

The Role of Critiquing in Cooperative Problem Solving

GERHARD FISCHER, ANDREAS C. LEMKE, THOMAS MASTAGLIO,
and ANDERS I. MORCH
University of Colorado, Boulder

Cooperative problem-solving systems help users design solutions themselves as opposed to having solutions designed for them. Critiquing—presenting a reasoned opinion about a user's product or action—is a major activity of a cooperative problem-solving system. Critics make the constructed artifact “talk back” to the user. Conditions under which critics are more appropriate than autonomous expert systems are discussed. Critics should be embedded in integrated design environments along with other components, such as an argumentative hypertext system, a specification component, and a catalog. Critics support learning as a by-product of problem solving. The major subprocesses of critiquing are goal acquisition, product analysis, critiquing strategies, adaptation capability, explanation and argumentation, and advisory capability. The generality of the critiquing approach is demonstrated by discussing critiquing systems developed in our group and elsewhere. Limitations of many current critics include their inability to learn about specific user goals and their intervention strategies.

Categories and Subject Descriptors: H.1.2 [Models and Principles]: User/Machine Systems—*human factors*; H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval; J.6 [Computer Applications]: Computer-Aided Engineering—*computer-aided design*; K.3.1 [Computers and Education]: Computer Uses in Education

General Terms: Design, Human Factors

Additional Key Words and Phrases: Cooperative problem-solving systems, critics, critiquing, design environments, high-functionality computer systems, intelligent support systems

1. INTRODUCTION

The critiquing approach is an effective way to use computer knowledge bases to aid users in their work and to support learning. Our experience with this approach consists of several years of innovative system building, integration

This research was partially supported by grant N00014-85-K-0842 from the Office of Naval Research, grants IRI-8722792 and IRI-9015441 from the National Science Foundation, grant MDA903-86-C0143 from the Army Research Institute, and grants from the Intelligent Interfaces Group at NYNEX and from Software Research Associates (SRA), Tokyo.

Authors' addresses: G. Fischer and A. C. Lemke, Department of Computer Science and Institute for Cognitive Science, University of Colorado at Boulder, Boulder, CO 80309. T. Mastaglio, U.S. Army TRADOC, P. O. Box 298, Fort Monroe, VA 23651; A. I. Morch, NYNEX Science and Technology, 500 Westchester Avenue, White Plains, NY 10604; email: gerhard@cs.colorado.edu, andreas@cs.colorado.edu, mastaglt%mon1@leavemh.army.mil, anders@nynexst.com.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1991 ACM 1046-8188/91/0400-0123 \$01.50

of cognitive and design theories, empirical observations and evaluation of prototypes. In this paper, we discuss the role of critiquing in cooperative problem solving. By illustrating the approach with examples from our own work and critics developed by others, we develop a general characterization of the critiquing process. We conclude with a discussion of potential future research on the critiquing paradigm.

2. THE ROLE OF CRITIQUING IN COOPERATIVE PROBLEM SOLVING

2.1 Cooperative Problem Solving

Cooperative problem solving [15, 41, 60, 63] in the context of this paper refers to cooperation between a human and a computer. To design successful cooperative problem-solving systems, issues such as what role each partner should play, when to take the initiative and how to communicate to the other partner must be resolved. These issues are shared with two related but different research areas: *Computer-Supported Cooperative Work (CSCW)* [33], which describes the cooperation between humans mediated by a computer and *Distributed Artificial Intelligence* [5], which refers to cooperation between computer systems. Cooperative problem solving requires more from a system than having a nice user interface or supporting natural language dialogs. The design of cooperative problem-solving systems must be based on a theory of problem solving that describes the functions of shared representations, mixed-initiative dialogues, argumentation and management of trouble. Cooperative problem-solving approaches exploit the *asymmetry* of the communication process. Humans use common sense, define the common goal, decompose problems into subproblems and so on. Computers provide external memory for the human, insure consistency, hide irrelevant information and summarize and visualize information.

Cooperative problem-solving systems are examples of human-computer cognitive systems [66]. They serve as cognitive amplifiers of the human. The goal of building such cooperative systems challenges the predominant goal of artificial intelligence: understanding and building autonomous, intelligent, thinking machines. Along with many other researchers [60], we believe that building cooperative problem-solving systems and interactive knowledge media is at least as important a goal as building autonomous thinking machines. The major difference between classical expert systems, such as MYCIN [6] and R1 [45], and cooperative problem-solving systems involves the roles of the human and computer. Most expert systems ask the user for input, make all decisions and then return an answer. In a cooperative problem-solving system, the user is an active agent empowered by the system's knowledge.

In this paper, we review critiquing systems in which the human generates an artifact and the computer critiques it. This is not the only role we have explored for critiquing. Alternately, computers can propose solutions and the humans subsequently critique and modify them. Examples of this latter approach are discussed by Nieper-Lemke [48] for layout of graphs and by Fischer and Stevens [29] for filters that reduce large information spaces. In both examples, the systems have algorithms for creating a first approxima-

tion of the desired artifact, which the users can then critique and modify. To generate the first approximations, the systems collect information about the graphs and the information spaces that is not necessarily known to the users.

The following aspects of cooperative problem solving are of special interest in this paper:

- *Breakdowns in cooperative problem-solving systems are not as detrimental as in expert systems.* One can never anticipate or “design away” all of the misunderstandings and problems that might arise in achieving a goal. System resources are needed to recognize and deal with the unexpected. A cooperative system needs to deal with open problems and know about the human problem solver’s intentions, which often change during problem solving.
- *Background assumptions do not need to be fully articulated.* Suchman [61] argues that background assumptions cannot be fully inventoried in any formal system. It is a strength of human experts that they know the larger problem context, which enables them to solve ill-defined problems and to learn while solving problems. This learning improves the conceptual structure of their knowledge. Experts can judge the relevance of design knowledge to design problems and they know when design rules should be broken. Expert performance degrades gracefully as they attempt to solve problems that are further removed from the core of their expertise. Current expert systems are limited in these capabilities.
- *Semiformal system architectures are appropriate.* Semiformal computer systems need not be capable of interpreting all information structures available to them. The systems deliver information to humans and humans read and interpret it. Semiformal systems can be used more extensively in cooperative systems than in expert systems and will play a large role in the design of effective joint human-computer systems.
- *Delegation problem.* Automating a task or delegating it to another person requires that the task be precisely described. Most tasks involve many background assumptions that delegators are incapable of describing. The cooperative approach eliminates the need to perfectly specify tasks. Instead, the cooperating agents incrementally evolve an understanding of the task.
- *Humans enjoy “doing” and “deciding.”* Humans often enjoy the process and not just the product; they want to take an active part. This is why they build model trains, why they plan their vacations, and why they design their own kitchens. Automation is a two-edged sword. At one extreme, it is a servant, relieving humans of the tedium of low-level operations and freeing them for higher cognitive functions. Many people do not enjoy checking documents for spelling errors, and welcome the automation provided by spelling checkers in word processors. At the other extreme, automation can reduce the status of humans to “button pushers” and strip their work of its meaning and satisfaction. People’s willingness to delegate tasks depends on the extent to which

they trust they will receive satisfactory solutions. Critics allow—and indeed force—them to exercise a great deal of personal control over, and to take responsibility for, the design of the product.

2.2 The Critiquing Approach

Critiquing is a major activity of a cooperative problem-solving system. Critiquing is the presentation of a reasoned opinion about a product or action (Figure 1).¹ The product could be a computer program, a kitchen design or a medical treatment plan; the action could be a sequence of keystrokes that corrects a mistake in a word processor document or a sequence of operating system commands. An agent—human or machine—capable of critiquing in this sense is a critic. Critics are made up of a set of rules or procedural specialists for different aspects of a product; sometimes each individual rule or specialist is referred to as a critic.

Critics do not necessarily solve problems for the user. The core task of critics is to recognize and communicate debatable issues concerning a product. Critics point out errors and suboptimal conditions that might otherwise remain undetected. Many critics also advise users on how to improve the product and explain their reasoning. Critics thus help users avoid problems and learn different views and opinions.

Characterization of domains suited for critiquing. Critics are particularly well suited for design tasks in complex problem domains. Design problems are ill-defined [58] or wicked [53]. They do not have an optimal solution and the problem cannot be precisely specified before attempting a solution. Critics can function with only a partial understanding of the task. Even if the system knows only aspects of the general problem domain, it can provide support by applying generic design knowledge.

Not all problems fit this description; there are problems in engineering design and operations research that can be precisely specified and for which optimal solutions can be found. Those types of problems yield more to algorithmic solutions and are not good candidates for the critiquing approach.

Expert systems are inadequate in situations where it is difficult to capture sufficient domain knowledge. Because they leave the human out of the decision process and all “intelligent” decisions are made by the computer, autonomous expert systems require a comprehensive knowledge base covering all aspects of the tasks being performed. Some domains, such as user interface design and computer network design, are not sufficiently understood and creating a complete set of principles that adequately capture their domain knowledge is infeasible. Other domains, such as high-functionality computer systems [10, 39], are so vast that a tremendous effort is needed to acquire all relevant knowledge. Critics are better suited to these situations

¹In the remainder of the paper the term “product” is often used in a generic sense, encompassing both product in a narrow sense as well as actions.

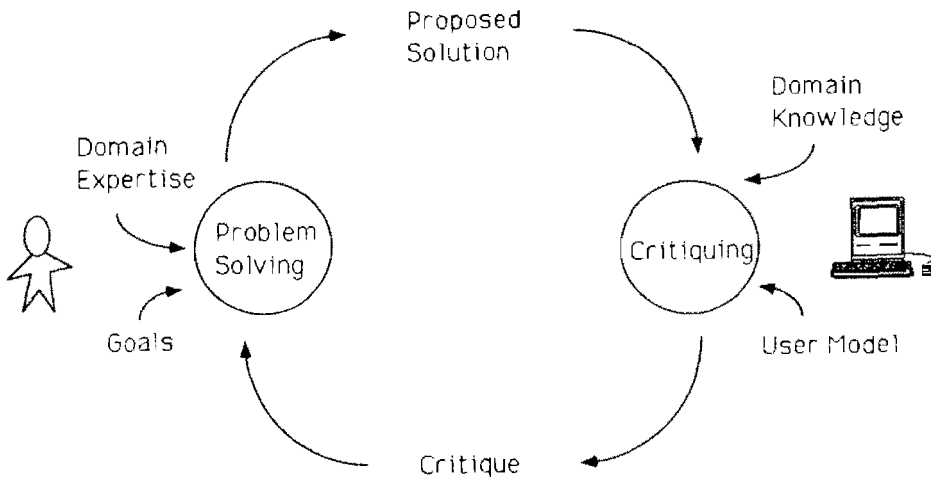


Fig. 1. The critiquing approach. This figure shows that a critiquing system has two agents, a computer and a user, working in cooperation. Both agents contribute what they know about the domain to solving some problem. The human's primary role is to generate and modify solutions; the computer's role is to analyze those solutions and produce a critique for the human to apply in the next iteration of this process.

because they need not be complete domain experts. Critics often are experts on only some aspects of the problem domain.

History. The term "critic" has been used to describe several closely related yet different ideas. It was first used in planning systems to describe internal demons that check consistency during plan generation. For example, critics in the HACKER system [62] discover errors in blocks-world programs. When a critic discovers a problem, it notifies the planner component, which edits the program as directed by the critic. The NOAH system [54] contains critics that recognize planning problems and modify general plans into more specific ones that consider the interactions of multiple subgoals. Critics in planners interact with the internal components of the planning system; critics in the sense of this paper interact with human users. Our work on critics tightly integrates the study of ill-defined problems, the development of conceptual frameworks, the development of prototypical systems instantiating the conceptual frameworks and system evaluations.

2.3 Descriptions of Some of Our Critiquing Systems

In this section, we provide an overview of critiquing systems that have influenced the development of the paradigm or that illustrate an interesting aspect of it. We describe in some detail the critiquing systems developed in our own work as mentioned in Figure 2.

Activist—Systems that volunteer information. Humans often learn by receiving answers to questions that they have never posed or were not able to pose. To ask a question, one must know how to ask it; one cannot ask

ACTIVIST	<ul style="list-style-type: none"> • active help systems • system volunteers information
LISP-CRITIC	<ul style="list-style-type: none"> • style rules define standard ways of designing artifacts • visual explanations • minimalist explanations
FRAMER	<ul style="list-style-type: none"> • extending construction kits to design environments • making the situation talk back • signaling breakdowns • checklists
JANUS	<ul style="list-style-type: none"> • integrating construction and argumentation to support reflection-in-action • relevancy to the task at hand • multiple critics with different points of view
MODIFIER	<ul style="list-style-type: none"> • competent practitioners know more than they can say (impossibility of completely articulating background assumptions) • tacit knowledge is triggered by situations, by breakdowns • critiquing knowledge is judgmental, instable, and never complete

Fig. 2. Features of our evolving critiquing systems.

questions about something if one is not aware of its existence. ACTIVIST [23] is a critic in the form of an active help system for a text editor. ACTIVIST looks “over the shoulder” of a user and infers user goals from observed actions. The system then matches the user’s actions to plans in its knowledge base that accomplish the same goals. ACTIVIST volunteers information at appropriate times based on a user model. After three suboptimal executions of a task type (measured by the number of keystrokes), ACTIVIST informs the user of a better procedure for the task. In order to be less intrusive, ACTIVIST ceases to critique actions when the user ignores its suggestions.

LISP-CRITIC—Applying critics to programming. LISP-CRITIC is a system designed to support programmers [14, 24]. It helps programmers to both improve the programs they are creating and acquire programming knowledge on demand. Programmers ask LISP-CRITIC for suggestions on how to improve their code. The system suggests transformations that make the code more cognitively efficient (i.e., easier to read and maintain) or more machine efficient (i.e., faster or smaller). Many of LISP-CRITIC’s suggestions require user confirmation because they preserve program correctness only if certain

conditions are met that cannot be derived from the program code alone. Figure 3 shows a screen image of LISP-CRITIC.

LISP-CRITIC is a passive critic; that is, users have to invoke the critic when they desire its suggestions. The system is embedded in the ZMACS editor on SYMBOLICS LISP machines. LISP-CRITIC analyzes the function definition within which the cursor is located. When the system finds pieces of code that could be improved, it shows the user its recommendation. Users can accept or reject the critic's suggestion and they can ask for an explanation to aid in making that decision. In the scenario in Figure 3, LISP-CRITIC suggests that the user replace a single conditional `cond` expression with an `if` expression. The user requests an explanation of why `if` is preferable to `cond`. The system develops an appropriate explanation based on a user model and displays the explanation in hypertext form. The user can use the explanation to access more detailed information available about LISP in an on-line documentation system (the Symbolics Document Examiner). This incremental unfolding of information spaces supports a minimalist explanation strategy [18].

LISP-CRITIC is an effective system for many users. To adequately support a wide range of user expertise, the system incorporates a user modeling component [42], which acquires and maintains information about the domain knowledge and goals of each individual user. LISP-CRITIC uses these models to customize explanations to cover exactly what the user needs to know. The models are also suitable for determining the subset of rules to fire for each user.

FRAMER—Extending construction kits to design environments. FRAMER [39, 40] is a design environment for the design of program frameworks, components of window-based user interfaces on SYMBOLICS LISP machines (Figure 4). The purpose of the FRAMER design environment is to support designers in using a high-level abstraction: program frameworks.

FRAMER contains a knowledge base of design rules for program frameworks. The rules evaluate the completeness and syntactic correctness of the design as well as its consistency with the interface style used on SYMBOLICS LISP machines. Each critic is classified as either mandatory or optional. Mandatory critics represent absolute system constraints that must be satisfied for program frameworks to function properly. Optional critics inform the user of issues that typically are dealt with in another way. The critics are active and the system displays the messages relevant to the currently selected checklist item in the window entitled *Things to take care of*.

Each message is accompanied by up to three buttons: *Explain*, *Reject*, and *Execute*. The *Explain* button displays an explanation of the reasons the designer should consider this critic suggestion; it also describes ways to achieve the desired effect. Optional suggestions have a *Reject* or *Unreject* button depending on the state of the suggestion. The *Execute* button accesses the advisory capability of FRAMER, which is available for issues that have a reasonable default solution.

A previous version of FRAMER employed a passive critiquing strategy. Experiments [39] showed that users often invoked the critic too late, after a

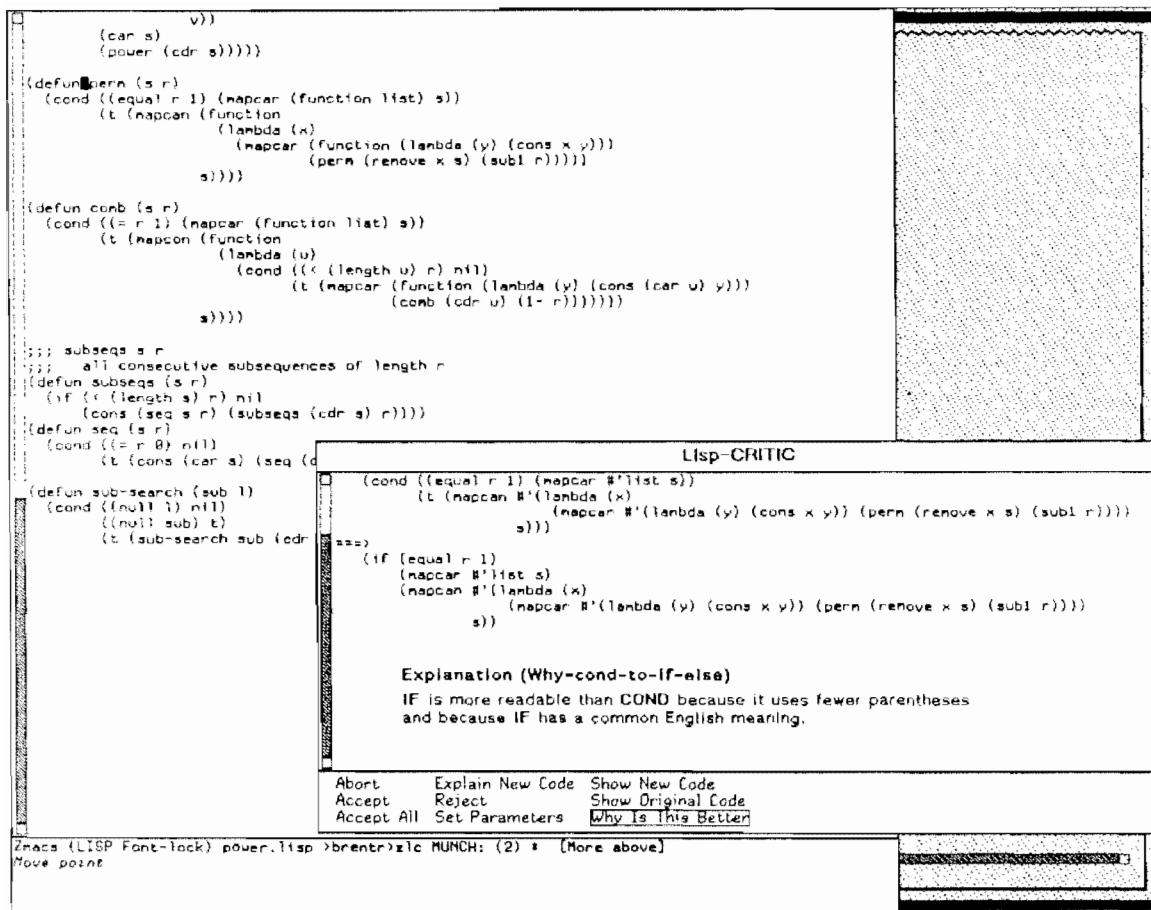


Fig. 3. The user interface of LISP-CRITIC. The large editor shows the program on which a user is working. The LISP-CRITIC window on top of it displays a "cond-to-if" transformation and an explanation of why LISP-CRITIC recommended changing the cond to an if expression.

Check List

- Initial program framework
- Program name
- Mapping this program
- Arrangement of panes
- Convert from function
- Detailed defining areas
- Icons of icons
- Panels
- Command tables
- Code Generation

What you can do:

Check list item: Arrangement of panes

Arrange the panes as desired in your program framework shown in the work area. Choose from the following mouse commands.

Work Area		Palette	
Mouse Button	Operation	Mouse Button	Operation
Left	Move pane.	Left	Get pane of this type.
Middle	Resize pane.	Middle	Describe this type.
Right	Menu of all possible operations.		
Shift-Left	Exit pane contents.		
Shift-Middle	Delete pane		

Things to take care of:

- Add a menu bar.
- Move the title pane to the top of the frame.
- Remove the overlap of DRR and TITLE. (Required)
- Fill the empty space inside the program framework. (Required)

Work Area

Palette

<input type="button" value="Titlepane"/>	<input type="button" value="Captionpane"/>
<input type="button" value="Datapane"/>	<input type="button" value="Textpane"/>
<input type="button" value="Menupane"/>	<input type="button" value="Imagepane"/>

Fig. 4 FRAMER. This figure shows a screen image of a session with FRAMER. The checklist describes the elements of the task of designing a program framework. The What you can do window shows the detailed options pertaining to a checklist item. The window entitled Things to take care of displays the critic messages. The work area is the place where frameworks are assembled in a direct manipulation interaction style. A palette contains title panes, display panes, and other primitive parts for constructing program frameworks. FRAMER also offers a catalog (not shown) for design by modification.

major incorrect decision had already been made. The newest version of FRAMER addresses this problem by continuously displaying messages. FRAMER prevents its users from permanently ignoring the critics through its checklist. Each item in the checklist describes one subactivity of designing program frameworks, and the user checks off those subactivities that have been completed. Checklist items cannot be checked off until all critic suggestions associated with a subactivity are either resolved or explicitly rejected.

JANUS—Integrating construction and argumentation. JANUS is a design environment based on the critiquing approach that allows designers to construct residential kitchens [25, 26]. JANUS contains two integrated subsystems: JANUS-CONSTRUCTION (Figure 5) and JANUS-ARGUMENTATION (Figure 6). JANUS-CONSTRUCTION is a construction kit with a set of critics and JANUS-ARGUMENTATION is an argumentative hypertext system containing information about general principles of design.

JANUS contains a critiquing component with knowledge about building codes, safety standards and functional preferences. JANUS uses this knowledge to signal breakdowns and to link construction to argumentation. JANUS displays messages explaining the nature of the breakdowns in the *Messages* window. Clicking with the mouse on a message activates JANUS-ARGUMENTATION in the context that discusses the associated breakdown.

In Figure 5, the critic points out that the circumference of the work triangle (i.e., sink, refrigerator and stove) is greater than 23 feet. Designers who are unaware of the work triangle rule do not perceive a breakdown if that rule is violated. The associated section of JANUS-ARGUMENTATION (Figure 6) explains the rationale for this rule including any exceptions. The *Cataloging Example* window of JANUS-ARGUMENTATION shows an example from the catalog illustrating a way to satisfy the work triangle rule. Critics are implemented as condition-action rules, which are triggered whenever the design is changed.

MODIFIER—Making critics user-extensible. No practical situation fits exactly into a preconceived knowledge framework. Application domains and user requirements are constantly changing. These changing environments require design environments that designers can adapt to fit unanticipated needs. Initial domain knowledge in our design environments is represented in *seeds* [19]. The seeds consist of objects in the palette, examples in a catalog, critics and argumentation.

The evolution of design environments will be severely limited if the domain experts are unable to incorporate new knowledge themselves. But domain experts are in most cases unwilling to acquire detailed knowledge about programming and knowledge engineering—therefore mechanisms supporting *end-user modifiability* are required [21]. End-user modifiability is of crucial importance in design environments for the following reasons: (1) competent practitioners usually know more than they can say; (2) tacit knowledge is triggered by situations and by breakdowns; (3) background assumptions cannot be completely articulated; (4) situations of practice are complex, unique, uncertain, conflicted and unstable; and (5) initial moves

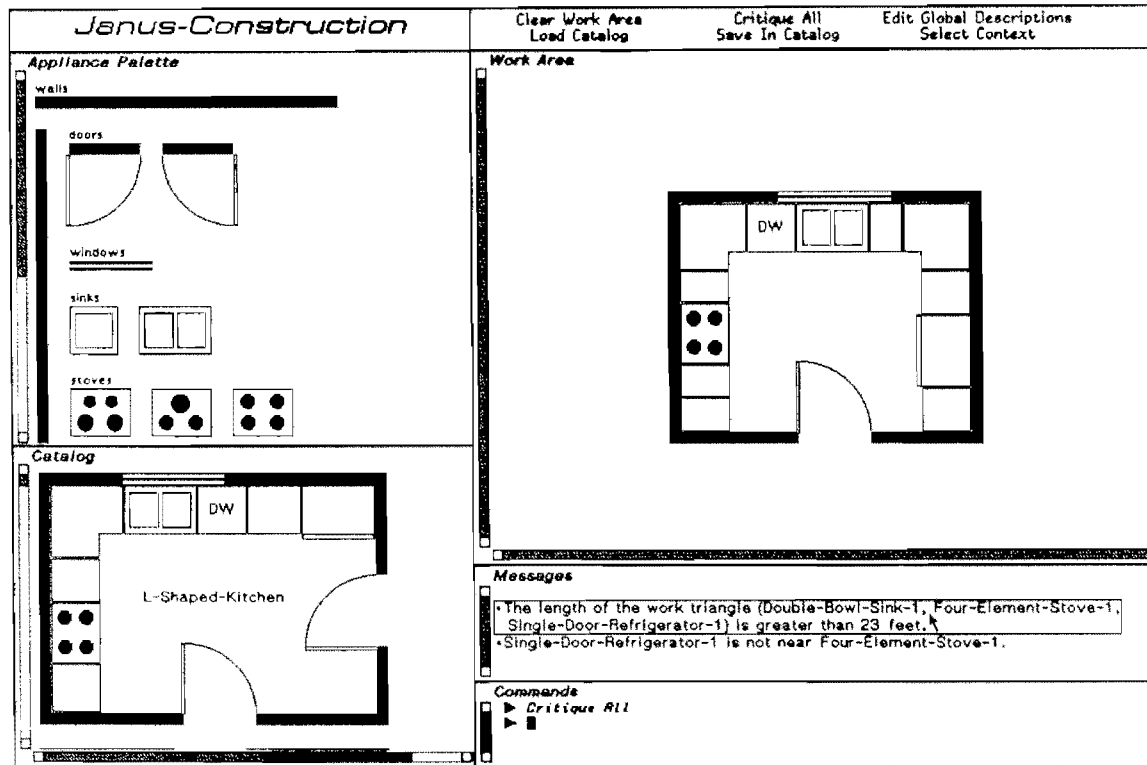
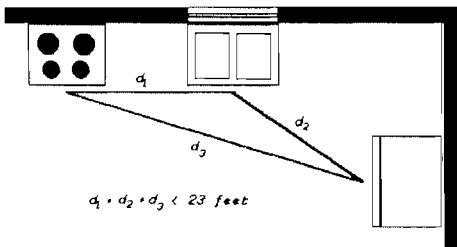


Fig. 5. JANUS-CONSTRUCTION: The work triangle critic. JANUS-CONSTRUCTION is the construction part of JANUS. Building blocks (design units) are selected from the *Palette* and moved to desired locations inside the *Work Area*. Designers can reuse and redesign complete floor plans from the *Catalog*. The *Messages* pane displays critic messages automatically after each design change that triggers a critic. Clicking with the mouse on a message activates JANUS-ARGUMENTATION and displays the argumentation related to that message (see Figure 6).

Janus-Argumentation

Answer (Refrigerator, Sink, Stove)
The distance between sink, stove and refrigerator, the *work triangle*, should be less than 23 feet.



$d_1 + d_2 + d_3 < 23 \text{ feet}$

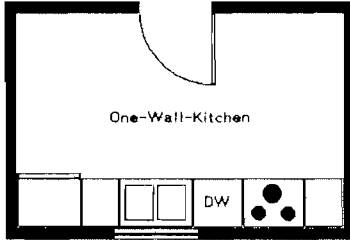
Figure 10: the work triangle

Argument (Walking Distance)
The work triangle is an important concept in kitchen design. The work triangle denotes the center front distance between the three main appliances: *sink*, *stove* and *refrigerator*. This length should be less than 23 feet to avoid unnecessary walking and to ensure an efficient work flow in the kitchen!

Argument (Small Room)
In small kitchens where the work triangle is less than 16 feet.

Viewer: Default Viewer

Catalog Example



One-Wall-Kitchen

The length of the work triangle (Stove, Refrigerator, Sink) is less than 23 feet.

Visited Nodes

- Answer (Refrigerator, Sink, Stove) Section

Commands

<ul style="list-style-type: none"> <input type="checkbox"/> Show Example: "Answer (Refrigerator, Sink, Stove)" <input type="checkbox"/> Show Example Answer (Refrigerator, Sink, Stove) 	<ul style="list-style-type: none"> Show Outline Search For Topics Show Argumentation Show Context Resume Construction Show Construction Show Example Show Counter Example
---	---

Fig. 6. JANUS-ARGUMENTATION: Rationale for the work triangle rule. JANUS-ARGUMENTATION is an argumentative hypertext system based on the PHI method [43]. The *Viewer* pane shows a diagram illustrating the work triangle concept and arguments for and against the work triangle answer. The top right pane shows an example illustrating this answer generated by the ARGUMENTATION ILLUSTRATOR. The *Visited Nodes* pane lists in sequential order the previously visited argumentation topics. By clicking with the mouse on one of these items, or on any bold or italicized item in the argumentation text itself, the user can navigate to related issues, answers, and arguments. Hypertext access and navigation features are inherited from the SYMBOLICS DOCUMENT EXAMINER.

must be reframed, as the changed situation most often deviates from the initial appreciation. The breakdowns are not experienced by the knowledge engineers, but by the domain experts using the system. In order to support evolution on a continual basis, the people experiencing the breakdowns are in the best position to do something about it. End-user modifiability is not a luxury but a necessity in cases where systems do not fit a task, a style of working or a personal sense of aesthetics.

MODIFIER (Figure 7) [21] extends JANUS with knowledge-based components that support the following types of modifications: (1) introducing new appliances (e.g., a “microwave”) into the palette, (2) adding new critic rules (e.g., “the microwave should be next to the refrigerator”) to the system, (3) adding definitions of new relationships (e.g., “between”) and (4) creating composite objects (e.g., a “cleanup center”).

2.4 Making the Situation “Talk Back” with Critics

Construction kits (such as FRAMER and JANUS) support human problem-domain communication [22] with domain-oriented building blocks. Designers using JANUS said that they experienced a sense of accomplishment when using the system because it enabled them to construct something quickly, but without needing detailed knowledge of computers. Construction kits support a design process that Schoen [56] calls reflection-in-action. Designers experiment with various shapes and discover their implications and consequences. They are likely to find unexpected meanings in the situations they create, and, “if they are good designers, they will reflect-in-action on the situation’s back talk” [57]. These unexpected meanings become apparent when a breakdown occurs [65], that is, when the designer is unable to continue action.

But construction kits do not in themselves lead to the production of interesting artifacts [22, 49]. Construction kits do not help designers perceive the shortcomings of an artifact they are constructing. As passive representations, constructions in the work area do not talk back unless the designer has the skill and knowledge to form new appreciations and understandings while constructing. Designers often do not experience breakdowns before the artifact is actually put in use. For example, designers who are unaware of the work triangle rule will not experience a breakdown if that rule is violated (see Figures 5 and 6). Critics enhance the back talk of the situation, they trigger breakdowns early in the design phase before the designer has made too many commitments that make repair expensive. Critics such as those in JANUS (1) use knowledge of design principles to detect and critique suboptimal solutions constructed by the designer, and (2) provide access paths to the relevant information in the argumentative space.

2.5 Critics in Integrated Design Environments

ACTIVIST and LISP-CRITIC operate in a stand-alone mode and are not tightly integrated with a larger design environment. We have demonstrated with JANUS, FRAMER, and with Schoen’s theory of reflection-in-action that

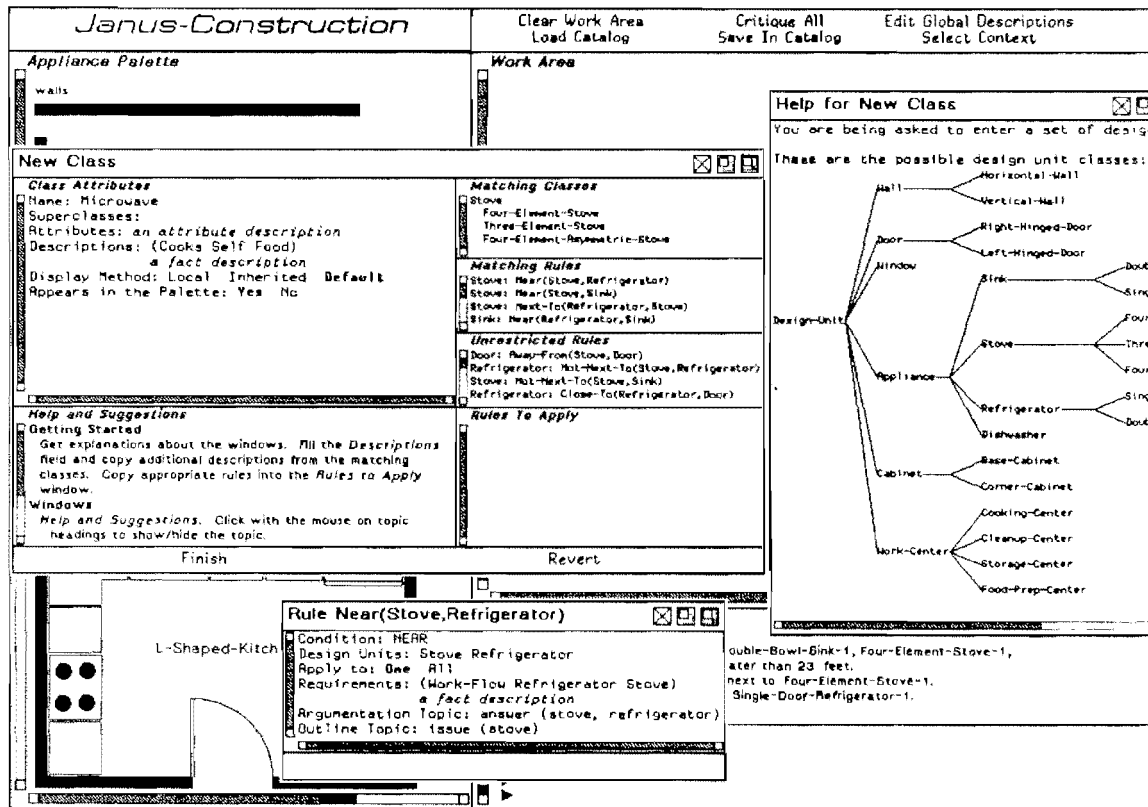


Fig. 7. MODIFIER: Introducing a microwave oven. MODIFIER is a component of JANUS. The screen image shows the system facilities for adding a new class of domain objects: microwave oven. The *New Class* window describes the attributes of microwave ovens. The window on the right shows a tree of existing domain objects. A major part of introducing a new type of object is to decide which existing critic rules should apply and what new rules should be created. The rule window at the bottom of the screen displays a critic rule that fires when a stove is placed near a refrigerator. The designer is checking to see if this rule should apply to microwave ovens as well.

integrated design environments are necessary, and critics should be considered as embedded systems rather than as stand-alone components. This insight has led us to the development of the multifaceted architecture for design environments (Figure 8). The architecture consists of the following five components:

- A *construction kit* is the principal medium for modeling a design. It provides a palette of domain concepts and supports construction using direct manipulation and electronic forms.
- An *argumentative hypertext system* contains issues, answers and arguments about the design domain. Users can annotate and add argumentation as it emerges during the design process.
- A *catalog* is a collection of prestored designs. These illustrate the space of possible designs in the domain and support reuse and case-based reasoning.
- A *specification component* allows designers to describe characteristics of the design they have in mind. The specifications are expected to be modified and augmented during the design process, rather than to be fully articulated at the beginning. They are used to retrieve design objects from the catalog and to filter information in the hypertext.
- A *simulation component* allows designers to carry out “what-if” games—that is, to simulate various usage scenarios involving the artifact being designed.

This multifaceted architecture derives its power from the *integration* of its components. Used individually, the components are unable to achieve their full potential. Used in combination, however, each component augments the value of the others in a synergistic manner. At each stage during the design process, the partially completed design that is embedded in the design environment serves as a stimulus suggesting to users what they should attend to next.

The components of the architecture are integrated by the following linking mechanisms (see Figure 8):

- CONSTRUCTION ANALYZER*. Users need support for construction, argumentation and perceiving breakdowns. The CONSTRUCTION ANALYZER is a critiquing system. The firing of a critic signals a breakdown and provides entry into the exact place in the argumentative hypertext system at which the corresponding argumentation is located.
- ARGUMENTATION ILLUSTRATOR*. The explanation given in the form of argumentation is often highly abstract and conceptual. Concrete design examples matching the explanation help users to understand the concept. The ARGUMENTATION ILLUSTRATOR helps users to understand information given in the argumentative hypertext by finding a catalog example that illustrates the concept [16] (see the top right window in Figure 6).
- CATALOG EXPLORER*. CATALOG EXPLORER helps users search the catalog space according to the task at hand [27]. It retrieves design examples

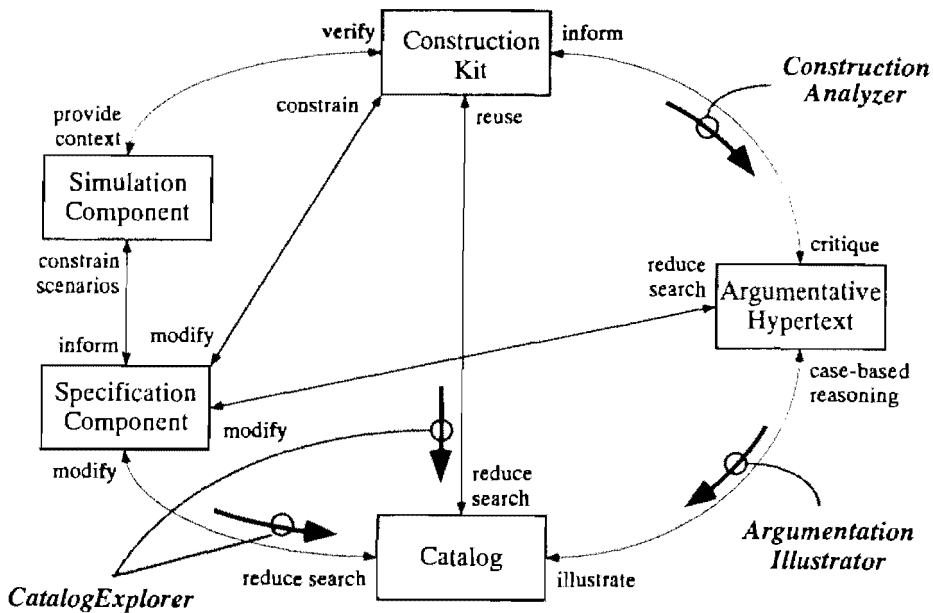


Fig. 8. A multifaceted architecture. The components of the multifaceted architecture. The links between the components are crucial for exploiting the synergy of the integration.

similar to the current construction situation and orders a set of examples by their appropriateness to the current specification.

2.6 Supporting Learning with Critics

The computational power of high-functionality computer systems can be used to provide qualitatively new learning environments. Learning technologies of the future should be multifaceted, supporting a spectrum that extends from open-ended, exploratory environments such as LOGO [50] to system-guided tutoring environments [64].

Tutors, such as the LISP TUTOR [2] step a student through a predesigned curriculum consisting of problems to be solved or information to be read. These tutors are adequate for novices in a domain, but tutors are of little help when users are involved in their “own doing” and need to *learn on demand* [17]. Tutors do not know the user’s problem; they provide their own set of example problems for the user to solve. To support user-centered learning activities, computational environments must match individual needs and learning styles. Giving users control over their learning and working requires that they become the initiators of actions, setting their own goals.

By contrast, users have unlimited control in *open learning environments*, but these environments have other problems. They do not sufficiently help learners who are stuck in a problem-solving activity or who have reached a suboptimal plateau in their problem-solving behavior. Users are often unwilling to learn more about a system or tool than what is required to solve their

immediate problems. This tendency leads to inefficient and error-prone ways of creating artifacts, lower quality artifacts and in some situations failure to successfully create an artifact at all. To successfully cope with new problems as they arise, users can benefit from *critics* that point out shortcomings in their solutions and suggest ways to improve them. With critics, users retain control and are interrupted only when their products or actions appear to be significantly inferior to the system's solution.

There is a spectrum of educational systems that give students various degrees of freedom in setting their own goals. Gaming systems such as WEST (arithmetic) [7] and DECISIONLAB (management decision making) [55] support guided discovery learning. They create a task for students but give them the freedom of exploring their own personal solution approaches. WEST constructs a diagnostic model of the student weaknesses. The system has explicit intervention and tutoring strategies enabling the system "to say the right thing at the right time." This approach works well because in the domain of arithmetic, the computer expert can play an optimal game and it can determine the complete range of alternative behaviors. This condition is not met in some other domains, such as kitchen design.

The Voltaville system [32] allows more exploratory behavior than WEST. Voltaville is a prototype discovery environment designed to build scientific inquiry skills in the context of learning the principles of DC circuits. A similar exploratory learning environment is STEAMER/Feedback MiniLab [30]. In this system, students assemble simulated devices, such as steam plant controllers, from primitive elements. The system recognizes some instances of known devices and identifies common bugs in them.

GRACE [3] is a learning environment for COBOL programming integrating a critic and a tutor. It supports both system-directed and exploratory learning. While the system is functioning as a critic, it can decide to adopt the tutoring mode to give remedial problems; conversely, while functioning as a tutor, the system may decide to let the student explore in the critiquing mode. In either case, the system provides direct accessible hypertext reference information for on-line help.

By integrating working and learning, critics offer unique opportunities: Users understand the purposes or uses for the knowledge they are learning; they learn by actively using knowledge rather than passively perceiving it and they learn at least one condition under which their knowledge can be applied. A strength of critiquing is that learning occurs as a *natural by-product* of the problem-solving process.

3. THE CRITIQUING PROCESS

The design and evaluation of the systems discussed in the previous section led to an understanding of the theoretical aspects of critiquing. Our theoretical framework specifies the following subprocesses of critiquing: goal acquisition, product analysis, critiquing strategies, adaptation capability, explanation and argumentation, and advisory capability (Figure 9). Each subprocess raises associated design issues and alternative resolutions of these issues are

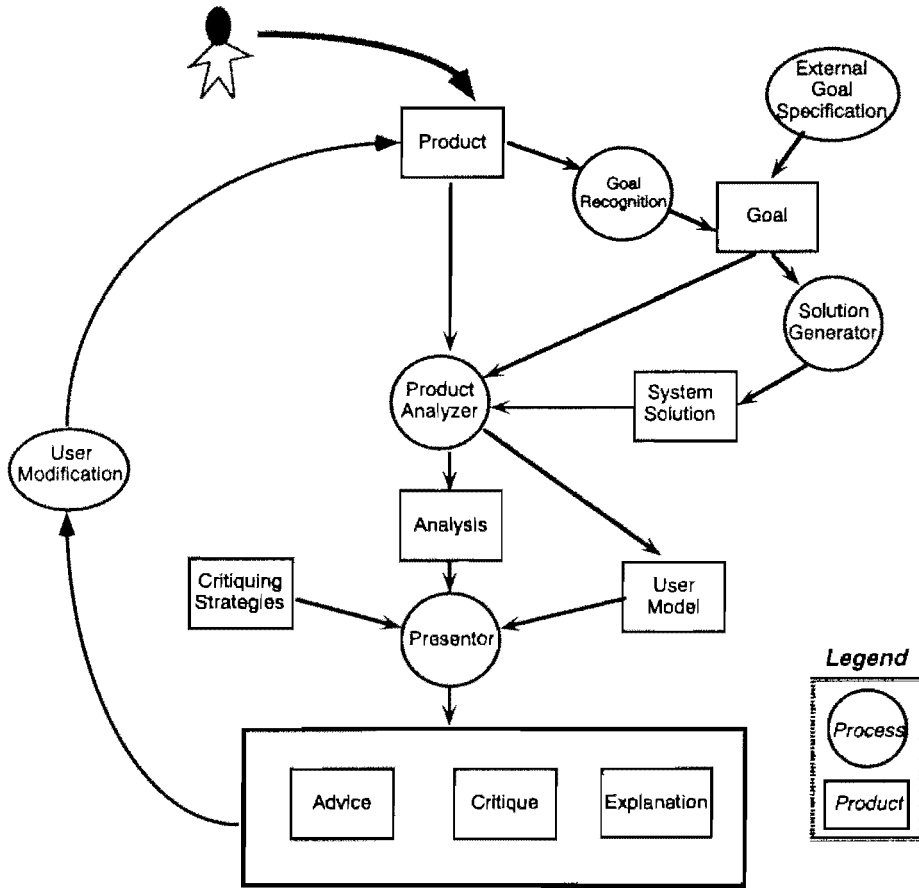


Fig. 9. The critiquing process. Users initiate the critiquing process by presenting a product to the critic. In order to evaluate the product, the critic needs to obtain a specification of the user's goals either by recognizing them from the product or by explicit input from the user. The product analyzer evaluates the product against the goal specification. Some critics do this by generating their own solution and comparing it to that of the user. 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 output of the critic, the user generates a new version of the product, and the design process goes through the same cycle, integrating the new insight.

discussed. Many issues are not currently fully understood and are subjects of ongoing research. Most critiquing systems, including JANUS, implement only a subset of all processes. We use JANUS and other systems to illustrate the processes.

3.1 Goal Acquisition

Critiquing a product requires at least a limited understanding of the intended purpose of the product. Problem knowledge can be separated into domain knowledge and goal knowledge. Having only domain knowledge without any understanding of the particular goals of the user, a critic can

reason only about characteristics that pertain to all products in the domain; for example, in a procedural domain, the syntactical correctness and compatibility of preconditions and postconditions of operators.

Domain knowledge allows JANUS, for example, to point out that stoves should not be placed in front of a window because this arrangement constitutes a fire and burn hazard. For a more extensive evaluation of a product, some understanding of the user's specific goals and situation is required. A critic can acquire an understanding of the user's goals in several ways:

- The simplest approach is *implicit goal acquisition*. A general goal is directly built into the critic system. For example, JANUS is built for the problem domain of residential kitchen design, and the user's goal is assumed to be to design a "good" residential kitchen. The knowledge base of the system needs to be modified to cope with designs of other kinds of kitchens, such as in restaurants or mountain cabins.
- User goals can be recognized by observing the evolving product. A kitchen with a table and chairs suggests that the user intends to eat meals in the kitchen. A critic can recognize this goal and suggest better solutions, such as a counter, that use less space and do not interfere with the work flow. Goal recognition presupposes solutions that approximate a solution to the user's problem. If the product fails to come close to the user's goal, the critic cannot infer that goal or might infer a goal different from the user's goal. For goal recognition, results of research on plan recognition in artificial intelligence [8] can be applied.
- The specification component provides an explicit representation of the problem to be solved [27, 28]. User specifications often contain conflicting preferences, such as low cost and large size, and a specification component must be capable of representing these.

The condition parts of critic rules reference the goal knowledge obtained from the specification component and the goal recognizer. For example, rules about eating areas will fire only if the system knows that the user wants to include an eating area.

3.2 Product Analysis

There are two general approaches to critiquing: *differential* and *analytical* critiquing. In the former approach, the system generates its own solution, compares it with the user's solution and points out the differences. An advantage of differential critiquing is that all differences can be found. Some domains allow radically different, but equally valid, solutions. This is a potential problem if the system generates its solution without regard to the user's solution approach. If user and system solutions differ fundamentally, the critic can say only that the system solution achieves good results, but it cannot explain why the user's solution is less than optimal. The ATTENDING system [46], for example, first parses the user's solution into a goal/subgoal hierarchy and then evaluates each node in a top-down manner using its own solution generator. By choosing the user's approach whenever it is not suboptimal, the system is guaranteed to approximate the user's solution as closely as possible.

Different solution attempts fulfill the goals to varying degrees or may be associated with other undesirable effects. In such situations, *metrics* (utility functions) are needed to measure the quality of alternative solutions [23]. Based on the controversial nature of design problems, alternative, conflicting metrics can be defined and have to be reconciled by negotiation and argumentation. A critique generated by JANUS-CONSTRUCTION is backed up with “pro” and “con” arguments in JANUS-ARGUMENTATION. LISP-CRITIC suggests transformations that increase either cognitive or machine efficiency. The ROUNDSMAN system [52] is a critic in the domain of breast cancer treatment, a domain in which there is no well-defined metric to compare different treatments. Therefore, ROUNDSMAN bases its critique on studies from the medical literature.

An *analytic critic* checks products with respect to predefined features and effects. Analytical critics identify suboptimal features using pattern matching [14], finite state machines or augmented transition networks [23] and expectation-based parsers [13]. In analytical approaches, critics do not need a complete understanding of the product. JANUS is an analytical critic that uses a set of rules to identify undesirable spatial relationships among kitchen design units, but it does not identify all possible problems within a kitchen design. Its rule base allows it to criticize kitchens without exactly knowing the requirements and preferences of the kitchen designer.

Critics for large designs must operate on intermediate states and not only on complete products. A design rule in the domain of kitchen design specifies a certain minimum window area. The critiquing component of JANUS must be able to deal with temporary violations to avoid bothering users when they have not yet included all the windows in their design.

Some critics receive a stream of information that is not yet separated into individual products or actions. ACTIVIST, for example, receives a stream of keystrokes that contain subsequences representing meaningful actions such as transposing two words. Such systems face several problems: action sequences are hard to delineate; sequences of actions may constitute a useful plan but may also be the beginning of a different, larger, not yet complete plan; and different plans may overlap or be included within each other. For example, users may delete a word at one place in a text, then correct a spelling mistake and finally paste the word at a different place. This composite action sequences needs to be recognized as an interleaved execution of a correct-spelling plan and an exchange-words plan. Critics must decide how long to wait for later parts of a plan and whether interspersed actions interfere with the interrupted plan. WIZARD [13] is an active help system for users of the VMS operating system command language. WIZARD uses an expectation-based parser to recognize contiguous and noncontiguous command sequences containing interspersed commands from other goals.

3.3 Critiquing Strategies

Critiquing strategies and a user model control the presentation component of a critic. The critiquing strategies determine what aspects of a design to critique and when and how to intervene in the working process of the user.

Critiquing strategies differ, depending on the dominant use of a critiquing system either to help users solve their problems or as educational systems.

Personal and social issues. Critics will be accepted and used only if they address personal and social concerns. Critics should be integrated into the work environment in a way such that users welcome their existence. Like recommendations from colleagues or co-workers, messages from a critic can be seen as helpful or hindering, as supportive or interfering with the accomplishment of goals. Critiquing strategies should consider intrusiveness and emotional impact on the user.

Intrusiveness is the users' perception of how much the critiquing process is interfering with their work. Critics can either interfere too much or fail to provide sufficient help, depending on the frequency of feedback, the complexity of the tasks and the sophistication of the user. As our experience with FRAMER has shown, critics should intervene when the user has made a potentially suboptimal choice and later choices are likely to depend on it. To avoid the cost of revising whole chains of decisions, critics should point out problems early. On the other hand, it is not critical to critique independent choices immediately after they have been made. FRAMER employs its checklist to prevent users from completely ignoring unintrusively displayed critic messages.

Emotional impact relates to how users feel about having a computer as an intelligent assistant. Critiquing from a computer might be more tolerable than critiquing from a human because it is handled as a private matter between the user and the computer. Everybody who has used a spelling checker has used a simple critiquing system. Users do not face the negative aspects that can be associated with interpersonal communication. A critique can cause anxiety if users know it can be seen by other people who might form a negative opinion of them. Users of our systems have welcomed the input from the critics. We have not observed any negative emotional impact.

What should be critiqued? *Educational critics*, whose prime objective is to support learning, and *performance critics*, whose prime objective is to help produce better products, have different requirements for their critiquing strategies. Performance critics should help users create high-quality products in the least amount of time using as few resources as possible. Learning is not the primary concern of performance systems but can occur as a by-product of the interactions between users and critics. Educational critics should maximize the educational effect; the quality of the product is a secondary concern.

Most performance critics (e.g., FRAMER, JANUS, ROUNDSMAN [52], KATE [12]) do not select specific aspects of a product to critique; they evaluate the product as a whole to achieve the highest possible quality. Some critics selectively critique based on the policies specified by users. LISP-CRITIC, for example, operates differently depending on whether cognitive efficiency or machine efficiency is specified as the primary concern for writing LISP programs.

Educational critics (e.g., the WEST system [7]) usually employ a more complex intervention strategy that is designed to enhance the educational experience and the learners motivation. For example, WEST never critiques the student on two consecutive moves. Continuous critiquing without giving users a chance to explore their own ideas is intrusive and may reduce the motivation to learn.

Negative versus positive critics. Most existing critics operate in the *negative* mode by pointing out suboptimal aspects of the users' products. *Positive* critics recognize the good parts of a product and inform users about them. Positive performance critics help users retain the good aspects of a product in further revisions; positive educational critics reinforce the desired behavior and aid learning. JANUS has a *Praise Design Unit* command that acts as a positive critic by printing out all the design principles a design unit satisfies.

Intervention strategies. Intervention strategies determine when and how a critic should signal a breakdown. *Active critics* exercise control over the intervention strategy by critiquing a product or action at an appropriate time. They function like active agents by continuously monitoring user actions. Active critics can respond to individual user actions. *Passive critics* are explicitly invoked by users when they desire an evaluation. Passive critics usually evaluate the (partial) product of a design process, not the individual user actions that resulted in the product. The active critiquing strategy is infeasible if a continuous evaluation of the design is computationally too expensive. Analyses in digital circuit design, for example, are typically run in batch mode [37]. The problem can be alleviated by using a truth maintenance system [59]. Passive critics can also be used if the critic information is used only at certain times. For example, WANDAH [31] is a system that assists writers by providing feedback on structural problems and statistical properties of the text. This information is useful to review and revise a text but is not used during the initial generation of a text.

For active critics, intervention strategies must specify when to send messages to users. Intervening immediately after a suboptimal or unsatisfactory action has occurred (immediate intervention strategy) has the advantage that the problem context is still active in the user's mind (action-present), and the user still knows how he arrived at the solution. The problem can be corrected immediately without causing dependent problems. A disadvantage of active critics is that they may disrupt a cognitive process (knowing-in-action), causing short-term memory loss. Users then need to reconstruct the goal structure that existed before the intervention. JANUS is a predominantly active critic, but users can also request critiquing on demand by running the *Critique All* command.

Critics can use any of various intervention modes that differ in the degree to which they attract users' attention. A critic can force users to attend to the critique by not allowing them to continue with their work. A less intrusive mode is the display of messages in a separate critic window on the screen. This gives users a choice of whether to read and process the message immediately or first complete an action in progress. However, there is a risk

that messages go unnoticed, and users often have trouble following written advice [35]. In response to this problem, Wroblewski et al. [67] have coined the notion of “advertising.” Advertising means drawing the user’s attention to the work materials that bear more work rather than drawing attention to a separate window. Systems advertise services, for instance, by specially marking those objects that are affected by critic messages.

3.4 Adaptation Capability

To avoid repetitive messages and to accommodate different user preferences and users with different skills, a critiquing system needs an adaptation capability. A critic that persistently critiques the user on a position with which the user disagrees is unacceptable, especially if the critique is intrusive. Equally unacceptable is a critic that constantly repeats an explanation that the user already knows.

Critics can be adaptable or adaptive. Systems are called adaptable if the user can change the behavior of the system (see MODIFIER in Section 2.3). An adaptive system is one that automatically changes its behavior based on information observed or inferred. An adaptation capability can be implemented by simply disabling or enabling the firing of particular critic rules, by allowing the user to modify or add rules and by making the critiquing strategy dependent on an explicit, dynamically maintained user model. ACTIVIST [23] uses a dynamically maintained user model in its intervention strategy. For each text-editing goal it knows and for each user, ACTIVIST records how often the goal was accomplished, how often it was accomplished optimally, how often the user was notified and other information. ACTIVIST uses this model to adapt its strategy. For example, a message concerning the same error will be given only a limited number of times, and no message will be given after the user has successfully executed a plan a certain number of times.

How to acquire and represent individual user models remains a topic of ongoing research [42]. User modeling in critics (e.g., in ACTIVIST) shares ideas and goals with student modeling in intelligent tutoring systems [9] and advice-giving natural language dialogue systems [38].

3.5 Explanation and Argumentation

Users who are not competent to assess the critic’s judgments have been observed to blindly follow the critics’ suggestions [39]. Therefore, users need to be able to obtain information about the rationale for the critique. Simple explanation components provide prestored text explanations [40, 47]. But there is not always a simple explanation. If design is an argumentative process [53], an explanation component capable of presenting different alternatives and opinions and their corresponding advantages and disadvantages is necessary [20]. Argumentation is not a separate activity but, to be effective, must take place within the action present, that is, within the time period during which it can still make a difference as to what action is taken. If the time required to read and/or record explanatory argumentation is

greater than the action present, design is disrupted and the required context is lost. This observation led to the requirement for a strong integration of the components of our design environments (Section 2.5).

Another approach to explanation is to simulate and visualize the processes under consideration. LISP-CRITIC is capable of visualizing the effects of certain LISP functions [14]. KATE [12] critiques software specifications and backs up its critique with simulation scenarios designed to approximate the rich set of examples that software professionals use.

3.6 Advisory Capability

All critics detect breakdowns in the product (*problem detection mode*). Some critics require the *user* to determine how to resolve the breakdowns by making changes to address the problems pointed out by the critic. Other critics are capable of suggesting alternatives to the user's solution. We call these *solution-generating* critics. In the JANUS system, critics detect the problems and the argumentation component suggests alternative solutions and provides arguments for and against these alternatives.

4. DISCUSSION AND CONCLUSIONS

The systems described in this paper show that critiquing is an emerging paradigm for knowledge-based systems. This section assesses the system-building efforts and the conceptual framework previously presented.

Experiences. Empirical evaluations of the systems constructed in our research (see Fischer et al. [25] for JANUS, Fischer and Mastaglio [24] for LISP-CRITIC, and Lemke [39] for FRAMER) demonstrated that critiquing systems support incremental learning of high-functionality computer systems, provide a new approach in support of learning, extend construction kits to design environments and support cooperative problem solving. The evaluation results often pointed out new directions for our research.

FRAMER and LISP-CRITIC are tools used by researchers and students in our laboratory; this enabled us to evaluate them in actual work settings. The members of our group use FRAMER for designing their screen layouts. LISP-CRITIC has been used by students learning to program in LISP as well as by experienced programmers. Professional kitchen designers and computer experts acting as amateur designers have evaluated JANUS in a laboratory setting.

Building a knowledge-based system is a major effort and critics are no exception. Realistic systems that provide broad functionality and support tools are needed to test the usefulness of critics in actual settings. Critics are often *embedded systems*; for example, they constitute only one part of the JANUS and FRAMER design environments.

The strengths of critics are that they support users who are involved in their own work and that they integrate learning with that work. As noted in several recent research efforts [11, 44, 56, 61, 65], professional practice in design is both action *and* reflection. The basis for design is a combination of personal involvement and rational understanding, rather than detached

reflection. Systems such as JANUS and FRAMER make the situation “talk back” and critics help designers to deal with breakdowns in a constructive fashion. By showing that the artifact under construction has shortcomings, critics cause users to pause for a moment, to reflect on the situation and to apply new knowledge to the problem as well as to explore alternative designs. By serving as skill-enhancing tools, critics support the “Scandinavian approach to system design” [4, 11], which attempts to develop “systems for experts” rather than expert systems. Critics help users to become lay designers and they remind professional designers of the principles of good design.

Limitations of current critics and future research issues. One of the features that contributes to the strengths of critics is at the same time a potential weakness. Supporting users in their own doing means that detailed assumptions about what a user might do cannot be built into the system. Our critic systems have only a limited understanding of the goals users pursue. This limitation restricts the amount of assistance and detailed goal-oriented analysis that critics can provide. By moving from generic domains such as LISP programming to more narrowly defined domains such as kitchen design, our critiquing systems took advantage of the richer domain semantics and became more powerful. Other researchers have worked on systems that have a deep understanding of a very small set of problems [36]. We are working on solutions to the goal acquisition problem by developing a specification component for JANUS that allows users to communicate their goals to the system [27, 28]. This goal knowledge will activate or disable critics, for example, if the kitchen is specified for a short, tall or handicapped person.

Users should be able to modify critics without having to possess detailed programming knowledge. In developing MODIFIER, first steps were made in this direction. Systems with sufficient inference and user modeling capabilities, such as ACTIVIST, can control the critics for the user. Interactions with kitchen designers demonstrated that they test their designs by mentally simulating tasks in the kitchen under construction. Critics should employ a simulation component when it is necessary to evaluate a design.

Observation of users of JANUS and FRAMER showed that users sometimes do not notice the critiques generated by the system or that they ignore the advice. A more detailed analysis of attention and intervention is required to develop critiquing strategies that insure that users do not miss important information, but at the same time users should not be interrupted when it is more appropriate for them to focus on other issues.

Currently, most critics support only “one-shot dialogs” [1]. The critiquing systems discussed in this paper respond to user actions; they give suggestions and provide explanations and argumentation. But systems do not achieve the cooperative problem-solving ability of human critics [51] and they do not increase problem understanding to the same degree. As mentioned briefly in Section 2.1, the role distribution can also be reversed: the computer generates and a person critiques and modifies (e.g., a computer creates a layout structure of a graph, which the user can then critique and modify). “One-shot

dialogs” can be truly overcome when the system is capable of switching roles, that is, when it can critique the user modifications.

Critics point out design principles when the designer violates them. The system’s knowledge of design principles could also be used to actively enforce the principles as constraints on user actions making violations impossible [34]. For example, the principle that the sink should be under the window could give rise to a constraint linking these two objects. Whenever the user moves one of them, the system moves the other object along. The conditions for preferring the critiquing approach or the constraint approach need to be further investigated.

Conclusions. In this paper, we have presented the critiquing approach to the design of knowledge-based computer systems supporting human work and learning. We have presented example critiquing systems and have described how they enhance the back-talk of construction situations. Critics activate otherwise passive drawings and constructions. The critiquing approach can be successfully applied in any domain in which tasks cannot be completely specified in advance and optimal solutions cannot be found algorithmically. Critics are partial problem solvers that cooperate with human users while expert systems generally offer only a complete solution that users must either accept or reject. Critics are modular, that is, individual critics can easily be added or removed without affecting the overall function of the system. But critics can help only after the designer has acted; they do not operate proactively. Critics support learning on demand for users involved in their own doing. Critics are components in the multifaceted architecture of integrated design environments.

Critics are not the only solution to building better knowledge-based systems, but we believe that a growing number of such systems will contain a critiquing component. Some of these systems will have elaborate problem understanding, but more commonly they will have limited yet helpful capabilities. Critics are an important step toward the creation of more useful and more usable computer systems for the future.

ACKNOWLEDGMENTS

Many people have contributed over the years to the development of our notion of the critiquing paradigm. The authors would like to thank especially the members of the Janus Design Project (Ray McCall, Kumiyo Nakakoji and Jonathan Ostwald); Heinz-Dieter Boecker, who developed many of the original ideas for LISP-CRITIC; Chris Morel, Brent Reeves and John Rieman, who each worked on different aspects of LISP-CRITIC at different times; and all the people who participated in discussions about the general framework for critiquing, notably: Hal Eden, Helga Nieper-Lemke, Thomas Schwab, Curt Stevens and the HCC research group as a whole.

REFERENCES

1. AARONSON, A., AND CARROLL, J. M. Intelligent help in a one-shot dialog: A protocol study. In *Human Factors in Computing Systems and Graphics Interface, CHI + GI'87 Conference Proceedings* (Toronto, Apr. 1987), ACM, New York, pp. 163-168.

2. ANDERSON, J. R., FARRELL, R. G., AND SAUERS, R. Learning to program in LISP. *Cognitive Sci.* 8, 2 (Apr.-June 1984), 87-129.
3. ATWOOD, M. E., BURNS, B., GRAY, W. D., MORCH, A. I., RADLINSKI, E. R., AND TURNER, A. The grace integrated learning environment—A progress report. In *Proceedings of the Fourth International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems (IEA/AIE 91)* (June 1991) pp. 741-745.
4. BODKER, S., KNUDSEN, J. L., KYNG, M., EHN, P., AND MADSEN, K. H. Computer support for cooperative design. In *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'88)*, ACM, New York, Sept. 1988, pp. 377-394.
5. A. H. BOND, AND L. GASSER, EDs. *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann, San Mateo, Calif. 1988.
6. BUCHANAN, B. G., AND SHORTLIFFE, E. H. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, Reading, Mass. 1984.
7. BURTON, R. R., AND BROWN, J. S. An investigation of computer coaching for informal learning activities. In *Intelligent Tutoring Systems*, D. H. Sleeman, and J. S. Brown, Eds., Academic Press, London/New York, 1982, pp. 79-98.
8. CHIN, D. N. GUEST EDITOR. Special issue on plan recognition. *User Modeling and User-Adapted Interaction* 1, 2 (1991).
9. CLANCEY, W. J. Qualitative student models. *Ann. Rev. Comput. Sci.* 1 (1986), 381-450.
10. DRAPER, S. W. The nature of expertise in UNIX. In *Proceedings of INTERACT'84, IFIP Conference on Human-Computer Interaction*, (Amsterdam, Sept. 1984), pp. 182-186.
11. EHN, P. *Work-Oriented Design of Computer Artifacts*. Almquist and Wiksell International, 1988.
12. FICKAS, S., AND NAGARAJAN, P. Critiquing software specifications. *IEEE Softw.* 5, 6 (Nov. 1988), 37-47.
13. FININ, T. W. Providing help and advice in task oriented systems. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, 1983, pp. 176-178.
14. FISCHER, G. A critic for LISP. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence* (Milan, Aug. 1987), pp. 177-184.
15. FISCHER, G. Communications requirements for cooperative problem solving systems. *Int. J. Inf. Syst. (Special Issue on Knowledge Engineering.)* 15, 1 (1990), 21-36.
16. FISCHER, G. Cooperative knowledge-based design environments for the design, use, and maintenance of software. In *Software Symposium '90* (Kyoto, June 1990), pp. 2-22.
17. FISCHER, G. Supporting learning on demand with design environments. In *Proceedings of the International Conference on the Learning Sciences 1991* (Evanston, Ill., Aug. 1991). Forthcoming.
18. FISCHER, G., MASTAGLIO, T., REEVES, B. N., AND RIEMAN, J. Minimalist explanations in knowledge-based systems. In *Proceedings of the 23rd Hawaii International Conference on System Sciences, Vol. III: Decision Support and Knowledge Based Systems Track*, IEEE Computer Society, 1990, pp. 309-317.
19. FISCHER, G., GRUDIN, J., LEMKE, A. C., MCCALL, R., OSTWALD, J., AND SHIPMAN, F. Supporting asynchronous collaborative design with integrated knowledge-based design environments. Dept. of Computer Science, Univ. of Colorado, Boulder, Colo. 1991.
20. FISCHER, G., LEMKE, A. C., MCCALL, R., AND MORCH, A. Making argumentation serve design. In *Human-Computer Interaction*. In press.
21. FISCHER, G., AND GIRGENSOHN, A. End-user modifiability in design environments. In *Human Factors in Computing Systems, CHI'90 Conference Proceedings* (Seattle, Wash., Apr. 1990), ACM, New York, pp. 183-191.
22. FISCHER, G., AND LEMKE, A. C. Construction kits and design environments: Steps toward human problem-domain communication. *Human-Computer Interaction* 3, 3 (1988), 179-222.
23. FISCHER, G., LEMKE, A. C., AND SCHWAB, T. Knowledge-based help systems. In *Human Factors in Computing Systems, CHI'85 Conference Proceedings* (San Francisco, Apr. 1985), ACM, New York, pp. 161-167.
24. FISCHER, G., AND MASTAGLIO, T. Computer-based critics. In *Proceedings of the 22nd Annual Hawaii Conference on System Sciences, Vol. III: Decision Support and Knowledge Based Systems Track*. IEEE Computer Society, Jan., 1989, pp. 427-436.

25. FISCHER, G., MCCALL, R., AND MORCH, A. Design environments for constructive and argumentative design. In *Human Factors in Computing Systems, CHI'89 Conference Proceedings* (Austin, Tex., May 1989), ACM, New York, pp. 269-275.
26. FISCHER, G., MCCALL, R., AND MORCH, A. JANUS: Integrating Hypertext with a knowledge-based design environment. In *Proceedings of Hypertext '89* (Pittsburgh, Penn., Nov. 1989), ACM, New York, pp. 105-117.
27. FISCHER, G., AND NAKAKOJI, K. Making design objects relevant to the task at hand. In *Proceedings of AAAI-91, Ninth National Conference on Artificial Intelligence* (Cambridge, Mass.). In press.
28. FISCHER, G., AND NAKAKOJI, K. Empowering designers with integrated design environments. In *Proceedings of the First International Conference on Artificial Intelligence in Design* (Edinburgh, UK). In press.
29. FISCHER, G., AND STEVENS, C. Information access in complex, poorly structured information spaces. In *Human Factors in Computing Systems, CHI'91 Conference Proceedings* (New Orleans, La., 1991), ACM, New York, pp. 63-70.
30. FORBUS, K. An interactive laboratory for teaching control system concepts. Rep. 5511, BBN, Cambridge, Mass. 1984.
31. FRIEDMAN, M. P. WANDAH—A computerized writer's aid. In *Applications of Cognitive Psychology, Problem Solving, Education and Computing*. Lawrence Erlbaum Associates, Hillsdale, N.J., 1987, Ch. 15, pp. 219-225.
32. GLASER, R., RAGHAVAN, K., AND SCHAUBLE, L. Voltaville: A discovery environment to explore the laws of DC circuits. In *Proceedings of the International Conference on Intelligent Tutoring Systems* (Montreal, June 1988), pp. 61-66.
33. I. GREIF, ED. *Computer-Supported Cooperative Work: A Book of Readings*. Morgan Kaufmann, San Mateo, Calif., 1988.
34. GROSS, M. D., AND BOYD, C. Constraints and knowledge acquisition in Janus. Dept. of Computer Science, Univ. of Colorado, Boulder, Colo. 1991.
35. HILL, W. C. How some advice fails. In *Human Factors in Computing Systems, CHI'89 Conference Proceedings* (Austin, Tex., May 1989), ACM, New York, pp. 197-210.
36. JOHNSON, W. L., AND SOLOWAY, E. PROUST: Knowledge-based program understanding. In *Proceedings of the 7th International Conference on Software Engineering* (Orlando, Fla., Mar. 1984), IEEE Computer Society, pp. 369-380.
37. KELLY, V. E. The CRITTER system: Automated critiquing of digital circuit designs. In *Proceedings of the 21st Design Automation Conference* (1985), pp. 419-425.
38. A. KOBZA AND W. WAHLSTER, EDs. *User Models in Dialog Systems*. Springer-Verlag, New York, 1989.
39. LEMKE, A. C. Design environments for high-functionality computer systems. Ph.D. thesis. Dept. of Computer Science, Univ. of Colorado, Boulder, July 1989.
40. LEMKE, A. C., AND FISCHER, G. A cooperative problem solving system for user interface design. In *Proceedings of AAAI-90, Eighth National Conference on Artificial Intelligence*. (Cambridge, Mass., Aug. 1990), pp. 479-484.
41. MALONE, T. W., GRANT, K. R., LAI, K-Y., RAO, R., AND ROSENBLITT, D. Semi-structured messages are surprisingly useful for computer-supported coordination. In *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'86), MCC* (Austin, Tex., Dec. 1986), pp. 102-114.
42. MASTAGLIO, T. User modelling in computer-based critics. In *Proceedings of the 23rd Hawaii International Conference on System Sciences, Vol. III: Decision Support and Knowledge Based Systems Track*, IEEE Computer Society, 1990, pp. 403-412.
43. MCCALL, R. PHI: A conceptual foundation for design hypermedia. To appear in *Des. Stud.* 1991.
44. MCCALL, R., FISCHER, G., AND MORCH, A. Supporting reflection-in-action in the Janus design environment. In *The Electronic Design Studio*. The MIT Press, Cambridge, Mass., 1990, pp. 247-259.
45. McDERMOTT, J. R1: A rule-based configurer of computer systems. *Artif. Intell.* 19 (Sept. 1982), 39-88.

46. MILLER, P. *A Critiquing Approach to Expert Computer Advice: ATTENDING*. Pittman, London-Boston, 1984.
47. NECHES, R., SWARTOUT, W. R., AND MOORE, J. D. Enhanced maintenance and explanation of expert systems through explicit models of their development. *IEEE Trans. Softw. Eng. SE-11* (Nov. 1985), 1337-1351.
48. NIEPER-LEMKE, H. TRISTAN, ein generischer Editor fuer gerichtete Graphen. In *Prototypen benutzergerechter Computersysteme*. Walter de Gruyter, Berlin-New York, 1988, ch. XIV, pp. 243-257.
49. NORMAN, D. A. Cognitive engineering. In *User Centered System Design, New Perspectives on Human-Computer Interaction*. Lawrence Erlbaum Associates, Hillsdale, N.J., 1986, ch. 3, pp. 31-62.
50. PAPER, S. *Mindstorms: Children, Computers and Powerful Ideas*. Basic Books, New York, 1980.
51. REEVES, B. N. Locating the right object in a large hardware store—An empirical study of cooperative problem solving among humans. Tech. Rep. CU-CS-523-91, Dept. of Computer Science, Univ. of Colorado, Boulder, 1991.
52. RENNELS, G. D. *A Computational Model of Reasoning from the Clinical Literature*. Springer Verlag, New York, 1987.
53. RITTEL, H. W. J., AND WEBBER, M. M. Planning problems are wicked problems. In *Developments in Design Methodology*, N. Cross, Ed., Wiley, New York, 1984, pp. 135-144.
54. SACERDOTI, E. D. A structure for plans and behavior. Tech. Note 109, Stanford Research Institute, Stanford, Calif., 1975.
55. SCHIFF, J., AND KANDLER, J. Decisionlab: A system designed for user coaching in managerial decision support. In *Proceedings of the International Conference on Intelligent Tutoring Systems* (Montreal, June 1988), pp. 154-161.
56. SCHOEN, D. A. *The Reflective Practitioner: How Professionals Think in Action*. Basic Books, New York, 1983.
57. SCHOEN, D. A. *Educating the Reflective Practitioner*. Jossey-Bass, San Francisco, 1987.
58. SIMON, H. A. The structure of ill-structured problems. *Artif. Intell.* 4 (1973), 181-200.
59. STEELE, R. L. Cell-based VLSI design advice using default reasoning. In *Proceedings of 3rd Annual Rocky Mountain Conference on AI* (Denver, Colo. 1988), Rocky Mountain Society for Artificial Intelligence, pp. 66-74.
60. STEFIK, M. J. The next knowledge medium. *AI Magazine* 7, 1 (Spring 1986), 34-46.
61. SUCHMAN, L. A. *Plans and Situated Actions*. Cambridge University Press, Cambridge, UK, 1987.
62. SUSSMAN, G. J. *A Computer Model of Skill Acquisition*. American Elsevier, New York, 1975.
63. WEBBER, B. L., AND FININ, T. W. In response: Next steps in natural language interaction. In *Artificial Intelligence Applications for Business*, W. Reitman, Ed., Ablex, Norwood, N.J., 1984, ch. 12, pp. 211-234.
64. WENGER, E. *Artificial Intelligence and Tutoring Systems*. Morgan Kaufmann, Los Altos, Calif. 1987.
65. WINOGRAD, T., AND FLORES, F. *Understanding Computers and Cognition: A New Foundation for Design*. Ablex, Norwood, N.J., 1986.
66. WOODS, D. D. Cognitive technologies: The design of joint human-machine cognitive systems. *AI Magazine* 6, 4 (Winter 1986), 86-92.
67. WROBLEWSKI, D. A., McCANDLESS, T. P., AND HILL, W. C. DETENTE: Practical support for practical action. In *Human Factors in Computing Systems, CHI'91 Conference Proceedings* (New Orleans, La., 1991), ACM, New York, pp. 195-202.