the following problems prevent many users

Number of Computational Objects in Systems

EMACS:

- 170 function keys and 462 commands

UNIX:

- more than 700 commands and a large number of embedded systems

LISP-Systems:

- FRANZ-LISP: 685 functions
- WLISP: 2590 LISP functions and 200 ObjTalk classes
- SYMBOLICS LISP MACHINES: 19000 functions and 2300 flavors

Amount of Written Documentation

Symbolics LISP Machines:

- 10 Books with 3000 Pages
- does not include any application programs

SUN workstations:

- 15 books with 4600 pages
- additional Beginner's Guides: 8 books totaling 800 pages

Figure 2-1: Quantitative Analysis of Some Systems

and designers from successfully exploiting the potential of these high-functionality systems, because

- they do not know about the *existence* of tools;
- they do not know how to *access* tools;
- they do not know *when* to use these tools;
- they do not understand the *results* that tools produce for them;
- they cannot combine, adapt, and modify a tool to their *specific* needs.

A consequence of these problems is that these systems are underused and the broad functionality is of little value. We are strongly convinced that what is needed is *not quantitatively* more information but *qualitatively* new ways to structure and present information.

3. Intelligent Support Systems

We claim that in future computer systems, a high percentage of the computational power should be used to make these systems usable, manageable and useful. Our system-building efforts are based on the belief that *the intelligence of a complex computer system must contribute to its ease of use* and that communication between humans and computers cannot be restricted to nice pictures on the screen (the beauty of the interface should not overshadow the limited functionality and extensibility of a system). Truly intelligent and knowledgeable human communicators, such as good teachers, use a substantial part of their knowledge to explain their expertise to others. In the same way, the intelligence of a computer should be applied to providing effective communication.

3.1 Shortcomings of Support Systems on Paper

One of the most neglected questions in our information society is: how to produce effective materials and reference information to describe the operation of complex devices. Reading and understanding the operation manual for a video-recorder is far from trivial, despite the fact it may be only 20 pages long. So what can we expect from system descriptions which occupy 3000 to 5000 pages (see Figure 2-1)?

Making complex systems useful and usable is not only limited by our inability to produce effective descriptions, but there are inherent limitations in doing this with paper. In the case of program documentation systems, we have argued [Fischer, Schneider 84] that program documentation produced as a separate document by a word processing system has the following disadvantages:

- it is impossible to provide pieces of information automatically,
- it is impossible to maintain consistency between the program and its documentation automatically (or at least semi-automatically),
- it is impossible to generate different external views dynamically from one complex internal structure (e.g., to read a documentation either as a primer or as a reference manual),
- it is impossible to create links between the static description and the dynamic behavior.

Similar arguments apply to *form systems* where paper representations also have major shortcomings. In forms on paper, *all* cases (e.g. married or not married, employed or unemployed, citizen or foreigner) must be explicitly represented and paper is incapable in propagating values from one part of a form to another one.

3.2 An Architecture for Intelligent Support Systems

We need an architecture to support a coherent design strategy that treats intelligent support systems not as add-ons to existing systems but as integral parts of a user-centered design approach. In Figure 3-1 we illustrate a system architecture that we have developed in response to this design criterion.

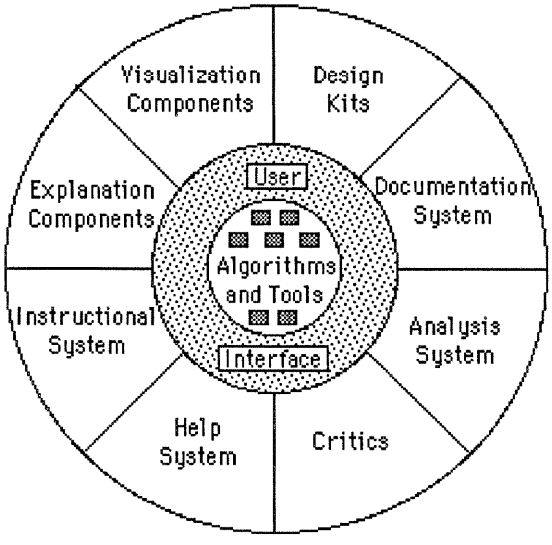


Figure 3-1: An Architecture for Intelligent Support Systems

We have constructed the following *prototypical systems* contained in the outer ring of Figure 3-1: a computer-supported documentation system [Fischer, Schneider 84], passive and active help systems [Fischer, Lemke, Schwab 85], a critic system [Fischer 87a] and several visualization tools as part of our software oscilloscope [Boecker, Fischer, Nieper 86].

3.3 The Potential of Computer-Based Support Systems

There exist support opportunities unique to interactive systems that we do not yet understand well enough how to exploit effectively. We will briefly discuss the potential of *computer-based* support systems.

Knowledge-Based Systems. We predict that in the long run, our perspective of programming as the main activity of creating computer systems will change. Designers and users will create knowledge stores, where the executable part (i.e., the code) is just one perspective of it.

In the design and implementation of our program documentation system [Fischer, Schneider 84], the knowledge base contained all of the knowledge about a system combined with a set of tools useful for *acquiring, storing, maintaining* and *using* this knowledge. The knowledge base was in part interpreted by the computer to maintain the consistency of the acquired knowledge about structural properties and it supported the users in debugging and maintaining their programs. Other parts of it were not directly interpretable by the machine which served only as a medium for structured communication between the different users.

Custom-tailored and user-centered representations. In order for a system to adapt to the users' needs, it has to incorporate information about the users' preferences, skills, and what they have previously looked at. The user should be able to communicate intentions so the system can highlight and prioritize what's important and hide connections and details which are considered irrelevant for the task at hand. The system must support mechanisms to generate *views of reduced complexity.* Filter mechanisms, where the filters can be defined by the user, allow the generation of views of reduced complexity showing only the information which is relevant for a specific user at a specific time (for an example see [Fischer, Schneider 84]).

Choosing program documentation again as an example, it has to serve different groups trying to perform different tasks. Therefore the amount and structure of the information offered to these groups of people has to be different, but all external views are generated by using different filters attached to the same knowledge base. At least the following groups should be distinguished: clients, designers, programmers and users.

Turning systems into coaches and critics. Computer-based systems have the potential to turn systems into coaches and critics while providing some of the support which is offered by face-to-face communication. They can help users ask the right, the interesting and the relevant questions (e.g. supporting a retrieval paradigm by reformulation [Williams 84]) and they can support a knowledge store in the form of an encyclopedia [Weyer, Borning 85] which allows the user to conjecture, create and experiment with information instead of just retrieving it. A prerequisite for a system to act as a coach and to present user-centered information is that the system contains a model of the user [Fischer, Lemke, Schwab 85]. It provides the necessary information to enhance incremental learning and support learning on demand [Fischer 87b].

How to find what we do not know? To ask a question, one must know enough to know what is not known! How can we find out about all the good stuff, if we don't even know to ask? Active systems (e.g., acting as critics and consultants [Fischer, Lemke, Schwab 85; Fischer 87a]) are needed which volunteer help in appropriate situations rather than responding to explicit requests. In order to volunteer information, the system needs some understanding of what the user is doing.

Dynamically generated information. A knowledge-based architecture allows information to be generated on "the fly". The visualization tools in our software oscilloscope [Boecker, Fischer, Nieper 86] all share this property; they can take the actual working context into account and visualize the structures the user is working on. This provides a qualitatively different level of support compared to "canned" visualizations stored on a video disc. Information on a video disc has to be preselected and cannot take the actual working context into account.

Situation models versus system models. Current support information is designer-oriented rather than user-oriented. Documentation and help information is structured to describe the system, not to address the problems experienced by the user. We claim that this is the main reason that human assistance, if available on a personal level, is still the most useful source of advice and help. Learners can ask a knowledgeable colleague a question in an infinite variety of ways; they get assistance in formulating the question, and they can articulate their problem in terms of the *situation model* rather than being required to do so in terms of a *system model* [Fischer, Kintsch 86].

A knowledge store trying to cover situation models must incorporate "user constructs", user-oriented organizations of knowledge and a presentation component which presents information in the user's concepts and words.

4. Making Systems more Useful and more Usable

As mentioned before, it is one of our major research goals to eliminate the design trade-off between useful and usable systems. In this section we briefly describe a few approaches from our work towards achieving this goal.

Most computer users are not interested in computers per se, but they want to use the computer to solve problems and to accomplish their tasks. To shape the computer into a truly usable and useful medium for them, we have to make it invisible and let them work *directly* on their problems and their tasks. *Human problem-domain communication* provides a new level of quality in human-computer communication, because it permits us to build the important abstract operations and objects of a given application area directly into the environment. This implies that the user can operate with personally meaningful abstractions. In most cases we do not want to eliminate the semantics of a problem domain by reducing the information to formulas in first-order logic or to general graphs. Whenever the user of a system can directly manipulate the concepts of an application, systems become more understandable and the distinction between programmers and non-programmers vanishes.

Construction and design kits [Fischer, Lemke 87] are system components that represent steps towards human problem-domain communication. A construction kit is a set of building blocks that models a problem domain. The building blocks define a design space (the set of all possible designs that can be created by combining these blocks). Design kits go beyond construction kits in that they bring to bear general knowledge about design (e.g. which meaningful artifacts can be constructed, how and which blocks can be combined with each other) that is useful for the designer. In [Fischer, Lemke 87] we describe a number of prototypical examples of systems which try to provide a foundation for human problem-domain communication.

Systems which make an attempt to model many different problem domains (which is a necessity for systems based on the human problem-domain communication paradigm) will be large and complex in order to provide all the necessary abstractions (evidence is provided by Figure 2-1).

By exploiting the possibilities of intelligent support systems, we can make these systems more usable. In addition, our previous statement "reality is not user-friendly" should not be misunderstood. There is no "conversation law on complexity" [Simon 81], which requires that the complexity and usability of a system is a given constant. Complexity can be reduced by

- exploiting what people already know (e.g., human problem-domain communication exploits this aspect);
- using familiar representations (based on previous knowledge and analogous to known situations);
- exploiting the strengths of human information processing (e.g., the power of our visual system [Boecker, Fischer, Nieper 86]);
- segmenting information into microworlds [Fischer 87b];

- eliminate learning through "better" systems.

To elaborate the last objective mentioned: in analyzing the design tradeoffs for mastering complex systems, it is crucial that we learn when to *enhance* learning and understanding and when to *reduce* learning by providing more adequate representations. The amount of expertise necessary to do a task is determined not only by the nature of the task but also by the design of the system being used to accomplish the task. We are convinced that qualitatively different ways of communication between humans and computers can greatly reduce the need for learning in many situations. Innovative systems should be explored as alternatives to support tools developed for inferior systems.

We illustrate the tradeoff between enhancing learning and reducing learning with the example shown in Figure 4-1. In almost all current interfaces to UNIX, directory trees cannot be presented externally in a two-dimensional representation which can be modified by direct manipulation. Using one of our software oscilloscope tools, we can represent the directory as a tree structure on the screen. In the traditional representation, switching back and forth between different directories requires long, error-prone command sequences using the `cd` command. An active help system like our ACTIVIST system [Fischer, Lemke, Schwab 85] could discover this kind of behavior and suggest the `pushd` and `popd` commands instead. These commands create a stack with the used directories and make the switching process less cumbersome. This suggestion can be seen as an enhancement of learning that becomes obsolete through the display and direct manipulation possibilities offered by the software oscilloscope.

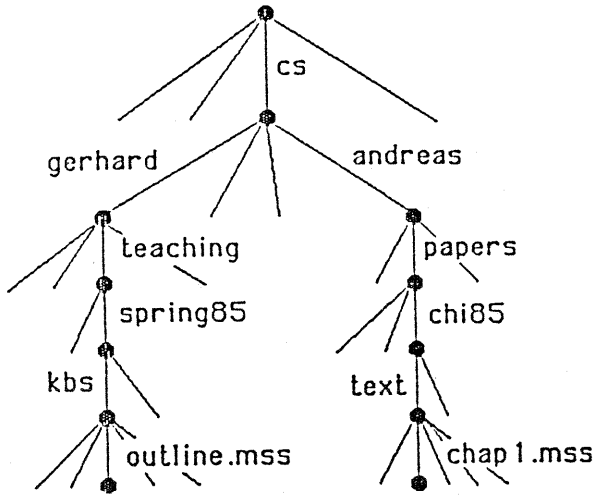
Many of the issues discussed before can contribute to the goal of reducing the cognitive effort and costs to learn and operate a complex system:

- **The software oscilloscope:** it exploits the basic human information-processing capabilities. The human visual system is very powerful. Instead of using the UNIX `pushd` command to manipulate an invisible stack, it is much easier to change working directories by pointing into a graphical display of the directory structure (see Figure 4-1).
- **Human problem-domain communication:** it explores the creation of *systematic domains* [Winograd, Flores 86], which represent objects and concepts in a way the professional is already familiar with.
- **Defaults:** they hide redundant or rarely used detail. A system should be easy to use for simple things. It should exhibit reasonable behavior when only a small part of its functionality is being used. Defaults should be used as much as possible and mechanisms should be provided to enable the user to override the defaults at will.
- **Support for user activities:** this implies structuring information in the situation model instead of the system model. Reduction of learning can be accomplished by easy and fast information retrieval. Information retrieval systems can be viewed as an extension of the human memory. A shortcoming of many existing information retrieval systems, like manuals, is that access is by *implementation unit* (e.g., LISP function, UNIX command) rather than by *application goal* on the task level.

5. Future Research

The task of science is to make use of the world's redundancy to describe that world simply [Simon 81]. We argued that there will always be a need for complex systems, but that there are unique and largely unexplored opportunities (especially in the area of intelligent support systems) to break the "conversation law of complexity" and to create systems which are useful and usable at the same time.

Computer systems of the future should be perceived and constructed as knowledge stores. The traditional view of programming should be replaced by creating a knowledge base about problem domains (see Figure 3-1). Executable programs, help, explanations, criticism and so on should all be contained or derived from this knowledge base. An architecture of this kind will also be able to personalize the factual information to the user's goals, needs and understanding [Fischer, Kintsch 86].

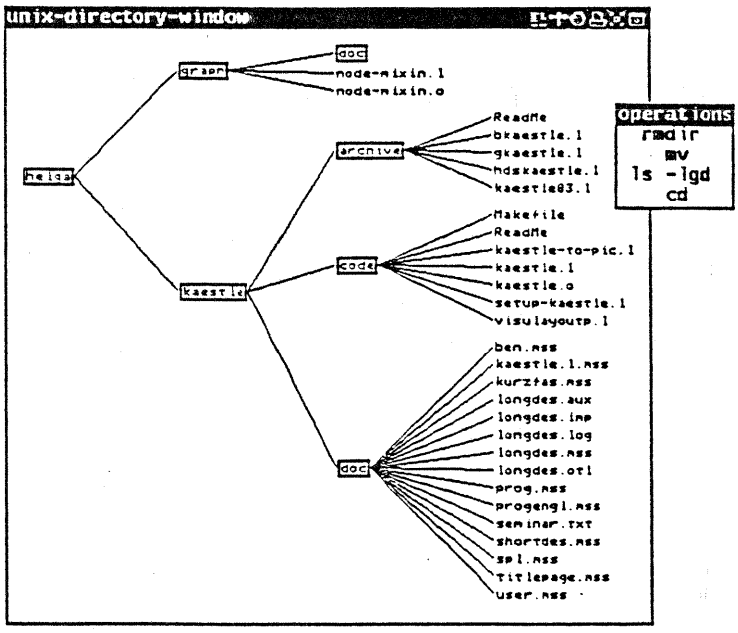



```
% cd ~gerhard/teaching/spring85/kbs

% cd ~andreas/papers/chi85/text
```

The UNIX commands to switch between directories

The file hierarchy as hand-drawn picture



The automatically generated, graphical layout structure.
File operations can be directly carried out on this structure.

Figure 4-1: Reducing Learning through New Presentations

Acknowledgements

The author would like to thank his colleagues and students at the University of Stuttgart and at the University of Colorado, Boulder who have developed the systems (described elsewhere) on which the contents of this paper is based. Without their contributions, the described research effort would not have been possible. The research was supported by grant No. N00014-85-K-0842 from the Office of Naval Research and grant No. MDA903-86-C0143 from the Army Research Institute.

References

[Boecker, Fischer, Nieper 86]
H.-D. Boecker, G. Fischer, H. Nieper, *The Enhancement of Understanding through Visual Representations*, Human Factors in Computing Systems, CHI'86 Conference Proceedings (Boston, MA), ACM, New York, April 1986, pp. 44-50.

[Fischer 87a]
G. Fischer, *A Critic for LISP*, Technical Report, University of Colorado, Boulder, 1987.

[Fischer 87b]
G. Fischer, *Enhancing Incremental Learning Processes with Knowledge-Based Systems*, in H. Mandl, A. Lesgold (eds.), *Learning Issues for Intelligent Tutoring Systems*, Springer-Verlag, Berlin - Heidelberg - New York, 1987.

[Fischer, Kintsch 86]
G. Fischer, W. Kintsch, *Theories, Methods and Tools for the Design of User-Centered Systems*, Technical Report, University of Colorado, Boulder, 1986.

[Fischer, Lemke 87]
G. Fischer, A.C. Lemke, *Design Kits: Steps Toward Human Problem-Domain Communication*, Paper Submitted to the Journal 'Human-Computer Interaction', University of Colorado, Boulder, 1987.

[Fischer, Lemke, Schwab 85]
G. Fischer, A.C. Lemke, T. Schwab, *Knowledge-Based Help Systems*, Human Factors in Computing Systems, CHI'85 Conference Proceedings (San Francisco, CA), ACM, New York, April 1985, pp. 161-167.

[Fischer, Schneider 84]
G. Fischer, M. Schneider, *Knowledge-Based Communication Processes in Software Engineering*, Proceedings of the 7th International Conference on Software Engineering, Orlando, Florida, March 1984, pp. 358-368.

[Simon 81]
H.A. Simon, *The Sciences of the Artificial*, The MIT Press, Cambridge, MA, 1981.

[Weyer, Borning 85]
S.A. Weyer, A.H. Borning, *A Prototype Electronic Encyclopedia*, ACM Transactions on Office Information Systems, Vol. 3, No. 1, January 1985, pp. 63-88.

[Williams 84]
M.D. Williams, *What Makes RABBIT Run?*, International Journal of Man-Machine Studies, Vol. 21, 1984, pp. 333-352.

[Winograd, Flores 86]
T. Winograd, F. Flores, *Understanding Computers and Cognition: A New Foundation for Design*, Ablex Publishing Corporation, Norwood, NJ, 1986.