

## Critics: An Emerging Approach to Knowledge-Based Human Computer Interaction

**Gerhard Fischer, Andreas C. Lemke and Thomas Mastaglio**  
Department of Computer Science and Institute for Cognitive Science  
University of Colorado, Boulder, USA

**Anders I. Morch**  
NYNEX Artificial Intelligence Laboratory  
White Plains, NY, USA

*To appear in: International Journal of Man-Machine Studies (IJMMS)*

**Abstract:** We describe the critiquing approach to building knowledge-based interactive systems. Critiquing supports computer users in *their* problem solving and learning activities. The challenges for the next generation of knowledge-based systems provide a context for the development of this paradigm. We discuss critics from the perspective of overcoming the problems of high-functionality computer systems, of providing a new class of systems to support learning, of extending applications-oriented construction kits to design environments, and of providing an alternative to traditional autonomous expert systems. One of the critiquing systems we have built — JANUS, a critic for architectural design — is used as an example for presenting the key aspects of the critiquing process. We then survey additional critiquing systems developed in our and other research groups. The paper concludes with a discussion of experiences and extensions to the paradigm.

**Acknowledgements.** Many people have contributed 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), the members of the LISP-CRITIC project (Heinz-Dieter Boecker, Chris Morel, Brent Reeves, and John Rieman), all the people who have participated in discussions about the general framework for critiquing (Thomas Schwab, Helga Nieper-Lemke, Curt Stevens, Tom DiPersio, and Hal Eden), and the HCC research group as a whole. This research was partially supported by grant No. IRI-8722792 from the National Science Foundation, grant No. MDA903-86-C0143 from the Army Research Institute, and grants from the Intelligent Interfaces Group at NYNEX and from Software Research Associates (SRA), Tokyo.



# Critics: An Emerging Approach to Knowledge-Based Human Computer Interaction

Gerhard Fischer, Andreas C. Lemke, and Thomas Mastaglio  
Department of Computer Science and Institute for Cognitive Science  
University of Colorado, Boulder, USA

Anders I. Morch  
NYNEX Artificial Intelligence Laboratory  
White Plains, NY, USA

## Abstract

We describe the critiquing approach to building knowledge-based interactive systems. Critiquing supports computer users in *their* problem solving and learning activities. The challenges for the next generation of knowledge-based systems provide a context for the development of this paradigm. We discuss critics from the perspective of overcoming the problems of high-functionality computer systems, of providing a new class of systems to support learning, of extending applications-oriented construction kits to design environments, and of providing an alternative to traditional autonomous expert systems. One of the critiquing systems we have built — JANUS, a critic for architectural design — is used as an example for presenting the key aspects of the critiquing process. We then survey additional critiquing systems developed in our and other research groups. The paper concludes with a discussion of experiences and extensions to the paradigm.

**Keywords:** critics, critiquing, high functionality computer systems, intelligent support systems, design environments, cooperative problem solving systems.

## Introduction

A critic is a system that presents a reasoned opinion about a product or action generated by a human. The critiquing approach is an effective way to make use of computer knowledge bases to aid users in their work and to support learning. Our experience with this approach includes several years of innovative system building efforts, the integration of cognitive and design theories, empirical observations, and the evaluation of prototypes. This paper combines our experience with the research efforts of others to articulate foundations and characteristics for the critiquing paradigm. We describe the rationale for critiquing (Section 2) and illustrate the approach using one of our systems (JANUS) as an example (Section 4). Section 5 gives a general characterization of the critiquing process. Other critics are surveyed in terms of the critiquing framework, showing the applicability and usefulness of critics in other domains (Section 6). We conclude with a discussion of future directions for research on the critiquing paradigm.

## Challenges for the Next Generation of Knowledge-Based Systems

The next generation of knowledge-based systems will present the following challenges:

- They will be high functionality systems, and their complete mastery will exceed the cognitive capabilities of most individuals.
- They will need to support a broad spectrum of learning and working activities.
- They should be integrated design environments.
- Rather than autonomous expert systems, they will often be joint human-computer systems supporting cooperative problem solving.

We will discuss how critics can meet each of these challenges.

## High-Functionality Computer Systems

As powerful computer hardware has become widely available, so have software systems for general applications with a large range of capabilities. Technical complexity and the associated human cognitive costs to master these systems have grown dramatically limiting the ability of users to take full advantage of them. One illustration of this situation is the Symbolics LISP machine; it contains over 30,000 functions and 3300 flavors (or classes) accompanied by 12 books with 4400 pages of written documentation. Even a modern microcomputer word processor has more than 400 pages of documentation and an amount of functionality that very few people master in its entirety.

For systems to be useful and applicable to a wide range of problems, they have to offer rich functionality. Modern computer systems are best understood not in their capacity to compute but to serve as *knowledge stores*. And because the amount of knowledge is extensive, these systems will not be small and simple, but will be large and complex.

“Reality is not user friendly,” i.e., typical problems are complex and cannot be easily solved using a small set of general building blocks and tools. Such minimal tools are not always useful even though, in principle, anything can be computed with them. These tools force the user to reinvent the wheel rather than supporting reuse and redesign of existing components.

Systems that offer a rich functionality are a mixed bless-

ing. In a very large knowledge space, something related to what we need is likely to exist but may be difficult to find. It is impossible and infeasible for any one individual to know such systems completely. Empirical studies (Draper, 1984) have shown that even very experienced users know only a subset of a large system. They encounter the following problems: They do not know about the *existence* of building blocks and tools; they do not know how to *access* tools, or *when* to use them; they do not understand the *results* that tools produce, and they cannot combine, adapt, and modify tools according to their *specific* needs. Our goal is to increase the usability of high functionality computer systems, not by “watering down” functionality or steering the user toward only a subset of the systems’ capabilities, but by facilitating learning about, access to, and application of the knowledge these systems contain. Critics contribute to these goals by providing the user with selective information at the time it is needed.

### Systems to Support Learning

The computational power of high functionality computer systems can provide qualitatively new learning environments. Learning technologies of the future should be multi-faceted, supporting a spectrum extending from open-ended, user-centered environments such as LOGO (Papert, 1980) to guided, teacher-centered tutoring environments (Wenger, 1987).

*Tutoring* is one way to support learning the basics of a new system. One can pre-design a sequence of microworlds and lead a user through them (Anderson & Reiser, 1985). However, tutoring is of little help in supporting learning on demand when users are involved in their “own doing.” Tutoring is not task-driven, because the total set of tasks *cannot* be anticipated. To support user-centered learning activities, we must build computational environments that match individual needs and learning styles. Giving users control over their learning and working requires that they become the initiators of actions and set their own goals.

In open learning environments users have unlimited control (Papert, 1980), but there are other problems. They do not support situations where users get stuck during a problem solving activity or settle at a suboptimal plateau of problem solving behavior. To successfully cope with new problems, users can benefit from a critic that points out shortcomings in their solutions and suggests ways to improve them.

In contrast to passive help systems, critics do not require users to formulate a question. Critics allow users to retain control; they interrupt only when users’ products or actions could be improved. 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 the knowledge can be applied. A strength of critiquing is that learning occurs as a *natural byproduct* of the problem solving process.

### Design Environments

To accomplish most things in this world, selective search, means-ends analysis, and other weak methods are not sufficient (Simon, 1986); one needs to employ strong problem solving techniques with knowledge about the task domain. Designers—architects, composers, user interface designers, database experts, knowledge engineers—are experts in *their* problem domain and are not interested in learning the “languages of the computer”; they simply want to use the computer to solve their problems and accomplish required tasks. To shape the computer into a truly usable as well as useful medium, we have to make low-level primitives invisible. We must “teach” the computer the languages of experts by endowing it with the abstractions of application domains. This reduces the transformation distance between the domain expert’s description of the task and its representation as a computer program. *Human problem-domain communication* is our term for this idea (Fischer & Lemke, 1988).

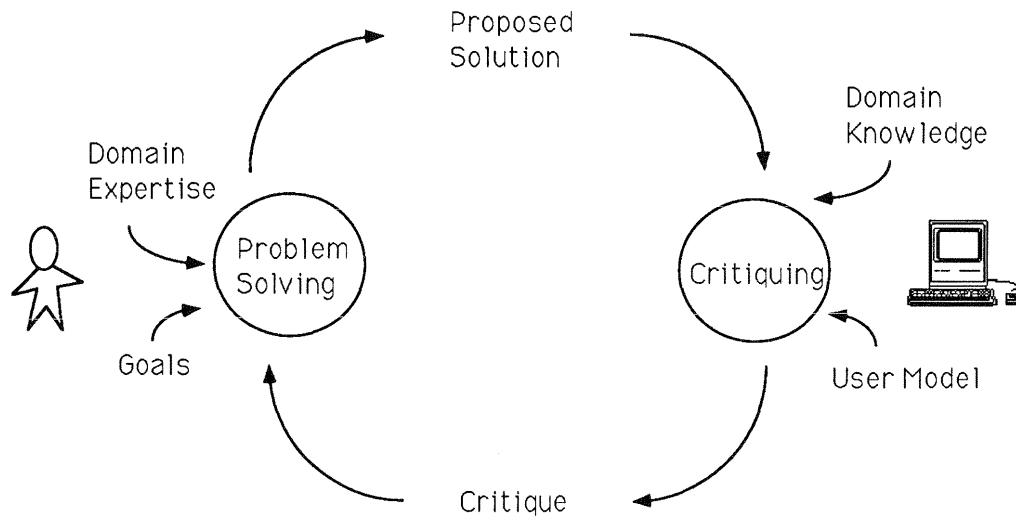
*Design environments* (Lemke, 1989; Fischer, McCall, & Morch, 1989a) are tools that foster human problem-domain communication by providing a set of building blocks that model a problem domain. Design environments also incorporate knowledge about which components fit together and how. These systems contain critics that recognize suboptimal design choices and inefficient or useless structures.

### Cooperative Problem Solving Systems

The goal of developing joint human-computer cognitive systems in which the computer is considered a cognitive amplifier has challenged the more widely understood goal of artificial intelligence: The understanding and building of autonomous, intelligent, thinking machines. For us, a more important goal is to understand and build interactive knowledge media (Stefik, 1986) or cooperative problem solving systems (Fischer, 1990). The major difference between classical expert systems, such as MYCIN and R1, and cooperative problem solving systems is in the respective roles of human and computer.

Traditional 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 and participates together with the system in the problem solving and decision making process. The precise roles played by the two parties may be chosen depending on their different strengths with respect to knowledge of goals and task domain. Critics are an important component of cooperative problem solving systems, especially when they are embedded in integrated design environments. These critics detect inferior designs, provide explanations and argumentation for their “opinion” and suggest alternative solutions.

Cooperative problem solving refers in our work to the cooperation between a human and a computer. It shares some research issues with two related but different research areas: *Computer Supported Cooperative Work*



**Figure 1: The Critiquing Approach**

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, while the computer's role is to analyze those solutions producing a critique for the human to apply in the next iteration of this process.

(CSCW) (Greif, 1988), which describes the cooperation between humans mediated by computer, and *Distributed Artificial Intelligence* (Bond & Grasser, 1988), which refers to cooperation between computer systems.

Traditional expert systems are inadequate in situations where it is difficult to capture all necessary domain knowledge. Leaving the human out of the decision process, autonomous expert systems require a comprehensive knowledge base covering all aspects of the tasks to be performed; all "intelligent" decisions are made by the computer. Some domains, such as user interface design, are not sufficiently understood, and creating a complete set of principles that adequately captures the domain knowledge is not possible. Other domains are so vast that tremendous effort is required to acquire all relevant knowledge. Critics are well suited to these situations because they need not be complete domain experts.

The traditional expert system approach is also inappropriate when the problem is ill-defined, that is, the problem cannot be precisely specified before a solution is attempted. In contrast, critics are able to function with only a partial task understanding.

## The Critiquing Approach

Critiquing is a way to present a reasoned opinion about a product or action (see Figure 1). The product may be a computer program, a kitchen design, a medical treatment plan; an action may be a sequence of keystrokes that cor-

rects a mistake in a word processor document or a sequence of operating system commands.<sup>1</sup> An agent, human or machine, that is capable of critiquing in this sense is classified as a critic. Critics often consist of a set of rules or specialists for individual aspects of a product; we sometimes refer to such an individual rule or specialist as a critic, not only to the complete critiquing system as a whole.

Critics do not necessarily solve problems for users. The core task of critics is the recognition and communication of deficiencies in a product to the user. Critics point out errors and suboptimal conditions that might otherwise remain undetected. Most critics make suggestions on how to improve the product. With this information users can fix the problems or seek additional advice or explanations.

Advisors (Carroll & McKendree, 1987) perform a function similar to critics except that they are the primary source for the solution. Users describe a problem, and they obtain a proposed solution from the advisor. In contrast to critics, advisors do not require users to present a partial or proposed solution to the problem.

Critiquing systems are particularly well suited for design tasks and for complex problem domains. In most

<sup>1</sup>In the remainder of the paper the term product is often used in a generic sense encompassing both product in a narrow sense and actions.

---

```

;; Stove should be away from a door
(define-crack-rule stove-door-rule stove
  "is not away from a door"
  "is away from a door"
  :argumentation-topic "answer (stove, door)"
  :apply-to (all door)
  :applicability
    (has-design-unit 'door)
  :condition
    (not (away-from du-1 du-2 threshold)))
; defined on design unit STOVE
; critic message
; praiser message
; access to JANUS-VIEWPOINTS
; test all doors (if any)
; test only if there is
; a door in the work area
; du-1 is stove, du-2 is door

```

**Figure 4:** Definition of the stove-door critic rule

---

cases design tasks are ill-structured problems for which no optimal solution exists. Complex problem domains require a team of cooperating specialists rather than a single expert. Not all problems fit this description; for example, there are problems in engineering design and operations research, where one can precisely specify problems and generate optimal solutions. Those types of problems yield to more algorithmic solutions and are not good candidates for the critiquing approach.

The term "critic" has been used to describe several closely related, yet different ideas. It was used first in planning systems to describe internal demons that check for consistency during plan generation. For example, critics in the HACKER system (Sussman, 1975) discover errors in blocks world programs. When a critic discovers a problem, it notifies the planner, which edits the program as directed by the critic. The NOAH system (Sacerdoti, 1975) 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.

### JANUS: An Example

To illustrate the critiquing approach and to provide an example for the subsequent theoretical discussion of critiquing, we present in this section the JANUS design environment. JANUS is based on the critiquing approach and allows a designer to construct residential kitchen floor plan layouts and to learn general principles underlying such constructions (Fischer, McCall & Morch, 1989a; Fischer, McCall & Morch, 1989b). JANUS is an integrated design environment addressing the challenges of human problem-domain communication and testing the feasibility of applying relevant information from large information stores to a designer's task.

JANUS contains two integrated subsystems: JANUS-CRACK and JANUS-VIEWPOINTS. JANUS-CRACK is a knowledge-based design environment supporting the construction of kitchens using domain-oriented building blocks called design units (Figure 2). JANUS-VIEWPOINTS is an issue-based hypertext system containing general prin-

ciples of kitchen design (Figure 3). The integration of JANUS-CRACK and JANUS-VIEWPOINTS allows argumentation to resolve problematic (breakdown) situations that occur when critics detect design rule violations.

### Knowledge Representation in JANUS

The critics in JANUS-CRACK know how to distinguish "good" designs from "bad" designs and can explain that knowledge. This knowledge includes design principles from Jones & Kapple (1984). These principles fall into three categories: building codes, such as "*The window area shall be at least 10% of the floor area.*", safety standards, such as "*The stove should be at least 12 inches away from a door.*", and functional preferences, such as "*The work triangle should be less than 23 feet.*" Functional preferences are soft rules and may vary from designer to designer. Building codes and safety standards are harder rules and should be violated only in exceptional cases.

The critics are implemented as condition-action rules, which are tested whenever the design is changed. The changes that trigger a critic are operations that modify the construction situation in the work area: *move*, *rotate*, and *scale*. Each type of design unit has a set of critic rules whose condition parts are relationships between design units that capture the design principles discussed above. Protocol studies have shown that they are important principles that professionals use during the design process (Fischer & Morch, 1988). The stove design unit, for example, has critic rules with the following condition parts: (*away-from stove door*), (*away-from stove window*), (*near stove sink*), (*near stove refrigerator*), (*not-immediately-next-to stove refrigerator*). The code for one of the stove critic rules is shown in Figure 4.

### JANUS as a Design/Learning Tool

JANUS supports two design methodologies: design by composition (using the *Palette*) and design by modification (using the *Catalog*). In addition, examples in the catalog can be used to support learning. The user can copy both good and bad examples into the work area. One learning example is shown in Figure 5. The system can critique such designs to show how they can be improved,

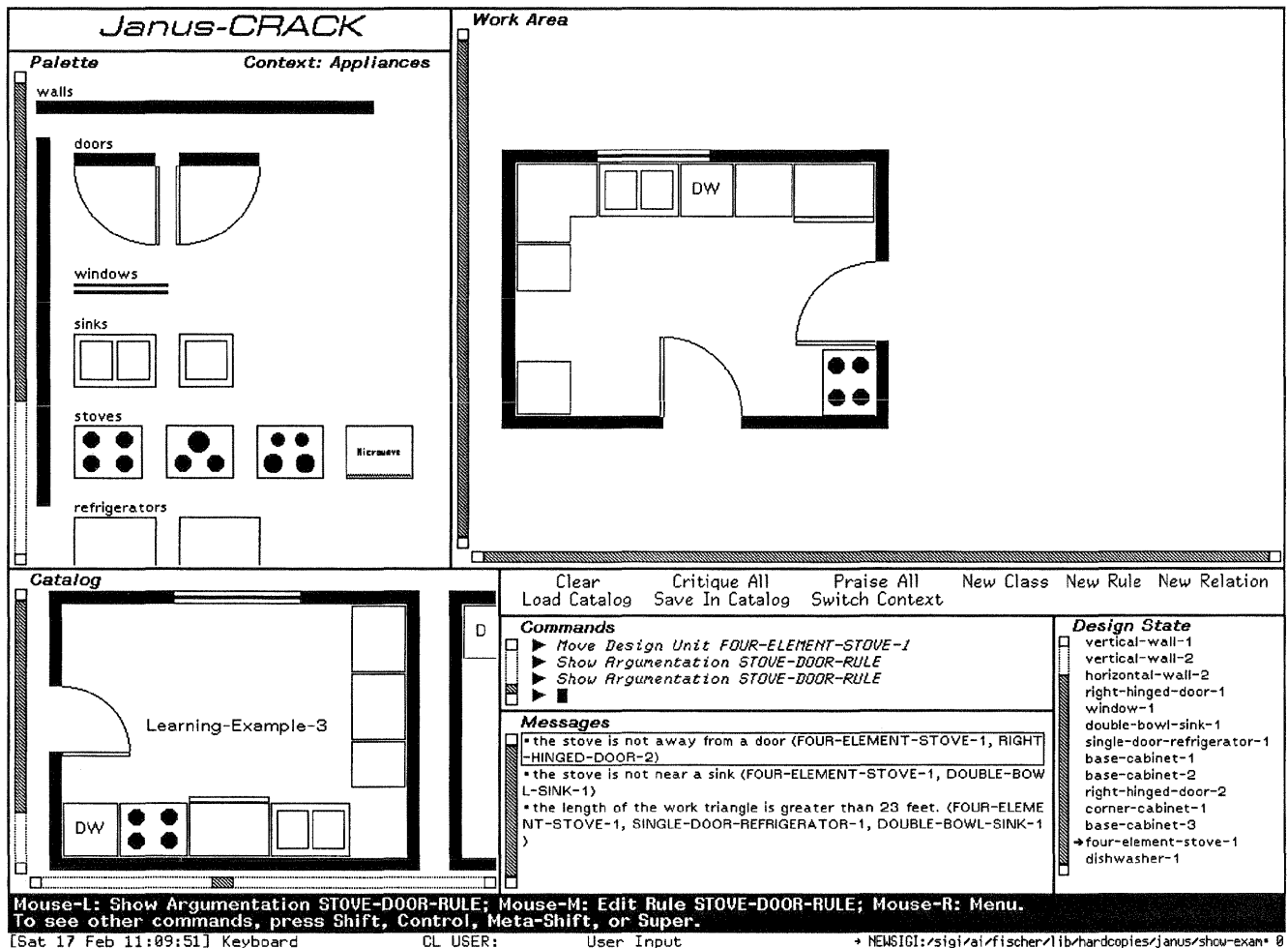


Figure 2: JANUS-CRACK: the STOVE-CRITIC

JANUS-CRACK 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-VIEWPOINTS and displays the argumentation related to that message (Figure 3).

thus allowing users to learn from negative examples. To learn about good features of prestored designs, designers can run the *Praise All* command, thus getting positive feedback as well. Users can add their own designs to the catalog for future reuse or as additional learning examples.

Users can modify and extend the JANUS-CRACK design environment by modifying or adding design units, critic rules, relationships (Fischer & Girgensohn, 1990). The ability to modify critic rules is important if a designer disagrees with the critique given. Standard building codes (hard rules) cannot be changed, but functional preferences (soft rules) vary from designer to designer and thus can and should be adapted. In this way, users have the capability to express their preferences. For example, if users disagree with the design principle that the stove should be away from a door, they can edit the stove door

rule by replacing the *away-from* relation between STOVE and DOOR with another relation (selected from a menu). After this modification, they will not be critiqued when a stove is not away from a door.

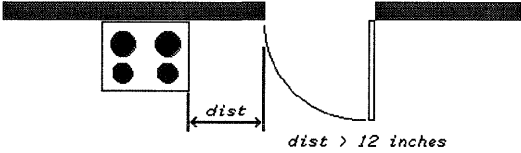
We have found JANUS to be a useful environment for design students. It teaches them about design principles. JANUS is also an efficient tool for skilled designers as it enhances their cognitive abilities for storing and remembering principles of good design such as the required building codes.

### A User Scenario with JANUS

In the following scenario, the designer has selected an L-shaped kitchen from the catalog for reuse. The goal is to modify this kitchen into a U-shaped kitchen by rear-

*Janus-ViewPoints*
*the stove is away from a door*

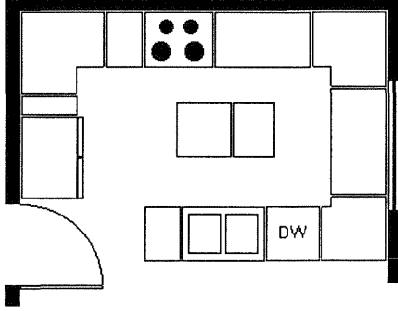
**Answer (Stove, Door)**  
The stove should be away from a door.



**Figure 5: stove-door**

**Argument (Fire Hazard)**  
By placing the stove too close to a door it will be a fire and burn hazard to unsuspected passers by (such as small children)

**Argument (Dining Room)**  
If the door leads into a dining room, it will be easy to bring hot food from the stove into the dining area!



**Visited Nodes**

- Answer (Refrigerator, Window) Section
- Description (Work Triangle) Section
- Answer (Refrigerator, Stove) Section
- Answer (Stove, Sink) Section
- ➔ Answer (Stove, Door) Section

**Viewer: Default Viewer**

---

**Commands**

- ▶ Show Example: "Answer (Stove, Door)"
- ▶ Show Example Answer (Stove, Door)

Show Outline

Search For Topics

Show Argumentation

Show Context

Done

Show Example

Show Counter Example

Show Construction

**Mouse-R: Menu.**  
To see other commands, press Shift, Control, Meta-Shift, or Super.

[Sat 17 Feb 11:09:04] Keyboard      CL USER:      User Input

**Figure 3: JANUS-VIEWPOINTS: Rationale for the stove-door rule**

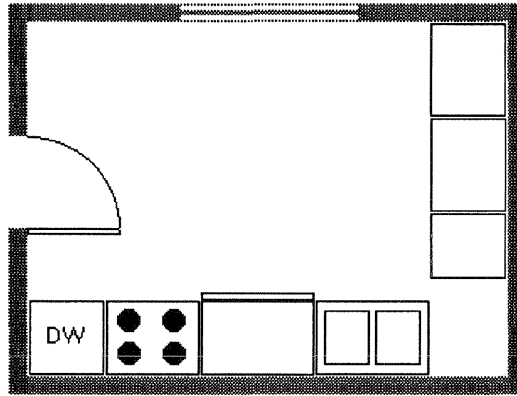
The JANUS argumentation component is a hypertext system implemented using the SYMBOLICS DOCUMENT EXAMINER. Clicking with the mouse on a critique in the JANUS construction mode (Figure 2) activates JANUS-VIEWPOINTS. The *Viewer* pane shows the arguments for and against the answer relating a stove and a door. The top right pane shows an example illustrating this answer. The *Visited Nodes* pane lists in sequential order the argumentation topics previously discussed. 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.

ranging some of the appliances and cabinets. Figure 2 shows the construction situation just after the stove (FOUR-ELEMENT-STOVE-1) was moved to the lower right corner of the kitchen floor plan.

Moving the stove triggers the stove critic, which tests the stove's location relative to the doors, sink, and refrigerator currently in the work area. Critic messages displayed in the messages pane tell the designer that the stove is not properly located relative to the door and the sink, and that the WORK TRIANGLE is greater than 23 feet. This identifies a problematic situation, and prompts the designer to reflect on it. The designer has broken a kitchen safety rule: The stove should be at least 12 inches away from a door.

The user may not have known the safety rule or may not understand the rationale for the rule, in which case an explanation is desirable. Instead of providing the designer with prestored text reflecting one expert's opinion, it is preferable to acquaint the designer with multiple perspectives. This is supported by JANUS-VIEWPOINTS, which is activated by clicking with the mouse on the critique in the messages pane. The designer enters JANUS-VIEWPOINTS automatically in the context relevant to the critique (Figure 3). This argumentative context shows an answer to the issue the designer is implicitly raising: "What should be the location of the stove?" The answer states how the stove should be positioned relative to a door and lists arguments for and against this answer. This argumen-





**Figure 5:** JANUS-CRACK: A learning example from the Catalog

The critics in JANUS detect the following suboptimal features of the kitchen shown in this figure: The width of the door is less than 36 inches, the dishwasher is not next to a sink, the stove is next to a refrigerator, the refrigerator is next to a sink, and the sink is not in front of a window.

tative context also shows an example of how this design rule was successfully applied in another design (shown in the upper right corner of Figure 3). The example is taken from the catalog of prestored designs in JANUS-CRACK.

All bold and italicized words in the Viewer pane (the largest pane in JANUS-VIEWPOINTS) and all topics in the *Bookmarks* pane allow further exploration with a mouse click. Hypertext access and navigation is made possible using this feature inherited from the SYMBOLICS DOCUMENT EXAMINER. After finishing the search for related information in JANUS-VIEWPOINTS, the designer resumes construction in JANUS-CRACK by selecting the Done command.

## The Process of Critiquing

Figure 6 illustrates the component processes of critiquing: goal acquisition, product analysis, critiquing strategy, explanation and advice giving. Not all of these processes are present in every critiquing system. This section describes these subprocesses and illustrates them with examples. JANUS does not illustrate all of the issues, and we will refer occasionally to systems that are described in Section 6.

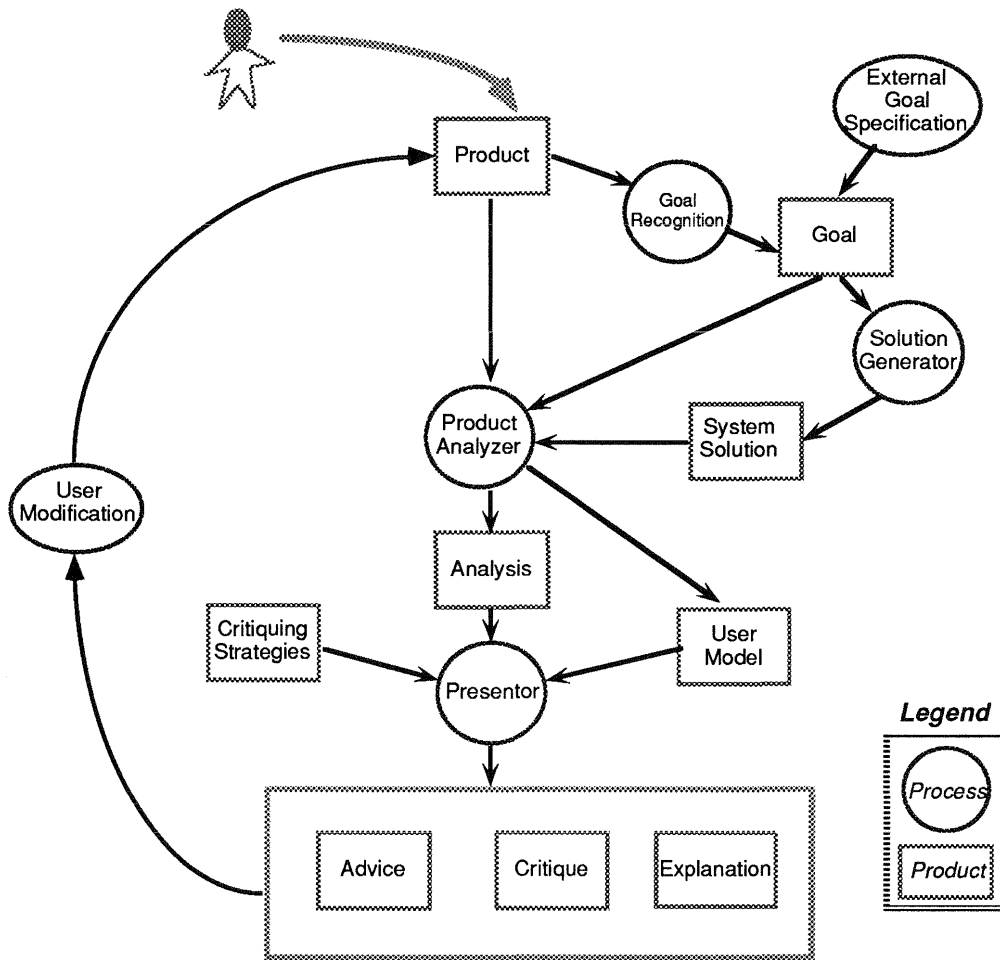
### Goal Acquisition

Critiquing a product is enhanced if the system has an understanding of the intended purpose of the product (problem knowledge). Problem knowledge can be separated into domain knowledge and goal knowledge. Domain knowledge without any understanding of the particular goals of the user restricts a critic to reasoning about characteristics that pertain to all products in the domain. For example, domain knowledge allows JANUS to point

out that stoves should not be placed close to doors, because this arrangement constitutes a fire hazard. For a more extensive evaluation of a product, some understanding of the user's specific goals and situation is necessary. JANUS, like most critics, does not have that understanding. The user's goal is assumed to be to design a functional residential kitchen; the critic does not take into account any individual requirements such as size of the kitchen or number of people in the family.

Critics that work with specific goal knowledge can acquire it by asking the user (external goal specification) or by analyzing the product the user has generated so far (goal recognition). A kitchen with a table and chairs located in the center of the kitchen suggests that the user intends to eat meals in the kitchen. The table and chairs allow a kitchen critic to recognize the goal of providing an eating area. Goal recognition is only possible if the current version of the artifact approximates a solution to the goal to be recognized. 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. Goal recognition is related to task-oriented parsing (Hoppe, 1988) and plan recognition, a research area in artificial intelligence (Schmidt, Sridharan & Goodson, 1978; London & Clancey, 1982; Carver, Lesser & McCue, 1984). Tutorial systems define a goal structure for the user. Critics, however, allow users to set their own goals and do not restrict the space of possible goals. This complicates the problem of goal recognition. A critic that implements goal recognition is ACTIVIST (Section 6.3).

A critic may also have access to an external specification of the problem to be solved. For example, users may communicate to the system that they need a kitchen with an eating area for informal meals. This can be done with



**Figure 6: 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 the user's goals either by recognizing them or from explicit user input. The product analyzer evaluates the product against the goal specification. Some critics do this by generating their own solution and comparing it to the user's. A presentation component uses the product analysis to formulate a critique, to give advice on how to make improvements, and to provide explanations. Critiquing strategies and a user model control the kind of critique, its form and timing. Based on the critique, the user generates a new version of the product, and the cycle repeats, integrating the new insight.

electronic questionnaires as well as with more sophisticated techniques such as natural language communication.

### Product Analysis

There are two general approaches to critiquing: *differential* and *analytical* critiquing. In the former approach, the system generates its own solution and compares it with the user's solution pointing out the differences. An advantage of differential critiquing is that all differences can be found. Some domains allow radically

different, but equally valid solutions, which 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 only say that the system solution achieves good results but cannot explain why the user's solution is less than optimal.

Different solution attempts fulfill the goals to different degrees or are associated with different undesirable effects. In such situations, *metrics* are needed to measure the quality of alternative solutions (Fischer, Lemke & Schwab, 1985). Based on the controversial nature of

design problems, alternative, conflicting metrics can be defined and may have to be reconciled by negotiation and argumentation.

An *analytical critic* checks products with respect to predefined features and effects. Analytical critics identify suboptimal features using pattern matching (e.g., Fischer, 1987), and expectation-based parsers (e.g., Finin, 1983). 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 between kitchen design units. JANUS does not identify all possible problems within a kitchen design. Its rule base allows it to critique kitchens without knowing exact requirements and preferences of the kitchen user.

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 (Fischer, Lemke & Schwab, 1985) is a critic for a text editor, which critiques keystroke sequences and, if possible, proposes shorter alternatives. Systems such as ACTIVIST 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 sequence needs to be recognized as an interleaved execution of a correct-spelling plan and an exchange-words plan. A critic capable of task-oriented parsing must decide how long to wait for later parts of a plan and whether interspersed actions interfere with the interrupted plan.

## Critiquing Strategies

Critiquing strategies and an optional 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 predominant use of the system, either to help users solve their problems or as a learning environment.

**The user's perception of critics.** Like recommendations from colleagues or co-workers, messages from a critic can be seen as helpful or hindering, as supportive of or interfering with work or 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. *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 humans if it is handled as a private matter between the human and the computer.

**What should be critiqued?** *Educational critics*, whose prime objective is to support learning, and *performance critics*, whose primary objective is to help produce better products, have different requirements for their critiquing strategies. A performance critic 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 interaction between user and critic. Educational critics should maximize the information users retain to improve their future performance.

Most performance critics (e.g., FRAMER, JANUS, ROUNDSMAN, KATE; see Section 6) 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 a policy specified by the user. 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, such as the WEST system by Burton & Brown, 1982 (see Section 6) usually employ a more complex intervention strategy that is designed to maximize information retention and motivation. For example, an educational critic may forego an opportunity to critique when it occurs directly after a previous critiquing episode.

Most existing critics operate in the *negative* mode, that is, they point out suboptimal aspects of the user's product or solution. A *positive* critic recognizes the good parts of a solution and informs users about them (the *Praise All* command in JANUS-CRACK). For performance critics, a positive critic helps users retain the good aspects of a product in further revisions; a positive educational critic reinforces the desired behavior and aids learning.

**Intervention strategies.** Intervention strategies determine when a critic should interrupt and how. *Active critics* exercise control over the intervention strategy by critiquing a product or action at an appropriate time. They function like active agents continuously monitoring users and responding 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.

For active critics the intervention strategy must specify when to send messages to the user. Intervening immediately after a suboptimal or unsatisfactory action has occurred (an immediate intervention strategy) has the advantage that the problem context is still active in the users' mind, and they remember how they arrived at the solution. The problem can often be corrected immediately. A disadvantage of active critics is that they may disrupt a cognitive process causing short term memory loss. Users then

need to reconstruct the goal structure that existed before the intervention. Delayed critic messages may appear out of context and hence come too late to prevent the user from heading towards an undesirable state.

Critics can use any of various intervention modes that differ in the degree to which users' attention is attracted. 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 whether to read and process the message immediately or first complete an action in progress. The messages should be displayed in such a way that they do not go unnoticed. Those messages that pertain to users' current focus of attention should be easy to find rather than being hidden among a large set of messages related to other aspects of the product.

### 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. A critic that constantly repeats an explanation that the user already knows is also unacceptable.

Critics can be adaptable or adaptive. Systems are called adaptable if the user can change the behavior of the system. 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.

User modeling in critics (Fischer, Lemke & Schwab, 1985) shares ideas and goals with student modeling in intelligent tutoring systems (Clancey, 1986) and advice giving natural language dialogue systems (Kobsa & Wahlster, 1989). Computer critics require dynamic, persistent user models that can change over time but are accessible to the human user for inspection and modification. How to acquire and represent individual user models is a topic of ongoing research (Mastaglio, 1990).

### Explanation Capability

Critics have to be able to explain the reasons for their interventions. This provides users with an opportunity to assess the critique and then to decide whether to accept it. Knowing why a product was critiqued helps users to learn the underlying principles and avoid similar problems in the future. In a critiquing system, explanations can be focused on the specific differences between the system's and the user's solutions, or on violations of general guidelines. Critics can either give detailed explanations spontaneously or provide them on demand. When users can indicate the issues they are interested in, the system can provide enhanced explanations on demand. One particular approach

uses argumentation as the fundamental structuring mechanism for explanations; this is illustrated in the JANUS-VIEWPOINTS system (Fischer, McCall & Morch, 1989).

### Advisory Capability

All critics detect suboptimal aspects of the user's product (*problem detection mode*). Some critics require the user to determine how to improve the product by making changes to address the problems pointed out by the critic. Other critics, however, are capable of suggesting alternatives to the user's solution. We call these *solution-generating* critics. In the JANUS system, a simple problem detecting critic points out that there is a stove close to a door. A solution-generating critic would, in addition, suggest a better location.

### Descriptions of Critics

The purpose of this section is to provide an overview of critiquing systems that have influenced the development of the paradigm or that illustrate an interesting aspect of it. We first describe in some detail two critic systems developed in our laboratory: LISP-CRITIC and FRAMER. After that, we survey systems developed by others.

#### LISP-CRITIC

LISP-CRITIC is a system designed to support programmers (Fischer, 1987; Fischer & Mastaglio, 1989). It helps its users to both improve the program they are creating and to acquire programming knowledge on demand. Programmers ask LISP-CRITIC for suggestions on how to improve their code. The system then suggests transformations that make the code more cognitively efficient (i.e., easier to read and maintain) or more machine efficient (i.e., faster or requiring less memory).

When LISP-CRITIC finds pieces of code that could be improved, it shows the user its recommendation (Figure 7). Users can accept the critic's suggestion, reject it or ask for an explanation to aid in making that decision. In Figure 7, LISP-CRITIC suggests that the user replace a conditional expression using *cond* with an expression using *if*. The user can request an explanation of why *if* is preferable to *cond*. The system develops an appropriate explanation, consulting 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). To adequately support a wide range of user expertise, LISP-CRITIC incorporates a user modeling component (Mastaglio, 1990). LISP-CRITIC uses the model to customize explanations so that they cover exactly what the user needs to know.

#### FRAMER

FRAMER (Lemke, 1989) is an innovative design environment for the design of program frameworks, components of window-based user interfaces on Symbolics

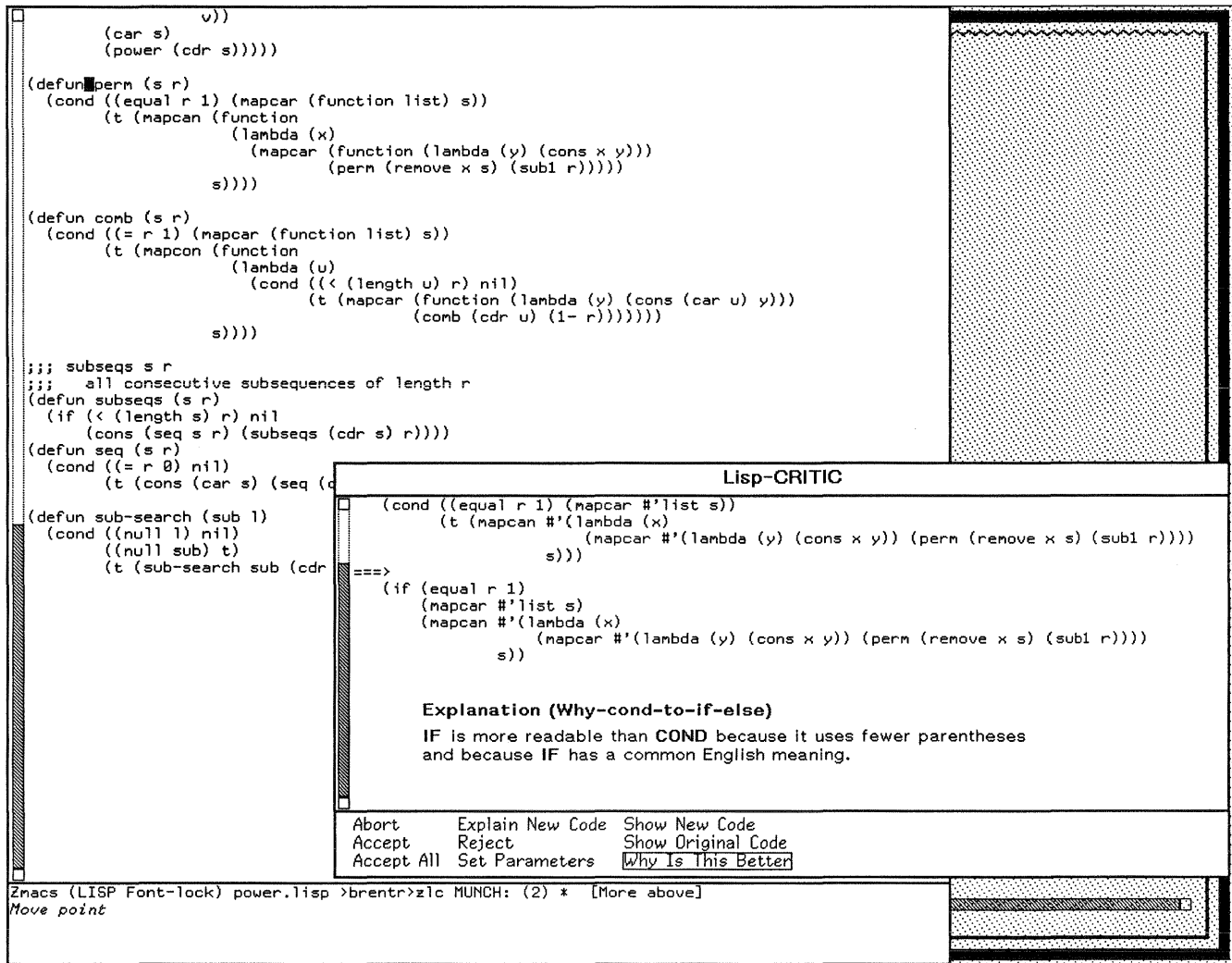


Figure 7: The User Interface of LISP-CRITIC

The large editor window shows a program that a user is working on. 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` function to an `if`.

LISP machines (Figure 8). The purpose of the FRAMER design environment is to enable designers to make use of a high-level abstraction — program frameworks — with little prior training.

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. The critics are either mandatory or optional. Mandatory critics represent absolute constraints that must be satisfied for program frameworks to function properly. Optional critics inform the user of issues that typically are dealt with differently. 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 why 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. Experimental evidence (Lemke, 1989) showed that users often invoked the critic too late when a major incorrect decision had already been made. The active strategy with continuous display of messages used in the newest version of FRAMER solved this problem. FRAMER prevents its users from permanently ignoring the critics by using the checklist. Checklist items

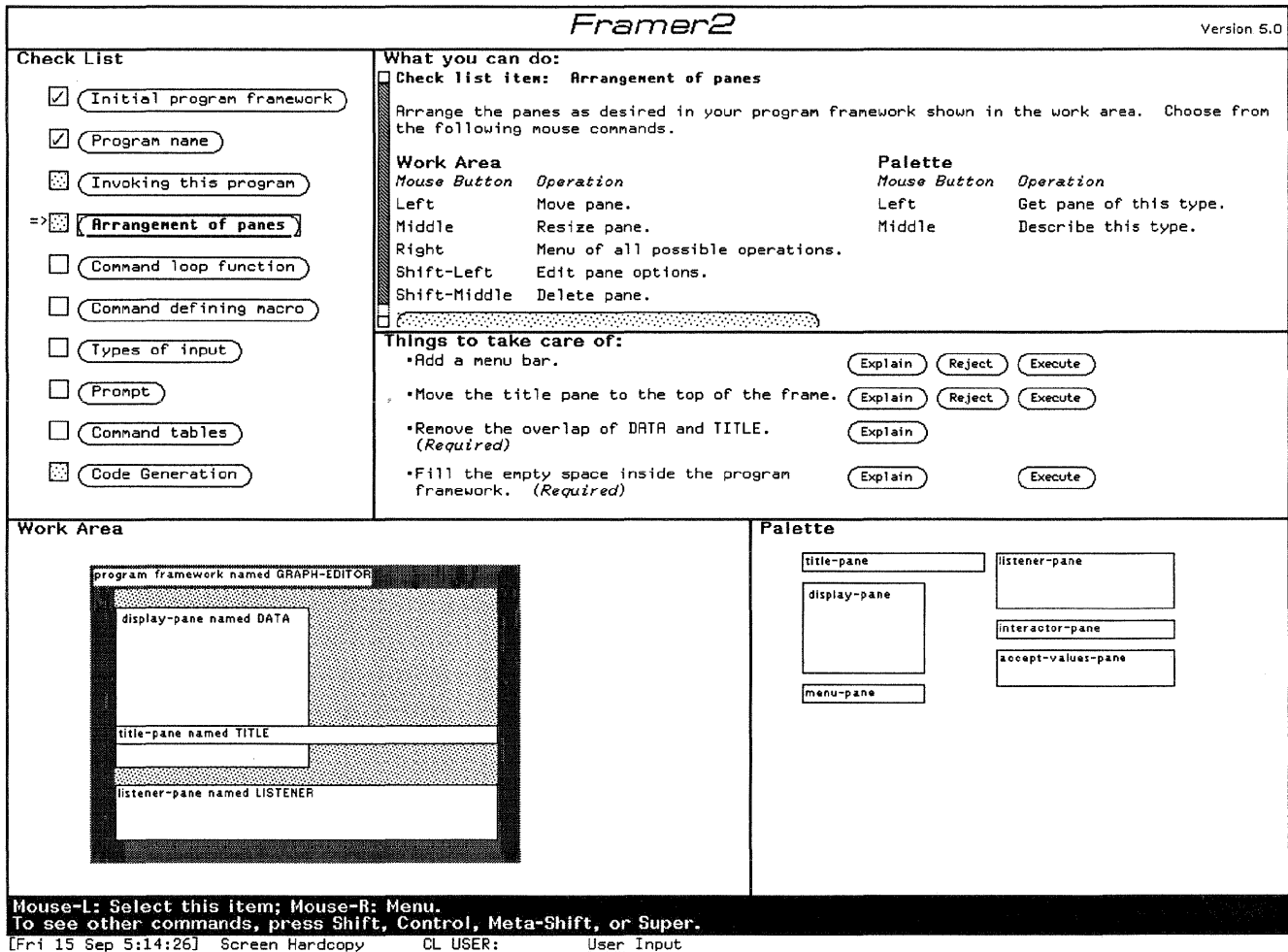


Figure 8: FRAMER

This figure shows a screen image of a session with FRAMER. The system has the following components. 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.

cannot be checked off until all suggestions are either resolved or rejected.

### Short Descriptions of Critics

What makes the critiquing approach attractive is that it has generality across a wide range of domains. The approach has been applied to the domains of medical diagnosis and patient management, electronic circuit design, learning environments, support of education programs, writing, programming, and text editing. This section offers short descriptions of critiquing systems from these domains. Most critics have been developed as research vehicles, but a few are successful commercial applications. We have tried to select those systems that have had sig-

nificant impact on the development of the critiquing paradigm and ones that have interesting features or address unique applications. We begin this section with a discussion of the WEST system because it pioneered fundamental ideas that the critiquing paradigm incorporates.

**WEST.** WEST was an early effort to build a computer coach or critic (Burton & Brown, 1982). WEST teaches arithmetic skill in a gaming environment (a game called "How the West was won"). Burton and Brown's goal was to augment an informal learning activity with a computer coach that would retain the engagement and excitement of a student directed activity. At the same time, the system was to provide context-sensitive advice on how to play better so students wouldn't get stuck at suboptimal

levels of playing.

Burton and Brown pioneered several important ideas in WEST. The computer coach builds a bridge between open learning environments and tutoring in order to support guided discovery learning. To prevent the coach from being too intrusive, the system constructs a model of each individual user. The system has diagnostic modeling strategies for inferring student problems from student behavior. WEST determines the causes of suboptimal behavior by comparing the concepts used in the solution of a built-in expert and those used in the user's solution. Explicit intervention and tutoring strategies (the most important one being "tutoring by issue and example") are explicitly represented in the system and operate based on the information contained in the model of the user. These knowledge structures enable the coach "to say the right thing at the right time" and to provide an overall coherence to the coach's comments.

Although WEST provides an early demonstration of how to construct an intelligent learning environment, its development pointed out certain limitations of the approach. The WEST system was a success in demonstrating the value of a computer coach in an informal learning activity. But the domain chosen for the system has a number of properties which do not hold for other domains in which critics are needed. The computer expert can play an optimal game (i.e., there is a best solution), and it can determine the *complete* range of alternative behaviors. In WEST one can speak of "bugs" whereas for many other domains one can only speak of "suboptimal" behavior. The metric to compute the best move is simple, whereas metrics in domains such as kitchen or software design involve a potentially large number of controversial issues. The set of issues for the game is closed whereas it is open-ended in many other domains. The goal of the user is obvious in WEST — to win the game while obeying its rules. This is another simplifying assumption which does not apply to many other domains. The explanation strategy in WEST relies on the assumption that the advice given is self-explanatory because it contains a good illustrating example. The existence of a best solution and no need for modifying the rules in the domain eliminates the need for an argumentation component such as the one found in JANUS.

**Medical applications of the critiquing approach.** Researchers in the domain of medicine developed several of the early critiquing systems. In general, these systems are designed to aid the physician in diagnosis and planning of patient treatment. Miller and colleagues at Yale Medical School have done a majority of the work in this area. Their systems assist a physician or nurse by analyzing plans for prescribing medication, managing the administration of medication, ventilator management, and administration of anesthetics (Miller, 1986).

Miller's ATTENDING system (Miller, 1986) uses the differential critiquing approach. ATTENDING parses the physician's plan into a hierarchical form. Starting from the top-level decisions, the system evaluates each step of

the physician's plan by trying to find alternatives associated with lower or equal risks to the patient. This method provides a more reasonable critique than one that discards the physician's solution and proposes a completely new solution developed by the system expert. By working from the physician's solution, the system's solution is as close to the physician's solution as possible, and the critique is more helpful and easier to understand.

The differential critiquing approach is also used in one version of ONCOCIN, an expert system for cancer therapy (Langlotz & Shortliffe, 1983). The critiquing approach was chosen because it eliminates the need for the physician to override the system solution when minor deviations in the therapy are desired for the convenience of the patient.

The ROUNDSMAN system (Rennels, 1987; Rennels, Shortliffe, Stockdale & Miller, 1989) is a critic in the domain of breast cancer treatment. ROUNDSMAN explicitly bases its critique on studies from the medical literature. It is a passive critic with explicit goal specification. ROUNDSMAN automatically provides a detailed explanation of its reasoning and suggests improvements to the physician's therapy proposal. It does not use a user or dialog model and, therefore, repeats similar explanations.

**Circuit design.** CRITTER (Kelly, 1985) is a design aid in the domain of digital circuit design. CRITTER requires a schematic diagram of the circuit and a set of specifications that the circuit must satisfy. Given this information, Critter produces a report about the circuit, which can be used in a subsequent design cycle to revise the design. CRITTER evaluates the circuit using various circuit analysis techniques and knowledge of the primitive components. The results of this evaluation include general information about whether the circuit will work and by what margins, its weaknesses, and performance limits.

NCR developed the Design Advisor (TM) (Steele, 1988), a commercial expert system that provides advice on application-specific, integrated circuit designs. The system uses a logic-based rule mechanism (Horn clauses) including both forward and backward chaining rules with a frame based representation and a truth maintenance system. The Design Advisor analyzes the performance, testability, manufacturability, and overall quality of CMOS semi-custom VLSI designs. The knowledge it applies is a hierarchy of design attributes compiled by studying major problems from several years of commercial VLSI design. The designer submits a proposed design to the system for analysis and critiquing using a batch type approach. The system returns its analysis to the designer who is responsible for actually modifying the proposed design.

**Discovery learning.** A suite of three computer-based coaching systems for discovery learning developed at LRDC, University of Pittsburgh, are based on critics. These systems each address a different domain: Smithtown — microeconomics (Raghaven, Schultz, Glaser & Schauble), Voltaville — direct current electricity (Glaser, Raghavan & Schauble, 1988), and Refract — geometrical optics (Reimann, Raghaven, Glaser 1988). These dis-

covery environments are designed to build scientific inquiry skills. Active critics judge the efficiency of the processes used to build scientific theory and inform users about errors that characteristically trap less successful students as well as guide them to effective strategies.

**Decision making.** The DecisionLab system from the European Computer Industry Research Center (Schiff & Kandler, 1988) applies the critiquing approach to coach users in managerial decision making. DecisionLab promotes "learning while doing" by providing constructive feedback on a user developed management plan in a simulated management game. The system critiques decisions and informs users when they pursue a non optimal approach. This system is of interest because it attempts to integrate a critic with a simulation exercise in a computational environment designed to support learning.

Another direction of research is investigating how to apply the critiquing approach to improve the performance of decision makers, not through training, but in the context of their actual work. Mili (1988) has proposed a system called DECAD, which watches over the shoulder of the decision maker, interjecting advice or a critique when appropriate. Critiquing is one of the approaches under investigation for a class of knowledge-based systems called "active and symbiotic decision support systems" (Mili & Manheim, 1988).

**Curriculum development.** The Alberta Research Council (Canada) and a company called Computer Based Training Systems have developed a knowledge-based system which provides assistance to teachers who do curriculum and course development (Wipond & Jones, 1988). The system includes an Expert monitor module that monitors the curriculum and course development process, intervening when necessary or when the teacher asks for assistance. It provides a critique which the user can accept or reject. The expert monitor is also capable of suggesting what the user should do next, where to look for relevant examples or how to get more help. Computer Based Training Systems is now marketing the product as part of their Computer Managed Learning software package.

**Writing.** WANDAH (Friedman, 1987) is a system that assists authors in all phases of writing. This system is commercially available for personal computers as 'HBJ Writer.' WANDAH has heuristics to help the user design and prepare a document. Once a writer has created some text (it need not be a completed document), it can be subjected to one of four sets of reviewing and revising aids. These aids go over the written work, provide feedback on structural problems, and recommend revisions. Testing of WANDAH showed that users find it easy to use and learn to write better.

**Text editing.** ACTIVIST (Fischer, Lemke & Schwab, 1985) is an active help system for a screen-oriented editor. ACTIVIST continuously monitors the editing actions of the user. By doing this, it recognizes, using finite state machines, sequences of actions that achieve some goal known to the system. ACTIVIST understands twenty different goals, such as deleting a word or moving the cursor

to the end of the current line. The system evaluates each recognized action sequence to update a user model. ACTIVIST uses the following critiquing strategy: 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. After a certain number of correct executions, the plan will no longer be watched. In order to be less intrusive, ACTIVIST ceases to critique actions when the user ignores its suggestions.

**Operating system usage.** WIZARD (Finin, 1983) is an active help system for users of the VMS operating system command language. Like ACTIVIST, WIZARD has to recognize sequences of commands that, taken together, form a plan to achieve a goal known to the system. The expectation-based parser used for this purpose allows non-contiguous command sequences that contain interspersed commands from other goals. Metrics such as amount of typing and use of system resources are implicit in the representations of the plans. Advice is given using text templates.

**Programming.** PROLOG Explaining (Coombs & Alty, 1984) critiques a user's explanation of PROLOG code. The system uses the critiquing approach to guide the user toward a better understanding of the PROLOG language. Users construct an explanation of what they believe the code will do. The system's job is to critique that explanation. The user may request a critique at any point during the process of the explanation building. Also, the system automatically critiques the explanation at the end of a program run.

The GRACE Project at the NYNEX Artificial Intelligence Laboratory (Dews, 1989; Atwood, Gray, Burns & Morch, 1990) has developed an integrated learning environment for COBOL programming. The GRACE system combines a tutor, a critic, and a hypertext system to support a spectrum of teaching methods ranging from guided teaching by the tutor, to integrating working and learning with the critic, and to exploratory browsing in hypertext. 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 directly accessible hypermedia documentation with text, graphics, and other presentation media.

**Software engineering.** KATE (Fickas & Nagarajan, 1988) critiques software specifications (for automated library systems) represented in an extended Petri net notation. The knowledge of the critic is represented as "cases" that consist of a pattern describing a behavior in a specification, links to one or more goals, simulation scenarios, and canned text descriptions. The critic evaluates the specification with respect to goals or policy values given by the user. The simulation scenarios back up the system's critique and are designed to approximate the rich set of examples that software professionals have been found to use.

**Mechanical design.** STEAMER/Feedback Mini-Lab



(Forbus, 1984) is an environment in which simulated devices, such as steam plant controllers, can be assembled and tested. A device is assembled from simple building blocks such as actuators and comparators. The Mini-Lab is able to generate code from the building block specifications to produce a simulation program for the device. After students have constructed their device, they can ask for a critique by the system. This critique identifies common bugs and recognizes some instances of known devices.

## Conclusion

The systems described in this paper show that critiquing is an emerging paradigm for knowledge-based systems. 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 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 (e.g., Schoen, 1983; Suchman, 1987; Bodker, Knudsen, Kyng, Ehn & Madsen, 1988; McCall, Fischer & Morch, 1989), 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 allow "the situation to talk back" through critics that point out breakdowns. 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" (Bodker et al., 1988). Critics help inexperienced users to become lay designers; for experienced users, they serve as reminders of the principles of good design.

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 that users pursue. This limitation restricts the amount of assistance and detailed goal-oriented analysis that critics can provide, in contrast to systems that have a deep understanding of a very small set of problems (for example, Johnson & Soloway, 1984).

Critics need inspectable knowledge structures so that users can understand, modify, and augment them (Fischer & Girgensohn, 1990). This modification should not require users to possess detailed programming knowledge. Users should be able to deactivate and reactivate individual critics according to their needs and their goals. With sufficient inference and user modeling capabilities, systems can adapt themselves dynamically (ACTIVIST con-

tains a mechanism to support this).

Observing users of JANUS and FRAMER showed that users do not always notice the critique 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 are not interrupted in situations where they should focus on other issues.

Currently, most critics support only "one-shot dialogs" (Aaronson & Carroll, 1987). They respond to actions taken by the user; they give suggestions and provide explanations and argumentation. But human critiquing is a more cooperative problem solving activity, during which an increased understanding of the problem develops.

We have attempted to provide answers to some of these issues by presenting the critiquing paradigm as an alternative approach to using knowledge-based computer systems to support human work and learning. Existing critiquing systems were surveyed. Critics are not the only solution to building better knowledge-based systems, but we believe that a growing number of them 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 towards the creation of more useful and more usable computer systems for the future.

## Acknowledgments

Many people have contributed 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), the members of the LISP-CRITIC project (Heinz-Dieter Boecker, Chris Morel, Brent Reeves, and John Rieman), all the people who have participated in discussions about the general framework for critiquing (Thomas Schwab, Helga Nieper-Lemke, Curt Stevens, Tom DiPersio, and Hal Eden), and the HCC research group as a whole. This research was partially supported by grant No. IRI-8722792 from the National Science Foundation, grant No. MDA903-86-C0143 from the Army Research Institute, and grants from the Intelligent Interfaces Group at NYNEX and from Software Research Associates (SRA), Tokyo.

## References

- Aaronson, A. & Carroll, J. M. (1987). Intelligent Help in a One-Shot Dialog: A Protocol Study. *Human Factors in Computing Systems and Graphics Interface, CHI+GI'87 Conference Proceedings (Toronto, Canada)*, 163-168. New York: ACM.
- Anderson, J. R. & Reiser, B. J. (1985). The LISP Tutor. *BYTE*, 10(4), 159-175.
- Atwood, M. E., Gray, W. D., Burns, B., Morch, A. I. & Radlinski, B. (1990). Cooperative Learning and Cooperative Problem Solving: The Case of Grace.

- Working Notes, 1990 AAAI Spring Symposium on Knowledge-Based Human-Computer Communication*, 6-10. Menlo Park, CA: AAAI.
- Bodker, S., Knudsen, J. L., Kyng, M., Ehn, P. & Madsen, K. H. (1988). Computer Support for Cooperative Design. *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'88)*, 377-394. New York: ACM.
- Bond, A. H. & Gasser, L. (Eds.). (1988). *Readings in Distributed Artificial Intelligence*. San Mateo, CA: Morgan Kaufmann Publishers.
- Burton, R. R. & Brown, J. S. (1982). An Investigation of Computer Coaching for Informal Learning Activities: In Sleeman, D. H. & Brown, J. S. (Eds.), *Intelligent Tutoring Systems* (pp. 79-98). London - New York: Academic Press.
- Carroll, J. M. & McKendree, J. (1987). Interface Design Issues for Advice-Giving Expert Systems. *Communications of the ACM*, 30(1), 14-31.
- Carver, N. F., Lesser, V. R. & McCue, D. L. (1984). Focusing in Plan Recognition. *Proceedings of AAAI-84, Forth National Conference on Artificial Intelligence (Austin, TX)*, 42-48. Los Altos, CA: William Kaufmann.
- Clancey, W. J. (1986). Qualitative Student Models. *Annual Review of Computing Science*, 1, 381-450.
- Coombs, M. J. & Alty, J. L. (1984). Expert Systems: An Alternative Paradigm. *International Journal of Man-Machine Studies*, 20.
- Dews, S. (1989). Developing an ITS in a Corporate Setting. *Proceedings of the 33rd Annual Meeting of the Human Factors Society*, 1339-1342.
- Draper, S. W. (1984). The Nature of Expertise in UNIX. *Proceedings of INTERACT'84, IFIP Conference on Human-Computer Interaction*, 182-186. Amsterdam: Elsevier Science Publishers.
- Fickas, S. & Nagarajan, P. (1988). Critiquing Software Specifications. *IEEE Software*, 5(6), 37-47.
- Finin, T. W. (1983). Providing Help and Advice in Task Oriented Systems. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, 176-178.
- Fischer, G. (1987). A Critic for LISP. *Proceedings of the 10th International Joint Conference on Artificial Intelligence (Milan, Italy)*, 177-184. Los Altos, CA: Morgan Kaufmann Publishers.
- Fischer, G. (1990). Communications Requirements for Cooperative Problem Solving Systems. *The International Journal of Information Systems (Special Issue on Knowledge Engineering)*.
- Fischer, G. & Girgensohn, A. (1990). End-User Modifiability in Design Environments. *Human Factors in Computing Systems, CHI'90 Conference Proceedings (Seattle, WA)*, 183-191. New York: ACM.
- Fischer, G. & Lemke, A. C. (1988). Construction Kits and Design Environments: Steps Toward Human Problem-Domain Communication. *Human-Computer Interaction*, 3(3), 179-222.
- Fischer, G., Lemke, A. C. & Schwab, T. (1985). Knowledge-Based Help Systems. *Human Factors in Computing Systems, CHI'85 Conference Proceedings (San Francisco, CA)*, 161-167. New York: ACM.
- Fischer, G. & Mastaglio, T. (1989). Computer-Based Critics. *Proceedings of the 22nd Annual Hawaii Conference on System Sciences, Vol. III: Decision Support and Knowledge Based Systems Track*, 427-436. IEEE Computer Society.
- Fischer, G., McCall, R. & Morch, A. (1989a). Design Environments for Constructive and Argumentative Design. *Human Factors in Computing Systems, CHI'89 Conference Proceedings (Austin, TX)*, 269-275. New York: ACM.
- Fischer, G., McCall, R. & Morch, A. (1989b). JANUS: Integrating Hypertext with a Knowledge-Based Design Environment. *Proceedings of Hypertext'89*, 105-117. New York: ACM.
- Fischer, G. & Morch, A. (1988). CRACK: A Critiquing Approach to Cooperative Kitchen Design. *Proceedings of the International Conference on Intelligent Tutoring Systems (Montreal, Canada)*, 176-185. New York: ACM.
- Forbus, K. (1984). *An Interactive Laboratory for Teaching Control System Concepts* (Report 5511). Cambridge, MA: BBN.
- Friedman, M. P. (1987). WANDAH - A Computerized Writer's Aid: In Berger, D. E., Pezdek, K. & Banks, W. P. (Eds.), *Applications of Cognitive Psychology, Problem Solving, Education and Computing* (pp. 219-225). Hillsdale, NJ: Lawrence Erlbaum Associates.
- Glaser, R., Raghavan, K. & Schauble, L. (1988). Voltaville: A Discovery Environment to Explore the Laws of DC Circuits. *Proceedings of the International Conference on Intelligent Tutoring Systems (Montreal, Canada)*, 61-66.
- Greif, I. (Ed.). (1988). *Computer-Supported Cooperative Work: A Book of Readings*. San Mateo, CA: Morgan Kaufmann Publishers.
- Hoppe, H. U. (1988). Task-Oriented Parsing: A Diagnostic Method to be Used by Adaptive Systems. *Human Factors in Computing Systems, CHI'88 Conference Proceedings (Washington, DC)*, 241-247. New York: ACM.
- Johnson, W. L. & Soloway, E. (1984). PROUST: Knowledge-Based Program Understanding. *Proceedings of the 7th International Conference on Software Engineering (Orlando, FL)*, 369-380. Los Angeles, CA: IEEE Computer Society.
- Jones, R. J. & Kapple, W. H. (1984). *Kitchen Planning Principles - Equipment - Appliances*.

- Urbana-Champaign, IL: Small Homes Council - Building Research Council, University of Illinois.
- Kelly, V. E. (1985). The CRITTER System: Automated Critiquing of Digital Circuit Designs. *Proceedings of the 21st Design Automation Conference*, 419-425.
- Kobsa, A. & Wahlster, W. (Eds.). (1989). *User Models in Dialog Systems*. New York: Springer-Verlag.
- Langlotz, C. P. & Shortliffe, E. H. (1983). Adapting a Consultation System to Critique User Plans. *Int. J. Man-Machine Studies*, 19, 479-496.
- Lemke, A. C. (1989). *Design Environments for High-Functionality Computer Systems*. Unpublished doctoral dissertation, Boulder, CO: Department of Computer Science, University of Colorado.
- London, B. & Clancey, W. J. (1982). Plan Recognition Strategies in Student Modeling: Prediction and Description. *Proceedings of AAAI-82, Second National Conference on Artificial Intelligence (Pittsburgh, PA)*, 335-338.
- Mastaglio, T. (1990). User Modelling in Computer-Based Critics. *Proceedings of the 23rd Hawaii International Conference on System Sciences, Vol III: Decision Support and Knowledge Based Systems Track*, 403-412. IEEE Computer Society.
- McCall, R., Fischer, G. & Morch, A. (1989). Supporting Reflection-in-Action in the Janus Design Environment. *Proceedings of the CAAD Futures '89 Conference*. Cambridge: Harvard University. Pre-Publication Edition.
- Mili, F. (1988). A Framework for a Decision Critic and Advisor. *Proceedings of the 21st Hawaii International Conference on System Sciences*, 381-386.
- Mili, F. & Manheim, M. L. (1988). And What Did Your DSS Have to Say About That: Introduction to the DSS Minitrack on Active and Symbiotic Systems. *Proceedings of the 21st Hawaii International Conference on System Sciences*, 1-2.
- Miller, P. (1986). *Expert Critiquing Systems: Practice-Based Medical Consultation by Computer*. New York - Berlin: Springer-Verlag.
- Papert, S. (1980). *Mindstorms: Children, Computers and Powerful Ideas*. New York: Basic Books.
- Raghaven, K., Schultz, J., Glaser, R. & Schauble, L. (1990). *A Computer Coach for Inquiry Skills*. Unpublished.draft submission to Intelligent Learning Environments Journal.
- Rennels, G. D. (1987). *A computational model of reasoning from the clinical literature*. Springer Verlag.
- Rennels, G. D., Shortliffe, E. H., Stockdale, F. E. & Miller, P. L. (1989). A computational model of reasoning from the clinical literature. *AI Magazine*, 10(1), 49-56.
- Riemann, P., Raghaven, K. & Glaser, R. (1988). *Refract, a Discovery Environment for Geometrical Optics* (Technical Report). Learning Research & Development Center, University of Pittsburgh.
- Sacerdoti, E. D. (1975). *A Structure for Plans and Behavior* (Technical Note 109). Stanford, CA: Stanford Research Institute.
- Schiff, J. & Kandler, J. (1988). Decisionlab: A System Designed for User Coaching in Managerial Decision Support. *Proceedings of the International Conference on Intelligent Tutoring Systems (Montreal, Canada)*, 154-161.
- Schmidt, C. F., Sridharan, N. S. & Goodson, J. L. (1978). The Plan Recognition Problem: An Intersection of Psychology and Artificial Intelligence. *Artificial Intelligence*, 11, 45-83.
- Schoen, D. A. (1983). *The Reflective Practitioner: How Professionals Think in Action*. New York: Basic Books.
- Simon, H. A. (1986). Whether Software Engineering Needs to Be Artificially Intelligent. *IEEE Transactions on Software Engineering, SE-12(7)*, 726-732.
- Steele, R. L. (1988). Cell-Based VLSI Design Advice Using Default Reasoning. *Proceedings of 3rd Annual Rocky Mountain Conference on AI*, 66-74. Denver, CO: Rocky Mountain Society for Artificial Intelligence.
- Stefik, M. J. (1986). The Next Knowledge Medium. *AI Magazine*, 7(1), 34-46.
- Suchman, L. A. (1987). *Plans and Situated Actions*. New York: Cambridge University Press.
- Sussman, G. J. (1975). *A Computer Model of Skill Acquisition*. New York: American Elsevier.
- Wenger, E. (1987). *Artificial Intelligence and Tutoring Systems*. Los Altos, CA: Morgan Kaufmann Publishers.
- Wipond, K. & Jones, M. (1988). Curriculum and Knowledge Representation in a Knowledge-Based System for Curriculum Development. *Proceedings of the International Conference on Intelligent Tutoring Systems (Montreal, Canada)*, 97-102.