COMPUTER-BASED CRITICS

Gerhard Fischer
Thomas Mastaglio

Reprinted from PROCEEDINGS OF THE TWENTY-SECOND ANNUAL
HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCE
Kailua-Kona, Hawaii, January 3-6, 1989

The Computer Society of the IEEE
1730 Massachusetts Avenue NW
Washington, DC 20036-1903

Washington • Los Alamitos • Brussels

IEEE

COMPUTER
SOCIETY
PRESS

Computer-Based Critics

Gerhard Fischer and Thomas Mastaglio

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

ABSTRACT

The computer-based critic is a paradigm for intelligent human-computer communication that overcomes some limitations of other approaches such as tutoring and advising. Critics are much more user-centered and support users working on their own activities. They provide information only when it is relevant. They allow users to do what they want and interrupt only when users' plans, actions, or products are considered significantly inferior. They are applicable to tasks in which users have some basic competence because users must be able to generate a plan, action, or product by themselves. They are most useful when no unique best solution exists in a domain and trade-offs have to be carefully balanced. Critics need to be knowledge-based. They must incorporate knowledge about the application domain, support explanation, model individual users, and provide innovative user interfaces. Over the last few years we have implemented a number of critics in different domains, including programming and design. The rationale, design, and evaluation of these systems is described as a starting point for a general framework for computer-based critics.

INTRODUCTION

Our goal is to establish the conceptual foundations for using the computational power that is or will be available on computer systems. We believe that artificial intelligence technologies can improve productivity by addressing, rather than ignoring, human needs and potential. In the spirit of Einstein's remark *"My pencil is cleverer than I"*, we are building systems that *augment human intelligence* -- in other words, we are building "systems for experts, not expert systems." Winograd and Flores [28] argue that the development of *tools for conversation*, the computer serving as a structured dynamic medium for conversation in systematic domains, is a more realistic and relevant way of exploiting information and communication technologies than is the most widely perceived goal of artificial intelligence, *"to understand and to build autonomous, intelligent, thinking machines"* [25].

We have used "intelligent support systems" as a generic name for systems that augment human capabilities. High functionality computer systems, such as UNIX or LISP machines which contain tens of thousands of objects and tools, have been the major application domain of our intelligent support systems. Our goal is to make usable the total space of functionality that computational environments have rather than diluting it or orienting the user toward only a subset of the system's capabilities. Intelligent support systems should facilitate access, application of knowledge, and learning. We have constructed a number of different intelligent support systems: documentation systems [14], active and passive help systems [12], design environments [11], and critics [7, 13], which we focus on in this paper. All of these systems have two things in common: they are knowledge-based and they use innovative techniques in human-computer communication.

In this paper we describe computer-based critics and articulate some of the general principles learned from our system-building experience. We propose a general framework for critics, present specific requirements, and describe two prototypical critic systems: LISP-CRITIC, which criticizes LISP programs, and CRACK, a system that assists the user in designing a kitchen. Then we illustrate the generalized main components of our critic systems and discusses their evaluation. We conclude with some plans for future work.

A Characterization of the Critic Paradigm

The computer-based critic is a useful and attractive approach for applying techniques from both human-computer communication and artificial intelligence research. Computer-based critics allow the potential of humans and computers to combine in a symbiotic system, that is, a successful combination of human skills and computing power to carry out a task that cannot be done either by the human or by the computer alone. Underlying symbiotic systems is acknowledgment of the fact that most knowledge-based systems are intended to assist human endeavor and that only a few are intended to be autonomous agents. Therefore, a subsystem supporting human-computer interaction is an absolute necessity. By using the capabilities of a knowledge-based architecture and innovative approaches to human-computer communication, critics allow users to remain in control and to solve problems they themselves want to work on, and yet critics support learning opportunities as well.

Intelligent Support Systems

Empirical investigations [6, 12] have shown that habitually only a small fraction of the functionality of complex systems such as UNIX, EMACS and LISP is used. Consequently it is of little use to equip modern computer systems with more and more computational power and functionality, unless we can help the user take advantage of them. The "intelligence" of a complex computer system must therefore be made to contribute to its ease of use and to provide effective communication, just as truly intelligent and knowledgeable human communicators, such as good teachers, use a substantial part of their knowledge to explain their expertise to others.

It is not sufficient for intelligent support systems just to solve a problem or provide information. The user must be able to understand the systems and and question their advice. One of our assumptions is that learners and practitioners will not ask a computer program for advice if they have no way of examining the program's expertise. Users must be able to access the system's knowledge base and reasoning processes. Domain knowledge has to be explainable.

Cooperative Problem Solving in Critic Systems

One model frequently used in human-computer systems (e.g., MYCIN [3]) is the *consultation model*. From an engineering point of view, it has the advantage of being clear and simple:

the program controls the dialogue, much as a human consultant does, by asking for specific items of data about the problem at hand. It precludes the user volunteering what he or she might think is relevant data. The program is viewed as an "all-knowing expert", and the user is left in the undesirable position of asking a machine for help.

The critiquing model supports cooperative problem solving. When a novice and an expert communicate, much more goes on than just the request for factual information. Novices may not be able to articulate their questions without the help of the expert, the advice given by the expert may not be understood, and the novice may request an explanation; each communication partner may hypothesize that the other has misunderstood, or the expert may give unsolicited advice, a phenomena we have explored in our work on *active help systems* [12]. Our systems should capture the essence of this human-to-human process. Critics are designed to incorporate as much of this process as possible.

Individualizing Computer Systems

User-centered learning. User-centered learning allows individuals to follow different learning paths. Forcing the same intellectual style on every individual is possibly much more damaging than forcing right-handedness upon a left-hander. To support user-centered learning processes, computational environments have to adapt to individual needs and learning styles. Giving users control over their learning and work requires them to initiate actions and set their own goals. Critics require individualized knowledge structures to support differential descriptions. They can use them to present explanations which represent new concepts in relation to knowledge previously held by specific users.

Incremental learning. Not even experts can completely master complex, high-functionality computer systems. Support for incremental learning is required. Incremental learning eliminates suboptimal behavior (thereby increasing efficiency), enlarges possibilities (thereby increasing functionality), supports learning on demand by presentation of new information when it is relevant, uses models of the user to make systems more responsive to the needs of individuals, and tailors explanations to the user's conceptualization of the task.

Learning on Demand. The major justification for learning on demand is that education is a distributed, lifelong process of learning material as it is needed. Learning on demand has been successful in human societies when learners can afford the luxury of a personal coach or critic. Aided by a human coach or critic, learners can articulate their problems in an infinite variety of ways. Computer-based support systems should be designed to conform to this metaphor.

On a broad scale, learning on demand is neither practical nor economical without computers. Learning on demand should include "learning to learn," providing the user with skills and showing the user how to locate and utilize information resources. It should not be restricted just to learning procedures but should help to restructure the user's conceptual model of the domain. It should not only provide access to factual information but also assist the user in understanding when that knowledge can be applied.

Learning on demand is a guided discovery approach to learning. It is initiated when the user wants to do something, not learn about everything. Learning on demand affords the following:

- It is easier to understand the uses for the knowledge being learned;
- Learning occurs because knowledge is actively used rather than passively perceived;
- At least one condition under which knowledge can be applied is learned;
- It can make a crucial difference in motivating learning.

Learning on demand can be differentiated according to whether the user or the system initiates the demand.

- *Demands Originating with the User.* The demand to learn more can originate with the user. It can be triggered by a discrepancy between an intended product and the actual product produced. Experimentation with a system may turn up interesting phenomena that users find worth exploring further. The user's mental model can serve as a driving force towards learning more. Users "feel" that there must be a better way of doing things. Adequate tools to support learning on demand are crucially important in making users willing to embark on an effort to increase their knowledge.
- *Suggestions from the Coach or the Critic.* The demand to learn cannot originate with users when they are unaware that additional functionality exists. The system has to take the initiative, but to avoid the problem that the system becomes too intrusive, a metric is necessary for judging the adequacy of a user's action. Interrupting too often can destroy motivation, but too few interruptions results in learning experiences being missed. Except for narrow problem domains (e.g., simple games [4]), optimal behavior cannot be uniquely defined. Therefore, the underlying metric should not be a fixed entity but a structure that users can inspect and modify, increasing the user's control over interaction with the system. Adequate communication structures must exist to make this a manageable task.

Tutoring episodes can play an important role in learning on demand. They can expose the user to certain tasks. The critic can offer to act as a tutor -- the crucial difference from the normal tutoring approach is that tutoring is initiated by the user and occurs in the context of the user's work.

Related Work

The critic paradigm is similar to the critiquing approach used in research efforts on medical systems [18, 19, 16, 22]. The critiquing approach uses domain knowledge to help physicians perform diagnoses or develop patient treatment plans. Techniques from expert systems research were modified after researchers recognized the need to assist physicians directly in their work, leaving them in control rather than attempting to replace them with an autonomous system. In contrast, our research and system development efforts have a human-computer interaction perspective. We ask how knowledge-based approaches can improve collaboration between a computer and a user.

REQUIREMENTS FOR CRITIC SYSTEMS

Design requirements for computer-based critics should be based on empirical studies. As we have studied human critics, it became obvious that knowledge is the most important feature of a good critic.

Empirical Studies

Cognitive scientists have studied human-to-human dyadic relationships. These studies emphasized psychological [5] and linguistic [15] aspects of dyadic human cooperative ef-

forts. Our own empirical work investigated why users work suboptimally, failing to take advantage of available system functionality. We observed the following problems:

1. Users do not know about the existence of tools and are not able to ask for them; passive help systems are of little use in such situations.
2. Users do not know how to access tools; retrievability is a big problem in information-rich societies and in complex, high-functionality systems.
3. Users do not know when to use these tools; they do not know the applicability conditions under which a piece of knowledge can be used successfully.
4. Users do not understand the results that tools produce; finding the information is in many cases not the end but the beginning of difficulties.
5. Users cannot combine, adapt, and modify a tool to their specific needs; reuse and redesign [8] have to be supported.

A consequence of these problems is that many systems are underused. We are strongly convinced that we need is not more information but new ways to structure and present it.

In other empirical studies we investigated how a model of the expertise of another user is acquired by a domain expert. This study was based on think-aloud protocols from experts [10]. A questionnaire showed that expertise is not consistent for a class of users. The results indicated that systems must model the individual's knowledge in terms of underlying domain concepts because simple classification approaches are inadequate.

The design of our critic systems has been influenced by these empirical studies. Our approach is based on two assumptions: that cooperative work is a powerful approach to both improving problem solving and learning, and that users need to be encouraged to explore.

Knowledge-Based Architectures

Knowledge-based systems are one promising approach to equipping machines with some human communication capabilities. Based on an analysis of human communication, we developed the model shown in Figure 1, and we have tried to instantiate this general architecture in a variety of systems.

The system architecture in Figure 1 contains two major improvements over traditional approaches:

- The **explicit** communication channel is widened (incorporating the use of windows, menus, pointing devices, etc.).
- Information can be exchanged over the **implicit** communication channel -- a prerequisite is shared knowledge structures.

There are four domains of knowledge shown in Figure 1:

1. *Knowledge about the problem domain:* Intelligent behavior builds upon in depth knowledge about specific domains. This knowledge constrains the possible actions and describes reasonable goals and operations. Most computer users are not interested in computers per se but want to use them to solve problems and accomplish tasks. To shape the computer into a truly usable and useful medium for them, we have to make it invisible and let them work *directly* on their problems and their tasks; we must support human problem-domain communication [11].
2. *Knowledge about communication processes:* Information structures that control communication should be made explicit.
3. *Knowledge about the communication partner: The* user of a system does not exist; there are many different kinds of users, and the requirements of an individual user change with experience. Systems will be unable to interact with users intelligently unless they have some means of finding out what the user really knows; they must be able to infer the state of the user's knowledge.
4. *Knowledge about the most common problems users have in using a system and about instructional strategies*: This knowledge is required if someone wants to be a good coach or teacher and not only an expert; a user support system should know when to interrupt a user. It must incorporate instructional strategies based on pedagogical theories, exploiting the knowledge contained in the system's model of the user.

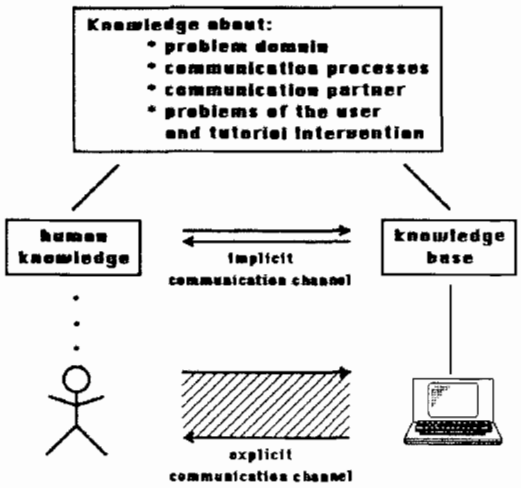


Figure 1: Architecture for Knowledge-Based Human-Computer Communication

Domain Knowledge

Expertise cannot exist without domain knowledge. The actual representation chosen for domain knowledge is not critical; rule-based systems, object hierarchies and frames are all appropriate. We have used rule-based systems because they support the incremental accumulation of domain knowledge. It remains to be seen how adequate our representation will be for some of the extensions we are currently pursuing.

Domain knowledge must be acquired; associated with that requirement are all the traditional issues of knowledge acquisition in knowledge-based systems. It may be that the critic methodology is an opportunity for using the content of previously developed knowledge bases, particularly those that are a part of expert systems that have not found acceptance as stand-alone systems.

Models of the User

To support incremental learning and learning on demand, systems should possess knowledge about a specific user, information about the user's conceptual understanding, the set of tasks for which the user uses the system, the user's way of accomplishing domain-specific tasks, pieces of advice given and whether they were remembered and accepted , and the situations in which the user asked for help.

In short each user must be treated as an individual. Computer systems based on a static model of users are often too rigid and limited to meet the demands of a diverse user community. There is no such thing as "the" user of a system: there are many different kinds of users and the requirements of an individual user change with experience. Robust and dynamic user models are a desirable design goal for computer-based critics.

Explanations

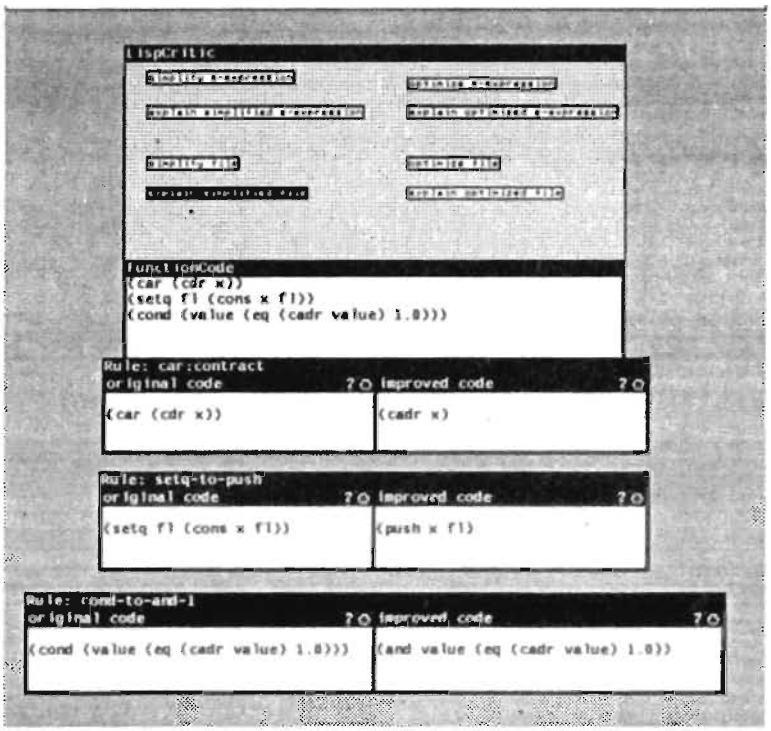
Explanation is critical for cooperative systems. It is a more difficult problem in critic systems than in tutoring systems because problems being addressed are arbitrary; that is the problem space is large, and the choice of which problem to solve is not controlled by the system.

Users learn best when they are situated in the context of their work and are able to receive explanations from an expert who can clear up misconceptions and clarify understanding. This helps the user to restructure his or her knowledge [21]. Learning is habitually supported with tutoring but a more likely situation, and one similar to that which evokes human-to-human interaction, is to provide for learning with a good explanation capability [27]. Good tutors (and critics) explain things by using concepts that a student already understands [26]. That explanations must be tailored to the user implies that the system must capture and represent the set of concepts each individual knows in a user model. The system then has to formulate (or select) explanations appropriate to the knowledge level and experience of each individual.

PROTOTYPICAL SYSTEMS

We have developed computer-based critics for several domains and have emphasized different issues, for example level of analysis, narrowly bounded versus open problem spaces and active versus passive approaches. We expect that by a careful analysis and detailed comparison of these system-building efforts, we will develop general principles for designing critics and other intelligent support system. In this section, we briefly describe two systems: LISP-CRITIC, a system that critiques LISP code and CRACK that assists kitchen designers (for a detailed descriptions see [7, 13]).

The LISP-CRITIC. The LISP-CRITIC, a passive critic for *FranzLisp* (see Figure 2), suggests improvements to program code. The critic works in one of two modes. Improvements can make the code either more *cognitively* efficient (i.e., more readable and concise) or more *machine* efficient (i.e., smaller and faster). Users can choose the kind of suggestions in which they are interested. LISP-CRITIC is more than a tutoring environment; it differs from LISP TUTOR [1] in that it augments the user's working environment by providing an available expert to assist him or her in producing a better program. In a session with LISP-CRITIC, as opposed to a structured tutoring episode, the user maintains control of both problem selection and the user-



This figure shows the LISP-CRITIC running on a bit graph terminal in a UNIX environment. The user can initiate an action by clicking a button. The FUNCTIONCODE pane displays the text of the program that LISP-CRITIC is working on. The other three windows show suggested transformations. The "?" in the title line of the windows is a button for obtaining an explanation.

Figure 2: The LISP-CRITIC

Replace a Copying Function with a Destructive Function

```
(rule append/.1-new.cons.cells-to-nconc/.1...     ;;; the name of the rule
   (?foo:(append append1)                         ;;; the original code
      (restrict ?expr                             ;;; condition
         (cons-cell-generating-expr expr))        ;;; (rule can only be applied
                                                  ;;; if "?expr" generates
                                                  ;;; cons cells)
      ?b)
   ==>
   ((compute-it:                                  ;;; the replacement
      (cdr (assq (get-binding foo)
                 '((append . nconc)
                   (append1 . nconc1)))))
    ?expr ?b)
   safe (machine))                                ;;; rule category
```

Example (see Figure 5):

```
(append (explode word) chars)
==>
(nconc (explode word) chars)
```

Figure 3: Example of a Rule in the LISP-CRITIC

computer interaction. In addition to improving the user's work, a by-product of this interaction is that the user learns more about LISP as a domain in the context of his or her work.

The system can be used by two different user groups. One group consists of intermediate users who want to learn how to produce better LISP code. We have tested the usefulness of LISP-CRITIC for this purpose by gathering statistical data on the programs written by students in an introductory LISP course. The other group consists of experienced users who want to have their code "straightened out." Instead of refining their code by hand (which in principle these users can do), they use LISP-CRITIC to help them carefully reconsider the code they have written. The system has proven especially useful with code that is under development, continuously being changed and modified.

LISP-CRITIC is able to criticize a user's code in the following ways:

- replace compound calls of LISP functions by simple calls to more powerful functions:
  `(not (evenp a))` may be replaced by
  `(oddp a);`
- suggest the use of macros:
  `(setq a (cons b a))` may be replaced by
  `(push b a);`
- find and eliminate 'dead' code:
  as in `(cond (...) (t ...) (dead code));`
- find alternative forms of conditional or arithmetic expressions that are simpler or faster;
- replace copying (garbage generating) function with a destructive function:
  `(append (explode word) chars)`
  may be replaced by
  `(nconc (explode word) chars);`
  see Figures 3 and5;
- specialized functions:
  replace `equal` by `eq` - use integer instead of floating point arithmetic wherever possible;
- evaluate or partially evaluate expressions:
  `(sum a 3 b 4)` may be simplified to
  `(sum a b 7).`

The Architecture of the LISP-CRITIC

The structure of the overall system is given in Figure 4. The user's code is simplified and analyzed according to the transformation rules, and protocol files are produced. They contain information (see Figure 2) that is used to generate explanations. The user model (for a more detailed discussion see [9]) obtains information from the rules that have fired, from the statistical analyzer, and from specialized knowledge acquisition rules which look for cues indicating that a specific concept of LISP is either known or not known by the user. In return, the user model determines which rules should fire and what explanations should be generated.

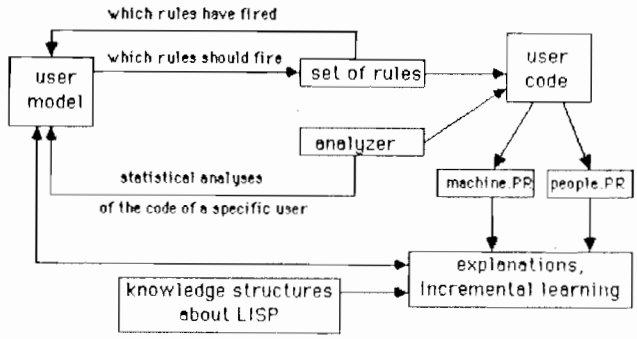
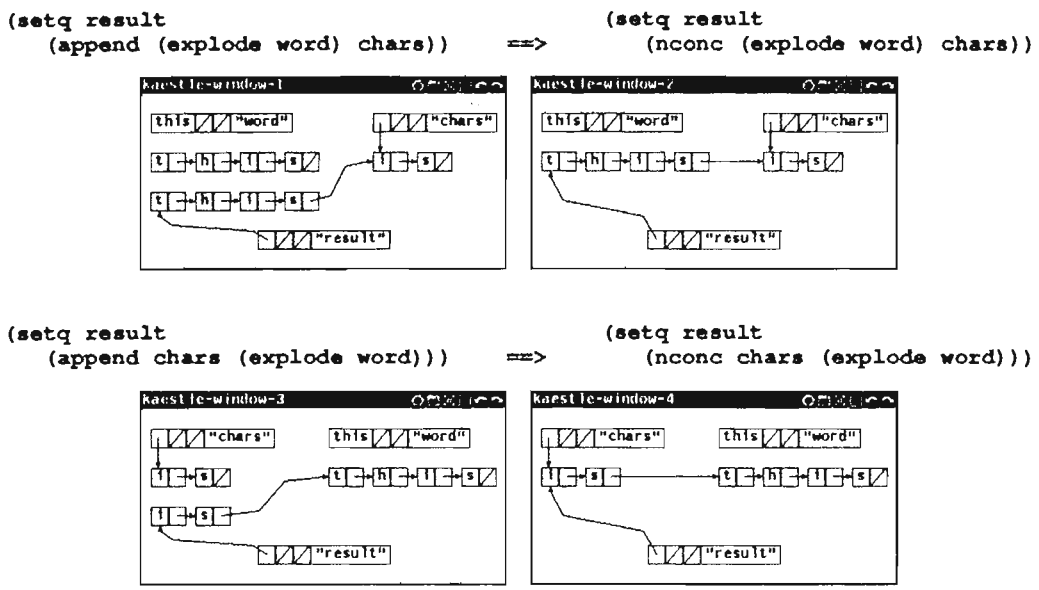
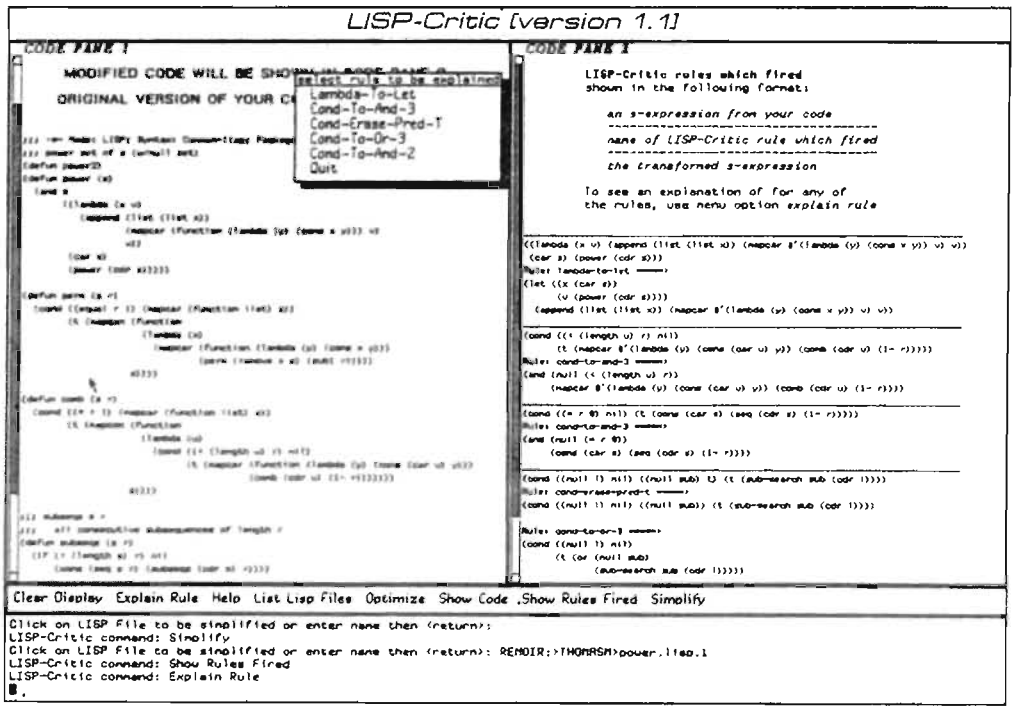


Figure 4: The Architecture of the LISP-CRITIC



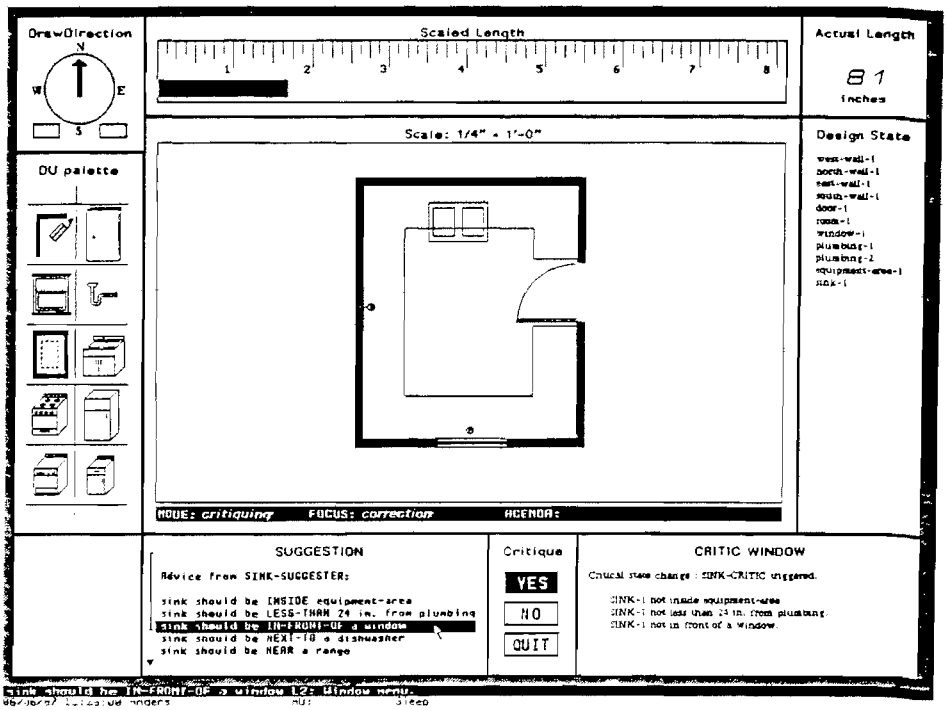
In the environment shown in the indlvidual screen images, the variable `word` is bound to the value `this` and the variable `chars` is bound to the list `(i s)`.

Figure 5: Illustration of the Validity of a Rule Using *Kaestle*



The interface shows the user working on a LISP-CRITIC code file. The user has seen the recommendations of LISP-CRITIC, has asked for a display of the rules which were applied and is about to request explanation of a particular rule.

Figure 6: The LISP-CRITIC Interface on the Symbolics Computer



CRACK's user interface is based on the metaphor of an "architect's workbench". Design units (DU's) are selected from the DU Palette, and their architectural symnbol moved within the work area (the center window). The user manipulates DU's by clicking on their name in the Design State window. The user can question suggestions and criticism by clicking on the text. Critiquing can be turned on and off.

Figure 7: Suggestions from the SINK-CRITIC

Support for Understanding the Criticism

Our experience with LISP-CRITIC in our LISP courses has been that the criticism given is often not understood. Therefore we use additional system components to illustrate and explain the LISP-CRITIC's advice. KAESTLE, a visualization tool that is part of our software oscilloscope [2], allows us to illustrate the functioning and validity of certain rules. In Figure 5, we use *Kaestle* to show why the transformation

```
(append (explode word) chars) ==>
(nconc (explode word) chars)
```

is safe (because `explode` is a cons-generating function; see Figure 3), whereas the transformation

```
(append chars (explode word)) ==>
(nconc chars (explode word))
```

is unsafe (because the destructive change of the value of the first argument by `nconc` may cause undesirable side effects.)

Present Research System Environment

LISP-CRITIC has been ported to other computing environments, most recently to the Symbolics 3600 (see Figure 6). Future research will use the Symbolics as a prototyping environment with COMMON LISP as the target domain. We emphasize issues in human computer interaction: usability, explanation, and user modelling.

CRACK

CRACK (see Figure 7) is a critic system that supports users designing kitchens. It provides a set of domain-specific building blocks and knows how to combine these building blocks into useful designs. It uses this knowledge "to look over the shoulder" of a user carrying out a specific design. If CRACK discovers a shortcoming in users' designs, it offers criticism, suggestions, and explanations. It assists users improve their designs through a cooperative problem solving process. CRACK is not an expert system; it does not dominate the design process by generating new designs from high-level goals or resolving design conflicts automatically. The users controls the behavior of the system at all times (e.g., the critiquing can be turned on and off), and if users disagree with CRACK, they can modify its knowledge base.

CRACK aids users in designing the layout of a kitchen floor plan while seated at a graphics workstation (see Figure 7). The system is actually a collection of critics, each of which is an expert on a specific design unit (DU). These critics serve a dual purpose: they monitor what the user is doing and interject their critique when appropriate, and they can provide a suggestion if asked. Users can also ask for an explanation of either a criticism or a suggestion. These explanations are "hard-wired" into the system.

Most of the knowledge contained in the critics was obtained from protocol studies, a questionnaire, and traditional kitchen design books. We found that the system needed a method for overriding these sources of knowledge when user preferences conflicted with them. CRACK allows users to modify a critic in order to better fit it to their preferences.

COMPONENTS OF OUR CRITIC SYSTEMS

Domain Knowledge

We represent domain knowledge in rule based formats. In the case of LISP-CRITIC, these rules are expressed in LISP in a format developed for this application. CRACK uses the ART expert system shell environment and its underlying rule based architecture for knowledge representation. Example rules for the LISP-CRITIC system are shown in Figure 3.

Model of the User

As discussed previously, computer-based critics must contain a user model in order to reach their full potential. Our work with CRACK indicated that it is possible to develop a usable system without an underlying user modelling component. Also, LISP-CRITIC in its initial form did not attempt to create individual user models and appeared to function at a satisfactory level. However, for these systems to be truly integrated into an individual's personal working environment, they must adjust to the knowledge level and preferences of the individual user.

Representing the User Model. Our first attempts in LISP-CRITIC to model the user were classification approaches. We categorized an individual by his or her expertise, inferred by observation of programming habits. This approach turned out to be inadequate and caused us to reflect on expertise in the domain of LISP. Knowledge needs to be represented in the user model as a collection of concepts that each individual knows. It cannot be assumed that a whole class of users know the same set of concepts just because they have the same background or experience. A survey of experienced LISP programmers in our department confirmed this intuition. Our test of expertise was the programmer's understanding of generalized variables in COMMON LISP [24] and preference for using and teaching the "setq" and "setf" special forms. We discovered a significant variability not only in preference but also in their understanding of the concept. These experiences have led us to represent each user as a collection of concepts that he or she knows or doesn't know about LISP along with an associated confidence factor.

Acquisition of the User Model. The problem of knowledge acquisition for the user model in LISP-CRITIC will be solved primarily by examining code written by the user. Techniques described in [7] have been developed to extend the system beyond recognizing pieces of code that can be improved to recognizing the use of both constructs and concepts that LISP-CRITIC thinks are preferable. A module statistically analyzes the code for average function length and depth of nesting. This analysis gives a measure of readability and allows the system to infer a crude approximation of the user's expertise.

Explicit acquisition of user knowledge has not been attempted for the LISP-CRITIC itself; however, we experimented with this approach when we attempted to build an initial model of the user for a tutoring system for a personal workstation environment. This approach appeared to work well in a limited domain, but it is severely limited in its ability to acquire an accurate initial model of the user's knowledge of a domain as complex as LISP.

Implicit acquisition of user knowledge will have to be supported in order to make our system robust. Our approach to implicit knowledge acquisition involves a hierarchy of levels:

1. CUES - low-level primitives evidenced by the use of particular syntax or constructs;
2. CHUNKS - the representation of LISP concepts in the user model;
3. STEREOTYPES - groups of the chunks used for inferring additional data in the user model.

The primary source for cues is LISP-CRITIC rules that fire when a pattern is found in the user's code. Collections of rules that have fired imply that the programmer knows a particular concept (possesses a chunk), and furthermore that the system believes this with a certain level of confidence. Similarly, collections of chunks trigger a stereotype [23]. Chunks in that stereotype in addition to the set that triggered the stereotype can now be indirectly inferred and added to the user model.

Explanations

Critics must be able to explain their actions in terms of knowledge about the underlying domain. Our first approach to these explanations was to select appropriate textual explanations from prestored information -- canned text. This approach was not entirely satisfactory because advice was often not understood and textual descriptions alone made the concepts hard to visualize.

We believe that human's efficient visual processing capabilities must be utilized fully. Traditional displays have been one-dimensional, with a single frame on the screen filled with lines of text. New technologies offer ways to exploit human visual perception with multiple window displays, color, graphics, and icons. Figure 5 shows one of our visualization tools that illustrates the rationale for a complicated rule in the LISP-CRITIC.

EVALUATION

Research on intelligent support systems must move beyond "arm-chair design". These systems are so complex that building them is not good enough. We have to test our implementations in real-world domains, those in which people actually use the computer as a medium for their work.

Evaluation Techniques

We have tested our critics systems with real users over extended periods of time. Various evaluation methods (e.g., think-aloud protocols [17] and questionnaires) showed that a strictly quantitative evaluation is not feasible because many important factors are only qualitative.

Results of Evaluation

The results of our evaluations of LISP-CRITIC showed its strengths and weaknesses.

Some of the strengths of LISP-CRITIC are:

- It supports users in doing their own tasks and it supports intermediate users, not just beginners;
- It enhances incremental learning;
- It fosters reusability by pointing out operations that exist in the system;
- It can be applied to every program (in the worst case nothing is found to critique;)
- It is not just a toy system because users have used it in the context of their everyday work;
- Using it does not require users to provide information in addition to the code.

Some of the weaknesses of LISP-CRITIC are:

- It use only low-level transformations (i.e., it operates primarily at the level of s-expressions;)

- It has absolutely no understanding of the user's problem; this limits analysis because LISP-CRITIC cannot distinguish between constructs the user does not know and those not required to solve this problem.
- The rules are not tied to higher-level concepts;
- The explanations should be generated more dynamically [20].

In our evaluation of CRACK, which has been an operational system almost a year, we accumulated feedback about its strengths and shortcomings. One of our colleagues who is not a professional kitchen designer, remodeled his kitchen. He considered CRACK a valuable tool. The criticism generated by the system during his design process illustrated several design concepts of which he was not aware. In addition to generating a specific design for his kitchen, our colleague increased his knowledge about kitchen design.

The system was also used by a design methodologist who considered the cooperative, user-dominated approach of CRACK its most important feature. He felt that this set CRACK apart from expert system oriented design tools that users have little control of and that often reduce users to spectators of the system's operations. We have deliberately avoided equipping the current version of CRACK with its own design capabilities. Too much assistance and too many automatic procedures can reduce the users' motivation by not providing sufficient challenge. In contrast to most current CAD systems, which are merely drafting tools rather than design tools, CRACK has some "understanding" of the design space. This knowledge allows the system to critique a design during the design process -- a capability absent in CAD systems.

Our evaluations also confirmed that the critic paradigm, although attractive and useful in many situations, does have limitations. It is not an expert system capable of generating, on its own, a complete and correct solution to every problem. Nor is it a better tutoring approach but merely one that is appropriate under certain circumstances. A totally naive user should still be exposed to initial instruction in a domain to prevent floundering and frustration. We do feel, however, that the critiquing approach uses techniques that approximate human-to-human cooperation in day-to-day work settings.

FUTURE RESEARCH AND CONCLUSIONS

The deficiencies we uncovered in our evaluation work are the basis for our future research agenda.

Structured Representation of Domain Knowledge

The results of our initial efforts indicated the need for representing domain knowledge in a form which can be used in the critiquing process itself, for explaining criticism, and for representing the user's knowledge state. Rules alone are inadequate. We are investigating the decomposition of LISP as a domain into concepts, called "chunks" in our user model. The user model is a collection of chunks which the system inferred a user does or does not know along with an associated degree of confidence in that inference. Rules will continue to be the applicative form of our LISP knowledge in the critiquing process. They will be catalogued and organized by our taxonomy of concepts, and used to guide explanation.

Beyond Canned Explanations

Explanations of LISP-CRITIC's "behavior" have been canned text pegged to the user's knowledge level (novice, intermediate, or expert). We are investigating approaches for generating explanations on the fly using the domain knowledge structure and the user model, thereby integrating the "explainable experts systems" approach [20].

Differential Descriptions

Another approach which depends heavily on the user model and on maintaining a record of context for the user's work is the use of differential explanations. Descriptions of concepts new to a particular user will be generated by relating them to concepts already known; the latter are contained in the user model.

Cooperative Problem Solving Systems

The long-term goal of this effort is to develop the full potential of the critic paradigm and to make it a prototype for designing cooperative problem solving systems. We would like to endow our critic systems with various techniques of deliberation that would allow users to choose a critic approach that fits their style of working and learning.

Conclusions

Computer-based critics incorporate many powerful ideas from human-computer communications and artificial intelligence into a system that makes use of the best aspects of human and computational cognition. They have the potential to provide a symbiotic relationship between a user and a knowledge-based system. This environment can support cooperative work between these two agents while helping the user learn in the context of his or her own work.

Implementation of this concept will require that computer-based critics contain domain knowledge represented in a form that is applicable both to problem solving and to explanations. An explanation component will use that knowledge base and an inferred user model to generate contextual explanations. The system will share its knowledge with the user while building up a dynamic user model.

We have developed several critic systems that incorporate some of these ideas and have formulated a plan to extend at least one of these system, the LISP-CRITIC. The successes and failures of this research will help us define the characteristics and design considerations for critic systems as well as gauge their potential. These results should be applicable to the entire class of cooperative problem solving systems.

Acknowledgment

Many people have contributed to the development of LISP-CRITIC over the last few years. The authors would like to thank especially Heinz-Dieter Boecker, who developed many of the original ideas; Andreas Lemke, who contributed to the general framework; Helga Nieper-Lemke, who developed KAESTLE; and Anders Morch who is the author of CRACK. We also thank John Reiman, Paul Johl, and Patrick Lynn for recent work on LISP-CRITIC user modelling and explanations components, Hal Eden for porting the system to the Symbolics environment. The research is partially supported by a grant from the Colorado Institute of Artificial Intelligence, *The CIAI is sponsored in part by the Colorado Advanced Technology Institute (CATI), an agency of the State of Colorado. CATI promotes advanced technology education and research at universities in Colorado for the purpose of economic development.*

References

1. J.R. Anderson, B.J. Reiser. "The LISP Tutor". *BYTE 10*, 4 (April 1985), 159-175.

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

3. B.G. Buchanan, E.H. Shortliffe. Human Engineering of Medical Expert Systems. In *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, Addison-Wesley Publishing Company, Reading, MA, 1984, Chap. 32, pp. 599-612.

4. R.R. Burton, J.S. Brown. An Investigation of Computer Coaching for Informal Learning Activities. In *Intelligent Tutoring Systems*, D.H. Sleeman, J.S. Brown, Eds., Academic Press, London - New York, 1982, ch. 4, pp. 79-98.

5. D.F. Dansereau. Cooperative Learning Strategies. In *Learning and Study Strategies: Issues in Assessment, Instruction and Evaluation*, Academic Press, New York, 1988, Chap. 7, pp. 103-120.

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

7. G. Fischer. A Critic for LISP. Proceedings of the 10th International Joint Conference on Artificial Intelligence (Milan, Italy), Los Altos, CA, August, 1987, pp. 177-184.

8. G. Fischer. "Cognitive View of Reuse and Redesign". *IEEE Software, Special Issue on Reusability 4*, 4 (July 1987), 60-72.

9. G. Fischer. Enhancing Incremental Learning Processes with Knowledge-Based Systems. In *Learning Issues for Intelligent Tutoring Systems*, Springer-Verlag, New York, 1988, Chap. 7, pp. 138-163.

10. G. Fischer, P. Johl, T. Mastaglio, J. Rieman. A Study of Expert Inferences of Novice Programmer Knowledge from Their Programs. in preparation, Department of Computer Science, University of Colorado, 1988.

11. G. Fischer, A.C. Lemke. "Construction Kits and Design Environments: Steps Toward Human Problem-Domain Communication". *Human-Computer Interaction 3*, 3 (1988), 179-222.

12. G. Fischer, A.C. Lemke, T. Schwab. Knowledge-Based Help Systems. Human Factors in Computing Systems, CHI'8! Conference Proceedings (San Francisco, CA), ACM, New York, April, 1985, pp. 161-167.

13. G. Fischer, A. Morch. CRACK: A Critiquing Approach to Cooperative Kitchen Design. Proceedings of the International Conference on Intelligent Tutoring Systems (Montreal, Canada), June, 1988, pp. 176-185.

14. G. Fischer, M. Schneider. Knowledge-Based Communication Processes in Software Engineering. Proceedings of the 7th International Conference on Software Engineering (Orlando, FL), IEEE Computer Society, Los Angeles, CA, March, 1984, pp. 358-368.

15. B. Fox, L. Karen. Collaborative Cognition. Proceedings of the Tenth Annual Conference of the Cognitive Science Society, Cognitive Science Society, 1988.

16. C. Langlotz, E. Shortliffe. "Adapting a Consultation System to Critique User Plans". *International Journal of Man-Machine Studies 19* (1983), 479-496.

17. C.H. Lewis. Using the 'Thinking-Aloud' Method in Cognitive Interface Design. RC 9265, IBM, Yorktown Heights, NY 1982.

18. P. Miller. *A Critiquing Approach to Expert Computer Advice: ATTENDING.* Pittman, London - Boston, 1984.

19. P. Miller. *Expert Critiquing Systems: Practice-Based Medical Consultation by Computer.* Springer-Verlag, New York - Berlin, 1986.

20. R. Neches, W.R. Swartout, J.D. Moore. "Enhanced Maintenance and Explanation of Expert Systems Through Explicit Models of Their Development". *IEEE Transactions on Software Engineering SE-11*, 11 (November 1985), 1337-1351.

21. J. Psotka, L.D. Massey, S. Mutter. Intelligent Instructional Design. In *Intelligent Tutoring Systems: Lessons Learned*, Lawrence Erlbaum Associates, Hillsdale,NJ , 1988, pp. 113-118.

22. P.L. Reichertz, D.A.B. Lindberg (Ed.). *A Computational Model of Reasoning from the Clinical Literature.* Springer-Verlag, New York, 1987.

23. E. Rich. "Users are Individuals: Individualizing User Models". *International Journal of Man-Machine Studies 18* (1983), 199-214.

24. G.L. Steele. *Common LISP: The Language.* Digital Press, Burlington, MA, 1984.

25. M.J. Stefik. "The Next Knowledge Medium". *AI Magazine 7*, 1 (Spring 1986), 34-46.

26. K. VanLehn. Student Modeling. In M. Polson, J. Richardson, Ed., *Foundations of Intelligent Tutoring Systems*, Lawrence Erlbaum Associates, Hillsdale,NJ , 1988, pp. 55-78.

27. E. Wenger. *Artificial Intelligence and Tutoring Systems.* Morgan Kaufmann Publishers, Los Altos, CA, 1987.

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