University of Colorado at Boulder

Center for LifeLong Learning and Design (L3D)

Department of Computer Science

ECOT 717 Engineering Center
Campus Box 430
Boulder, Colorado 80309-0430
(303) 492-1592, FAX: (303) 492-2844

Domain-Oriented Design Environments as Models for the Design of Collaborative Systems

Gerhard Fischer

Center for LifeLong Learning and Design (L3D)
University of Colorado
Campus Box 430
Boulder, CO 80309-0430
gerhard@cs.colorado.edu

Proceedings of the International Workshop on the Design of Cooperative Systems, Antibes-Juan-les-Pines France, January 1995, pp. 20-41

DOMAIN-ORIENTED DESIGN ENVIRONMENTS AS MODELS FOR THE DESIGN OF COLLABORATIVE SYSTEMS

Gerhard Fischer
Center for Lifelong Learning and Design (L^3D)
Department of Computer Science and Institute of Cognitive Science,
University of Colorado, Campus Box 430
Boulder, Colorado 80309
phone: 303-492-1592 fax: 303-492-2844
e-mail: gerhard@cs.colorado.edu
http://www.cs.colorado.edu/homes/gerhard/public_html/Home.html

Abstract. *Domain-oriented design environments* (DODEs) support collaboration between (1) all stakeholders in a design process and (2) between stakeholders and computational environments. They serve as model for the design of collaborative systems by exploring and supporting different relationships and task responsibilities between humans and computers. Within human-computer collaboration, they are grounded within the complementary approach rather than the emulation or replacement approach by exploiting the strengths and the weaknesses of human and computational agents.

By representing models of specific domains, DODEs provide a shared context and ground design with representations supporting mutual education and understanding by all stakeholders.

This paper focuses on the following major themes for collaboration: (1) collaboration between environment developers and domain designers, (2) collaboration between domain designers and clients, (3) indirect, long-term collaboration among designers in the development of a domain-oriented design environment itself as well as individual artifacts within a DODE, and (4) computational support for all of these collaborative efforts.

Resume. Les environnements de conception orientes domaine (DODES en anglais) fournissent un moyen pour faciliter la collaboration entre (1) les participants dans le processus de conception et (2) entre les participants et l'environnement informatique. Ils servent de modele pour la conception de systemes collaboratifs en permettant l'exploration et l'aide dans les responsabilites sur les relations et les taches entre humains et ordinateurs. Dans une collaboration homme-machine, ces environnements reposent sur une approche exploitant la complementarite plutot qu'une approche basee sur l'emulation ou la substitution, ces dernieres approches exploitant plutot les forces et faiblesses des agents humain et machine.

En representant les modeles de domaines specifiques, les environnements decrits dans ce papier fournissent un contexte partage et les bases d'une conception avec des representations qui favorisent une education et une comprehension mutuelles des participants.

Ce papier est centre sur les themes majeurs de toute collaboration: (1) la collaboration entre les developpeurs de l'environnement et les concepteurs du domaine; (2) la collaboration entre les concepteurs du domaine et les clients; (3) une collaboration indirecte et a long terme parmi les concepteurs dans le developpement d'un environnement de conception oriente domaine; et (4) une aide computationnelle pour tous ces efforts collaboratifs.

Keywords. symmetry of ignorance among stakeholders; seeding/evolutionary growth/reseeding model;

representation for mutual education and understanding; indirect, long-term collaboration; distributed cognition

1. Introduction

In our work, we have created collaborative systems named *domain- oriented design environments (DODEs)* in support of design. *Design* in this context refers to the broad endeavor of creating artifacts (as exercised by architects, industrial designers, curriculum developers, composers, etc., and as defined and characterized, for example, by [Simon 81; Schoen 83; Ehn 88]), rather than to a specific step in a software engineering life-cycle model.

A growing number of research efforts are using knowledge-based systems and new communication paradigms *to empower all stakeholders in design, not to replace them* (stakeholders in a design process are all people who are affected by the design artifact and who are involved in creating and evolving the design artifact). The idea of human augmentation, beginning with Engelbart [Engelbart, English 68], has been elaborated in the last 25 years (e.g., [Stefik 86; Hill 89; Fischer 90; Norman 93]). This paper will (1) introduce dimensions for the design of collaborative systems, (2) describe DODEs, (3) characterize DODEs as models of collaborative systems, and (4) assess them from different perspectives.

2. Dimensions for the Design of Collaborative Systems

Collaborative Problem Solving. Collaborative problem-solving approaches [Fischer 90] in which computational environments empower, augment, and complement human skills and knowledge are a more desirable and promising goal to pursue than is automatic programming. Collaborative problem-solving systems raise questions such as (a) which part of the responsibility can or should be exercised by the human and which part by the computer, and (b) how do the human and the intelligent system effectively communicate? Collaborative problem-solving approaches do not deny the power of automation [Billings 91], but they focus our concerns on the ''right kind of automation'' including interaction mechanisms designed for humans rather than for programs.

Symmetry of Ignorance. Nothing can be worse than designers who think everyone else is just like them. In the early days of computing, almost all systems were developed and used by computer professionals. Introspection by software designers served as a reasonable source of knowledge at that time, but it has lost its power today for the development of systems in application domains. Today, designing complex systems is an activity involving *many* stakeholders, making collaboration a necessity [Greenbaum, Kyng 91; Fischer et al. 92]. By emphasizing design as a collaborative activity, DODEs support (1) collaboration between environment developers and domain designers, (2) collaboration between domain designers and clients, (3) indirect, long-term collaboration among designers in the development of a DODE itself as well as individual artifacts within a DODE, and (4) computational support for all of these collaborative efforts.

Domain-Orientation. In a conventional, domain-independent software environment, designers who produce new software artifacts typically have to start with general programming constructs and methodologies. This forces them to focus on the raw materials necessary to implement a solution rather than to try to understand the problem. Design environments need to support *human problem-*

domain communication [Fischer, Lemke 88] by providing computational environments that model the basic abstractions of a domain (as pursued in efforts in domain modeling). They give designers the feeling that they interact with a domain rather than with low-level computer abstractions. Domain-orientation allows humans to take both the content and context of a problem into account, whereas the strength of formal representations is their independence of specific domains to make domain-independent reasoning methods applicable [Norman 93].

Modern application needs are not satisfied by traditional programming languages, which evolved in response to system programming needs. More emphasis should be put on the creation of computational environments that fit the needs of professionals of other disciplines outside the computer science community. The chief risks of using ideas from programming language design and formal specification techniques are in succumbing to the temptations of excess generality and in assuming that users and domain experts think like software designers. The semantics of DODEs are tuned to specific domains of discourse. This involves support for different kinds of primitive entities, for specification of properties other than computational functionality, and for computational models that match the users' own models.

Evolution. There is growing agreement (and empirical data to support it) that the most critical software problem is the cost of maintenance and evolution [CSTB 90]. Studies of software costs indicate that about two-thirds of the costs of a large system occur after the system is delivered. Much of this cost is due to the fact that a considerable amount of essential information (such as design rationale [Fischer et al. 91a]) is lost during development and must be reconstructed by the designers who maintain and evolve the system.

In order to make maintenance and enhancements "first class" activities in the lifetime of an artifact, (1) the reality of change needs to be accepted explicitly and (2) increased up-front costs have to be acknowledged and dealt with. We learned the first point in our work on end-user modifiability [Girgensohn 92], which demonstrated that there is no way to modify a system without detailed programming knowledge unless modifiability was an explicit goal in the original design of the system. The second point results from the fact that "design for redesign" requires efforts beyond designing for what is desired and known at the moment. It requires that changes be anticipated and structures be created that will support these changes.

The evolution of a software system is driven by breakdowns [Fischer, Nakakoji 92] experienced by the users of a system. In order to support evolutionary processes, domain designers need to be able, willing, and rewarded to change systems, thereby providing a potential solution to the maintenance and enhancement problems in software design. Users of a system are knowledgeable in the application domain and know best which enhancements are needed. An end-user modification component supports users in adding enhancements to the system without the help of the system developers. End-user modifiable systems will take away from system developers some of the burden of anticipating all potential uses at the original design time.

Representations for Mutual Education and Mutual Understanding. To address the symmetry of ignorance problem, DODEs provide representations for mutual education and mutual understanding [Ehn 88] that help increase the *shared context* [Resnick 91] necessary for collaboration. Environments

supporting collaboration must be (1) familiar to *all* participants, (2) use the practice of the users as a starting point, (3) allow the envisioning of work situations supported by the new systems, and (4) enhance incremental mutual learning and shared understanding among the participants.

Communication between stakeholders is difficult because they use different languages. Explicit representations [Fischer, Nakakoji, Ostwald 93] ground collaborative design by providing a context for communication. These representations help to detect communication breakdowns caused by unfamiliar terminology and tacit background assumptions, and turn breakdowns into opportunities for creating a shared understanding [Fischer, Nakakoji 92].

An important component of shared understanding is the *intent* of the collaborators. Understanding intent enhances mutual intelligibility by serving as a resource for assessing the relevance of information within the context of collaboration. In everyday communication between people, intent is often implicitly communicated against a rich background of shared experience and circumstances. Machines, however, have a limited notion of background, and this limits their ability to infer the intent of users [Suchman 87].

3. Domain-Oriented Design Environments

Domain-oriented design environments (DODEs) [Fischer 94] have emerged in our research work as computational environments in support of collaboration. They are semi-formal systems, integrating object-oriented hierarchies of domain objects, rule based critiquing systems, case-based catalog components, simulation components, checklists, and argumentative hypermedia systems. They are representational media supporting communications and negotiations between all involved stakeholders and between the designers and their work in progress. They do limited reasoning and interpretations, trigger breakdowns, deliver information, and support the exploration of the rationale behind the artifact.

The *goals* associated with DODEs are: (1) bring task to the forefront by supporting human problem-domain interaction, (2) create a shared context between designers and computational environments, (3) create an artifact-centered information repository facilitating collaboration between all stakeholders, (4) support learning on demand and information delivery, and (5) have human designers in control. The *theories* underlying DODEs are: (1) make objects and ideas ready-to-hand [Stahl 93], (2) support reflection-in-action [Schoen 83], (3) integrate problem framing and problem solving [Rittel 84], (4) allow design-in-use [Henderson, Kyng 91], (5) increase the back-talk of the situations [Fischer et al. 91b], and (6) make argumentation serve design [Fischer et al. 91a]. The *users* of DODEs are skilled domain workers belonging to the community of practice which a specific DODE supports.

A Process Model for DODEs. Our process model for continual development of design environments from an initial seed through iterations of growth and reseeding is illustrated in Figure 1:

- The *seeding* process, in which domain designers and environment developers work together to instantiate a domain-oriented design environment seeded with domain knowledge.
- The *evolutionary growth* process, in which domain designers add information to the seed as they use it to create design artifacts.

- The *reseeding* process, in which environment developers help domain designers to reorganize and reformulate information so it can be reused to support future design tasks.

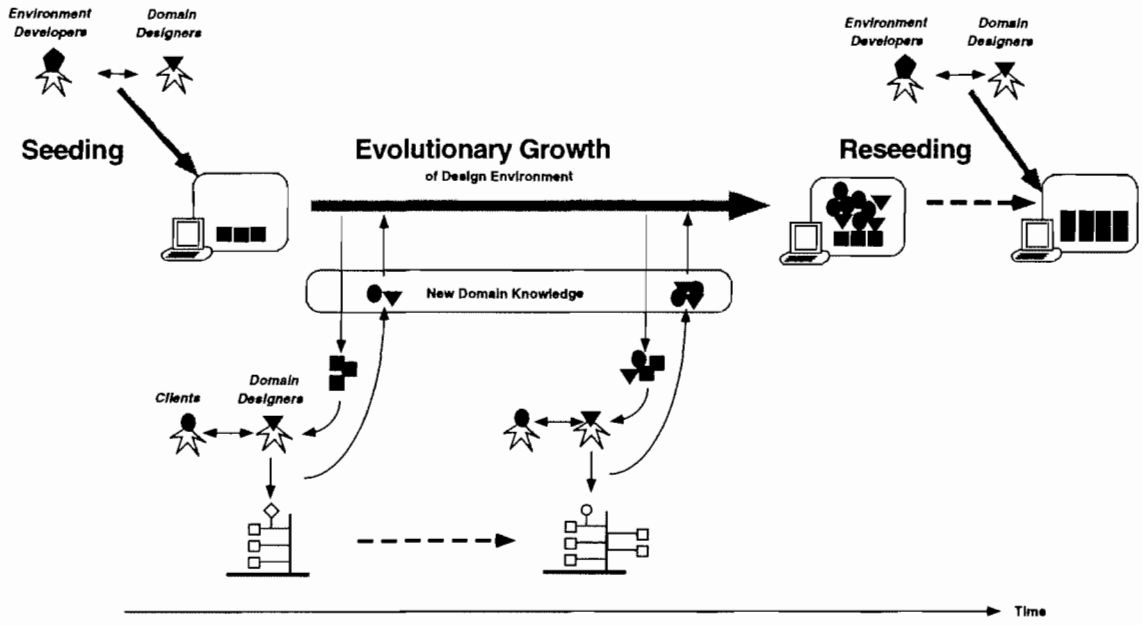

Figure 1: Seeding, Evolutionary Growth, and Reseeding: A Process Model for Domain-Oriented Design Environments

During seeding, environment developers and domain designers collaborate to create a design environment seed. During evolutionary growth, domain designers create artifacts that add new domain knowledge to the seed. In the reseeding phase, environment developers again collaborate with domain designers to organize, formalize, and generalize new knowledge.

The top half shows how DODEs are created through a collaboration between the environment developers and domain designers. The bottom half shows the use of a DODE in the creation of an individual artifact through a collaboration between domain designers and clients. The use of a DODE contributes to its evolution; i.e., new knowledge, developed in the context of an individual project, is incorporated into the evolving design environment.

The Seeding Process. A seed is built by customizing the domain- independent design environment architecture to a particular domain through a process of knowledge construction. Although the goal is to construct as much knowledge as possible during seed-building, for complex and changing domains complete coverage is not possible. Therefore, the seed is explicitly designed to capture design knowledge during use [Girgensohn 92].

Domain designers must participate in the seeding process because they have the expertise to determine when a seed can support their work practice. Rather than expecting designers to articulate precise and complete system requirements prior to seed building, we view seed building as knowledge *construction* (in which knowledge structures and access methods are collaboratively designed and built) rather than as knowledge *acquisition* (in which knowledge is transferred from an expert to a knowledge engineer and finally expressed in formal rules and procedures). New seed requirements are elicited by constructing and evaluating domain-oriented knowledge structures.

Evolutionary Growth Through Use. During the use phase, each design task has the potential to add to the knowledge contained in the system. New construction kit parts and rules are required to support design in rapidly changing domains. Issue-based information in the seed can also be augmented by each design task as alternative approaches to problems are discovered and recorded. The information accumulated in the information space during this phase is mostly informal because designers either cannot formalize new knowledge or they do not want to be distracted from their design task.

Reseeding. Acquiring design knowledge is of little benefit unless it can be delivered to designers when it is relevant. Periodically, the growing information space must be structured, generalized, and formalized in a reseeding process, which increases the computational support the system is able to provide to designers [Shipman 93].

The task of reseeding involves environment developers working with domain designers. After a period of use, the information space can be a jumble of annotations, partial designs, and discussions mixed in with the original seed and any modifications performed by the domain designers. To make this information useful, the environment developers work with the domain designers in a process of organizing, generalizing, and formalizing the new information and updating the initial seed.

An Example. NETWORK (see Figure 2), a DODE for computer network design [Fischer et al. 92; Reeves 93; Shipman 93] will be used as an example to illustrate our approach. The seeding process for NETWORK was driven by observations of network design sessions, prototypes of proposed system functionality, and discussions centered on the prototypes. In design sessions, a logical map of the network being designed served to ground design meetings, discussions, what-if scenarios, and disagreements. The logical map was chosen as the central representation of the artifact in network design, and a prototype construction kit was implemented based on the logical map [Fischer et al. 92]. Evaluation of the NETWORK seed indicated that designers need support for communication in the form of critiques, reminders, and general comments [Reeves, Shipman 92]. Pointer, annotation, and sketching tools were integrated into the environment so talking about the artifact could take place *within* the artifact (see Figure 6).

An important lesson we learned during the seeding of NETWORK was to base our design discussions and prototyping efforts on existing artifacts. Discussing the existing computer science network at CU Boulder was an effective way to elicit domain knowledge because it provided a concrete context that triggered domain designers' knowledge. We found high-level discussions of general domain concepts to be much less effective than discussions focused on existing domain artifacts.

Information to seed NETWORK was acquired from existing databases containing information about network devices, users, and the architectural layout of our building. The NETWORK seed contains formal representations of approximately 300 network devices and 60 users. Autocad™ databases created by facilities maintenance personnel provide architectural details of about 100 rooms. This information is represented in NETWORK's construction kit and in the underlying knowledge representation formalisms. The informal part of the NETWORK seed includes notes from the systems administration class, knowledge about the various research groups, and electronic mail of the network designers.

Our work with NETWORK showed the shortcomings of electronic mail in the context of a collaborative

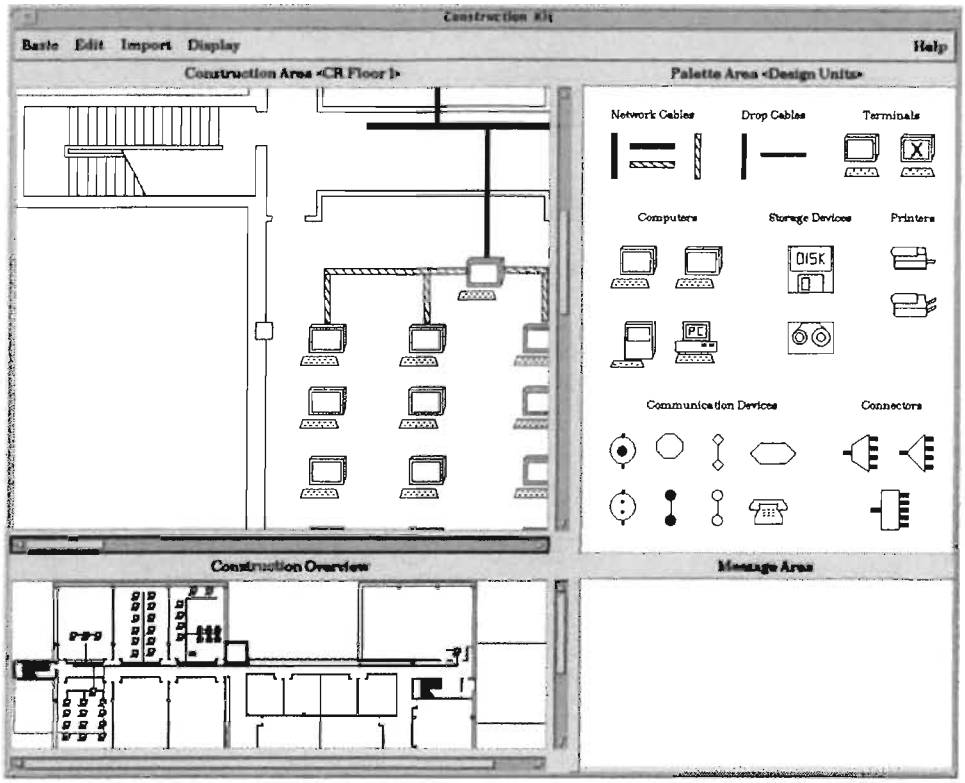

Figure 2: An Environment Supporting Computer Network Design

A screen image of the NETWORK seed. Shown are a palette of network objects (upper right), the construction area where logical networks are configured (upper left), an overview pane to provide designers with a global picture (lower left), and a pane for critiquing messages (lower right).

design effort. By communicating asynchronously via electronic mail using textual annotations, network designers separated the notes and the artifact in ways that made interpretation and understanding difficult. The necessity for integration was observed in two ways. First, in design sessions videotaped for analysis, *deictic references* (referring to items by the use of "these," "those," "here," etc.) were frequent. A long-term study of network designers showed that users took advantage of embedded annotations and made frequent use of deictic references [Reeves 93]. Second, discussion about the artifact guided the incremental design process. Designers took every opportunity to illustrate critiques. Only rarely was a detailed comment made and not accompanied by changing the artifact.

The logical map mentioned above served not only to represent the real network, but also as a medium through which changes were considered and argued (Figure 3). It focused as well as facilitated discussion. Frequently, in arguing over design artifacts, specific issues led to discussions of larger issues. Collaborating designers preferred to ground discussions in design representations. The logical maps served to (1) point out inconsistencies between an appealing idea and its difficulty of implementation, (2) remind participants of important constraints, and (3) describe network states before and after changes.

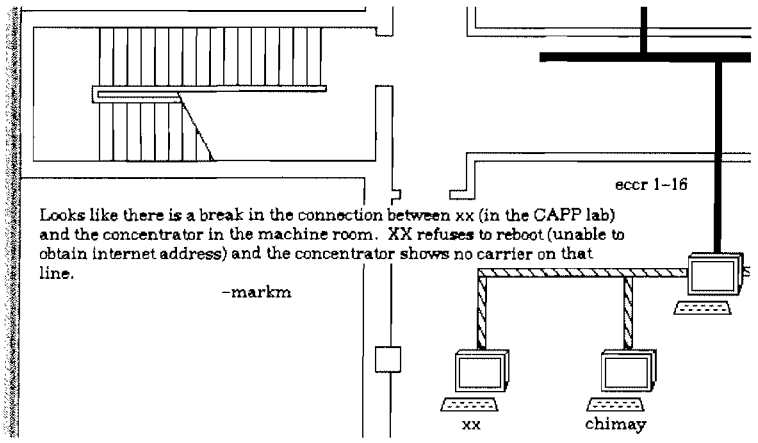

Figure 3: Logical Map with Embedded Discussion

The logical map serves to abstract away low-level details while allowing discussions about the artifact to be embedded in the design.

4. DODEs as Models of Collaborative Systems

Seeding: Mutual Education and Mutual Understanding between Environment Developers and Domain Designers. The predominant activity in designing complex systems is the participants teaching and instructing each other [Curtis, Krasner, Iscoe 88; Greenbaum, Kyng 91]. As argued with the existence of the "symmetry of ignorance", complex problems require more knowledge than any single person possesses, making communication and collaboration among all the involved stakeholders a necessity. Domain designers understand the practice and environment developers know the technology. None of these carriers of knowledge can guarantee that their knowledge is superior or more complete compared to other people's knowledge. The goal of the seeding process is to activate as much knowledge from as many stakeholders as possible taking into account that system requirements are not so much analytically specified as they are collaboratively evolved through an iterative process of consultation between all stakeholders [CSTB 90]. This iterative process of consultation requires representations (such as prototypes, mock-ups, sketches, scenarios, or use situations that can be experienced) which are intelligible and can serve as "objects-to-think-with" for all involved stakeholders.

This iterative process is important to support the interrelationship between problem framing and problem solving documented by many design methodologists (e.g., [Rittel 84; Schoen 83]). They argue convincingly that (1) one cannot gather information meaningfully unless one has understood the problem, but one cannot understand the problem without information about it; and (2) professional practice has at least as much to do with defining a problem as with solving a problem. New requirements emerge during development because they cannot be identified until portions of the system have been designed or implemented. The conceptual structures underlying complex software systems are too complicated to be specified accurately in advance, and too complex to be built faultlessly. Specification and implementation have to co-evolve, requiring the owners of the problems to be present in the development.

Evolutionary Growth through Design-in-Use. Software systems model parts of our world. Our world evolves in numerous dimensions as new artifacts appear (e.g., new gadgets in computer network design), new knowledge is discovered, and new ways of doing business are developed. Successful software systems need to evolve. Maintaining and enhancing systems need to become "first class design activities," extending system functionality in response to the needs of its users. There are numerous fundamental reasons why systems cannot be done "right." Designers are people, and people's imagination and knowledge are limited. The evolution is driven by using an existing generic design environment for the development of a new specific artifact. Figure 4 illustrates the intertwining between a designer learning from the existing environment, and at the same time contributing her/his knowledge and ideas to the environment.

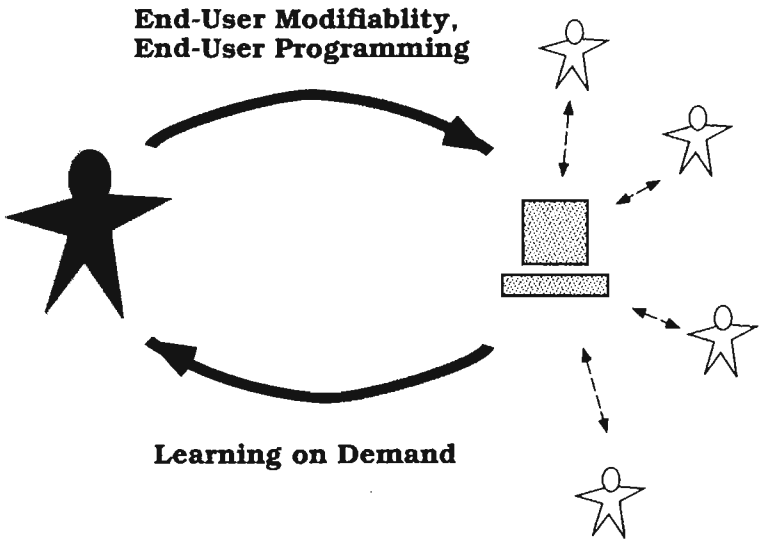

Figure 4: The Role of Learning on Demand and End-User Modifiability in the Evolution of Collaborative Systems

DODEs provide learning on demand opportunities for a designer through critiquing, simulation and access to contextualized argumentation and cases. But the information flow is not only one-directional. Using DODEs, designers will transcend the existing knowledge and contribute new knowledge themselves. Because these designers are domain designers and not software designers, end-user modifiability support is required.

Reseeding: Formalizing and Reorganizing. Acquiring design knowledge is of little benefit unless it can be delivered to designers when it is relevant. Periodically, the growing information space must be structured, generalized, and formalized in a reseeding process, which increases the computational support the system is able to provide to designers [Shipman 93].

The task of reseeding involves environment developers working with domain designers. After a period of use, the information space can be a jumble of annotations, partial designs, and discussions mixed in with the original seed and any modifications performed by the domain designers. To make this information useful, the environment developers work with the domain designers in a process of organizing, generalizing, and formalizing the new information and updating the initial seed.

Indirect, Long-Term Collaboration. Realistic design projects are increasingly large, complex, and

long in duration. The design process can extend over many years, only to be followed by extended periods of maintenance and redesign. Specialists from many different domains must coordinate their efforts despite large separations of distance and time. In such projects, constructive collaboration is crucial for success yet difficult to achieve. This difficulty is due in large part to ignorance by individual designers of how the decisions they make interact with decisions made by other designers. A large part of this, in turn, consists of simply not knowing what has been decided and why.

Meetings and other types of direct communication are the commonly used means for coordination and collaboration in design projects, but in many situations — especially ones involving long-term collaboration — these are not feasible. Design projects that extend over many years can involve a high turnover in personnel. Much of the design work on systems is done as enhancements and redesign, and the people doing this work are often not members of the original design team. But to be able to do this work well or at all requires ''collaboration'' with the original designers of the system. In general, people who are not in the project group at the same time need to coordinate and collaborate in long-term collaborative design.

In long-term collaborative design tasks, communication between designers is often *indirect* in the sense that the senders and receivers of information are not known a priori. Instead, the time and place in which communication happens are both unpredictable (see Figure 5).

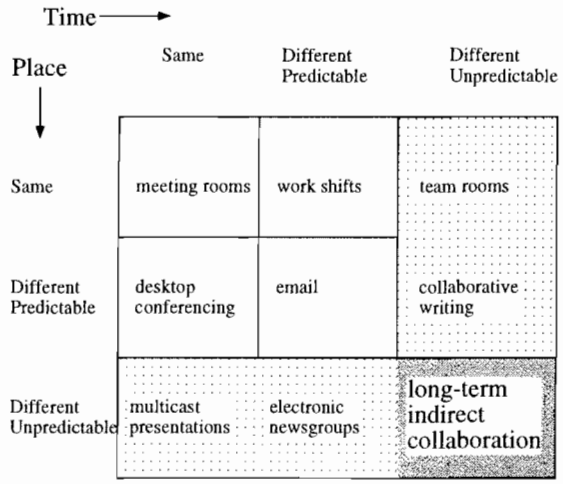

Figure 5: A Classification of Different CSCW Perspectives

This classification scheme extends the usual CSCW 2x2 matrix of same/different times and places. By focusing on indirect, long-term collaboration, we are specifically exploring group technologies that support and represent the intentions and actions of others who cannot be seen and contacted personally. Because communication and collaboration are indirect, knowledge must be represented *within* the system and activities are shared and grounded around evolving artifacts.

Long-term, indirect communication is of particular importance in situations where (1) direct communication is impossible, impractical or undesirable, (2) communication is shared around artifacts and information space evolution, (3) designed artifacts continue to evolve over long periods of time (e.g, over months or years), and (4) designers need to be informed within the context of their work on real-

world design problems.

Support for indirect coordination and collaboration must go beyond what electronic mail and most proposed CSCW software could provide, even in principle. This support should allow team members to work separately — across substantial distances in space and time — but alert them to the existence of potential interactions between their work and the work of others. Where such interactions exist, support should be provided for collaboration and conflict resolution. Designers must be able to *interact with design artifacts created by previous designers*. Technology enabling this could effectively create virtual cooperation between all designers who ever worked on the project.

Expectation Agents. Expectation agents [Girgensohn, Redmiles, Shipman 94] provide support for collaboration between developers and users in DODE. At the level of the development of the DODE itself, the collaboration takes place between the environment developers and the domain designers, and (2) at the level of the development of an individual artifact within a DODE, the collaboration takes place between the domain designers and the clients (see Figure 1). Expectation agents monitor users working with prototype systems and report mistaches between developers' expectations and a system's actual usage. At the same time, the agents provide users with an opportunity to communicate with developers (either synchronously or asynchronously). The agent-based interaction complements traditional participatory design in which users participate in the development process with face-to-face meetings. The agent-based approach enables the usability of a prototype to be observed unobtrusively while users perform their normal tasks. The evolution of a system from this perspective occurs as a feedback mechanism by responding to discrepancies between a system's actual and desired states. Crucial to an understanding of the need for evolving prototypes is that the ''desired state'' cannot be clearly articulated. Irrespective of training or task analyses, many design concepts remain tacit. The presence of an actual prototype forces design decisions and their full implications to become explicit.

Expectation agents can help in observing ''actual use'' in contrast to ''expected use''. Such agents know about a family of tasks the system is intended to support and the sequence of user interactions the developers envisioned for performing these tasks. In the case of a mismatch, agent may perform the following actions: (1) notify developers of the discrepancy, (2) provide users with an explanation based on developers' rationale, and (3) solicit a response to or comment about the expectation. What action is appropriate depends on the type of discrepancy and whether the situation allows for immediate communication between users and developers. In sum, this type of agent initiates a dialog between users and developers in which the developers communicate their intent to the users and users have the opportunity to respond. Additional follow-up discussions can take place outside of the agent-based software development environment.

The notion of agents introduced with this work is that of a mechanism to facilitate communication. The communication enables a mutual understanding to evolve between developers and end users. The emphasis is on accurately matching a system to the needs of its intended users and less on enforcement of requirements that, in actuality, may be inappropriate for a problem situation. In this role, agents provide several advantages.

5. Assessment of DODEs

Empirical Foundations Through Assessment Studies in Naturalistic Settings. The times of purely prescriptive design methodologies in software design belong to the past. ''Arm-chair'' design and supply-side computing are not sufficient to solve real-world problems [Thomas, Kellogg 89]. Software is created in the real world, deals with real tasks, and involves human beings with different interests, skills, and knowledge. To make future computing systems succeed requires more than concern for technology — *it requires concern for human beings, their tasks, and their organizations.* A specific challenge for DODEs which support communities of practice is to create seeds which offer sufficient functionality that domain designers can use the systems in their day-today work without doing additional work or sacrificing their productivity.

We have conducted numerous user studies to assess the value and importance of the domain-orientation [Fischer 94], the critiquing component [Fischer et al. 91b], the specification component [Nakakoji 93], the end-user modifiability component [Girgensohn 92], the annotation component [Reeves 93], the formalization component [Shipman 93], and the proactivity approach [Sullivan 94]. These evaluation have demonstrated that all of these components are supportive of collaboration in the design of complex artifacts. Specific studies often uncovered surprising results, e.g., [Bonnardel, Sumner 94] found that the critiquing component had little influence on the design product; but, by just knowing the critiquing system was active stimulated designers to deeply analyze why they were breaking design guidelines and to formally record their design rationale in the system's knowledge base.

Exploring Different Collaboration Paradigms. Human-computer collaboration can be conceptualized and operationalized from two different perspectives [Terveen 95]:

- the *human emulation* approach where the basic assumption is to endow computers with human-like abilities (such as natural language, speech, etc.), and
- the *human complementary* approach which exploits the asymmetric abilities of humans and computers and tries to identify the most desirable and most adequate role distributions between humans and computational environments [Billings 91].

Our work is grounded in the complementary approach. We have explored (depending on the task, the interest and the people involved) different styles of collaboration. In most of our work related to critiquing, humans design and computers critique [Fischer et al. 91b]. But critiquing in specific situations (i.e., where computers know the task, e.g., the size of a file directory tree or an inheritance structure in object-oriented design) can be done by humans, after the computer displayed a first approximation of the information structure. Another paradigm is proactivity [Sullivan 94] which allows designers to delegate certain tasks to the system - and in performing these tasks the system uses knowledge that may be unknown, and yet be of interest to the designer. In this context, work-centered events are the triggers of learning episodes for designers; the system provides them with contextualized information in the process of solving a problem today, which might be relevant for future problem solutions.

DODEs and Artificial Intelligence. DODEs put an emphasis on humans rather instead on automation. Rather than ''getting the human out of the loop,'' DODEs empower designers and users to create and evolve artifacts fitting their needs and desires. The major difference between classical expert systems and DODEs is that the human is much more an active agent and participant in the latter ones. Traditional expert systems asked the user many questions and then returned an answer. In a collaborative

problem solving system the user and the system share the problem solving and decision making and requiring much richer communication facilities than the ones which were offered by expert systems. It raises two important questions: (1) what part of the responsibility still has to be exercised by human beings? and (2) how do we organize things so that the intelligent part of the automatic system can communicate effectively with the human part of the intelligent system?

Collaborative problem solving systems can deal better than expert systems with the following issues (for details see [Fischer 90]):

1. partial understanding and knowledge of complex task domains is less detrimental,
2. two agents can achieve more than one — especially by exploiting the asymmetry between agents,
3. breakdowns are not as detrimental, especially if the system provides resources for dealing with the unexpected,
4. background assumptions do not need to be fully articulated beforehand, but can be incrementally articulated,
5. semi-formal system architectures are appropriate, and
6. humans can enjoy ''doing'' and ''deciding'' by being involved in the process.

Increasing the Shared Understanding. Design is a well-suited activity to explore concepts in collaboration because the design activity takes place within the computational environment. The ''situation awareness'' of a DODE is increased through the following mechanisms [Nakakoji 93] : (1) the domain orientation allows a default intent to be assumed, namely, the creation of a artifact in the given domain; (2) the construction situation is accessible and can be ''parsed'' by the system, providing the system with information about the artifact under construction; and (3) the specification component allows one to explicitly communicate high-level design intentions to the system. The embedding of annotations (see Figure 3) increases the situation awareness of a DODE even further [Reeves 93]. Figure 6 illustrates the implication of embedding communication within the environment.

Who is the Beneficiary and Who has to Do the Work. Research in software design in the past has operated as an overly prescriptive discipline, often postulating a ''new human'' [Simon 81] with interests (e.g., detailed knowledge of low- level computer operation), knowledge (e.g., about work procedures of an application domain), and motivations (e.g., to provide extensive amounts of design rationale, or to deal with formal methods). This idealized human had little correspondence with reality. Most issue of collaboration transcend narrow technical issues and succeed or fail based on social and organizational issues. For a designer enhancing an existing artifact, the existence of a rich design rationale and of mechanisms for end-user modifiability are highly desirable. But the original developers of the system need to do substantially more work to make this all available (e.g., in the case of expectation agents, end-users must care enough about the flaws in the system and the developers must care enough about the needs and complaints of the end-users). The necessity to invest in long-term benefits must be taken seriously. Unless there are organizational recognition and awards for these activities, designers will not be motivated to create these information structures, because they have to do the work without being the beneficiaries [Grudin 89].

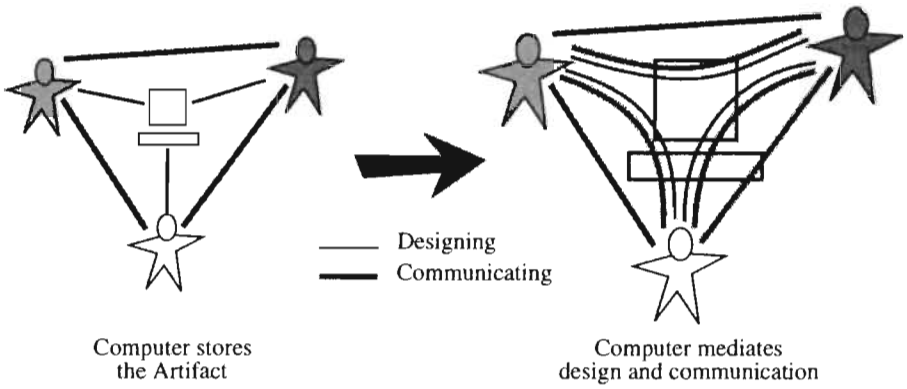

Figure 6: Embedding Artifact and Communication in Computational Environments

The left part of the figure shows communication (e.g., via e-mail) being outside of the "situation awareness" of the system. By talking about the artifact within the artifact (e.g., using annotation, see Figure 3), the shared understanding can be increased and the system can act more collaboratively by delivering information in a more contextualized way.

Current Limitations and Research Issues for DODEs. The appeal of the DODE approach lies in its compatibility

- with an emerging methodology for design [Cross 84; Ehn 88; Schoen 83; Simon 81],
- with views of the future as articulated by practicing software engineers [CSTB 90],
- with findings of empirical studies [Curtis, Krasner, Iscoe 88], and
- with the *integration* of many recent efforts to tackle specific issues in collaboration (e.g., representation of context and intent [Fischer, Nakakoji, Ostwald 93], recording design rationale [Fischer et al. 91a], supporting case-based reasoning, creating artifact memories [Terveen, Selfridge, Long 93], and so forth).

We are further encouraged by the excitement and widespread interest of DODEs and the numerous prototypes being constructed, used and evaluated in the last few years [Fischer 94].

DODEs raise numerous research issues. Creating seeds for a variety of different domains will require substantial resources and the willingness of people from different disciplines to collaborate. By being high- functionality systems, DODEs create a tool mastery burden. Our experience has shown that the costs of learning a programming language are modest compared to those of learning a full-fledged design environment. New tools (e.g., critics [Fischer et al. 91b], expectation agents [Girgensohn, Redmiles, Shipman 94], etc.), and support mechanisms for learning on demand [Fischer 91]) are needed to address these problems.

There are numerous other reasons that a DODE approach may not be readily accepted. Software designers often have difficulties with the idea that they do not create "universal solutions" that make everyone happy. They have difficulties in sacrificing generality for increased domain-specific support. DODEs replace the clean and controllable waterfall model with a much more interactive situation in

which the search for ''correct'' solutions is limited to downstream activities. DODEs will lead to further specialization of computer users into environment developers who create (in collaboration with domain designers) the seeds for design environments, and of domain designers who solve problems by exploiting the resources of the design environments

6. Conclusions

Ideas from many disciplines have contributed and are integrated in DODEs. Design research provided concepts such as reflection-in-action, symmetry of ignorance and the need for integrating problem framing and problem solving. Human computer interaction research contributed interaction paradigms, information presentation techniques, and a general framework for reflecting on the shared responsibilities and role distributions of humans and computers in collaborative systems. Artificial Intelligence provided ideas and techniques for knowledge representation and formal computational modeling. Social science research contributed ideas and background knowledge about work practices, design, role of artifacts in human activity, organizational learning, group memories, and motivation.

While being grounded in some aspects of the traditions of these disciplines, DODEs also transcend some of the existing frameworks. They are in sharp contrast to the isolation assumption of Artificial Intelligence which assumes that a human agent formulates the problem in a previously defined language understandable to the computational agent and that the problem statement will include background knowledge, a description of the the state of some world, operators to use in that world and a description of a desired state [Bobrow 91]. They transcend the design perspectives by increasing the back-talk of artifacts with computational agents [Fischer, Nakakoji 92]. They extend the CSCW perspective by embedding many computational mechanisms (such as critiquing, simulations, shared intent, artifact centered communication) thereby providing additional support for humans. They transcend a disembodied intelligence approach [Norman 93] by allowing cognition and knowledge be distributed among humans and computers and bringing all resources together through collaboration.

Acknowledgments

The author would like to thank the members of the Center for Lifelong Learning and Design at the University of Colorado who have made major contributions to the conceptual framework and systems described in this paper. The research was supported by (1) the National Science Foundation, Grant RED-9253425, (2) the ARPA HCI program, Grant N66001-94-C-6038, (3) Nynex, Science and Technology Center, (4) Software Research Associates (SRA), and (5) PFU. During the academic year 1994/95, the author is supported by the ''SEL-Stiftungsprofessur'' of the Technical University Darmstadt.

References

[Billings 91]
C.E. Billings, *Human-Centered Aircraft Automation: A Concept and Guidelines*, NASA Technical Memorandum 103885, NASA Ames Research Center, Moffett Field, CA, August 1991.

[Bobrow 91]
D.G. Bobrow, *Dimensions of Interaction*, AI Magazine, Vol. 12, No. 3, Fall 1991, pp. 64-80.

[Bonnardel, Sumner 94]
N. Bonnardel, T. Sumner, *From System Development to System Assessment: Exploratory STudy of the Activity of Professional Designers*, Proceedings of the 7th European Conference on Cognitive Ergonmics (Bonn, Germany), September 1994, pp. 23-36.

[Cross 84]
N. Cross, *Developments in Design Methodology*, John Wiley & Sons, New York, 1984.

[CSTB 90]
Computer Science and Technology Board, *Scaling Up: A Research Agenda for Software Engineering*, Communications of the ACM, Vol. 33, No. 3, March 1990, pp. 281-293.

[Curtis, Krasner, Iscoe 88]
B. Curtis, H. Krasner, N. Iscoe, *A Field Study of the Software Design Process for Large Systems*, Communications of the ACM, Vol. 31, No. 11, November 1988, pp. 1268-1287.

[Ehn 88]
P. Ehn, *Work-Oriented Design of Computer Artifacts*, Almquist & Wiksell International, Stockholm, Sweden, 1988.

[Engelbart, English 68]
D.C. Engelbart, W.K. English, *A Research Center for Augmenting Human Intellect*, Proceedings of the AFIPS Fall Joint Computer Conference, The Thompson Book Company, Washington, D.C., 1968, pp. 395-410.

[Fischer 90]
G. Fischer, *Communications Requirements for Cooperative Problem Solving Systems*, The International Journal of Information Systems (Special Issue on Knowledge Engineering), Vol. 15, No. 1, 1990, pp. 21-36.

[Fischer 91]
G. Fischer, *Supporting Learning on Demand with Design Environments*, Proceedings of the International Conference on the Learning Sciences 1991 (Evanston, IL), Lawrence Birnbaum (ed.), Association for the Advancement of Computing in Education, Charlottesville, VA, August 1991, pp. 165-172.

[Fischer 94]
G. Fischer, *Domain-Oriented Design Environments*, Automated Software Engineering, Vol. 1, 1994, pp. 177-203.

[Fischer et al. 91a]
G. Fischer, A.C. Lemke, R. McCall, A. Morch, *Making Argumentation Serve Design*, Human Computer Interaction, Vol. 6, No. 3-4, 1991, pp. 393-419.

[Fischer et al. 91b]
G. Fischer, A.C. Lemke, T. Mastaglio, A. Morch, *The Role of Critiquing in Cooperative Problem Solving*, ACM Transactions on Information Systems, Vol. 9, No. 2, 1991, pp. 123-151.

[Fischer et al. 92]
G. Fischer, J. Grudin, A.C. Lemke, R. McCall, J. Ostwald, B.N. Reeves, F. Shipman, *Supporting Indirect, Collaborative Design with Integrated Knowledge-Based Design Environments*, Human Computer Interaction, Special Issue on Computer Supported Cooperative Work, Vol. 7, No. 3, 1992, pp. 281-314.

[Fischer, Lemke 88]
G. Fischer, A.C. Lemke, *Construction Kits and Design Environments: Steps Toward Human Problem-Domain Communication*, Human-Computer Interaction, Vol. 3, No. 3, 1988, pp. 179-222.

[Fischer, Nakakoji 92]
G. Fischer, K. Nakakoji, *Beyond the Macho Approach of Artificial Intelligence: Empower Human Designers - Do Not Replace Them*, Knowledge-Based Systems Journal, Vol. 5, No. 1, 1992, pp. 15-30.

[Fischer, Nakakoji, Ostwald 93]
G. Fischer, K. Nakakoji, J. Ostwald, *Facilitating Collaborative Design through Representations of Context and Intent*, Proceedings of AAAI-93 Workshop, AI in Collaborative Design (Washington DC), 1993, pp. 293-312.

[Girgensohn 92]
A. Girgensohn, *End-User Modifiability in Knowledge-Based Design Environments*, Ph.D. Dissertation, Department of Computer Science, University of Colorado, Boulder, CO, 1992, Also available as TechReport CU-CS-595-92.

[Girgensohn, Redmiles, Shipman 94]
A. Girgensohn, D. Redmiles, F. Shipman, *Agent-Based Support for Communication between Developers and Users in Software Design*, Proceedings of the 9th Annual Knowledge-Based Software Engineering (KBSE-94) Conference (Monterey, CA), IEEE Computer Society Press, Los Alamitos, CA, September 1994, pp. 22-29.

[Greenbaum, Kyng 91]
J. Greenbaum, M. Kyng (eds.), *Design at Work: Cooperative Design of Computer Systems*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1991.

[Grudin 89]
J. Grudin, *Why groupware applications fail: Problems in design and evaluation*, Office Technology and People, Vol. 4, No. 3, 1989, pp. 245-264.

[Henderson, Kyng 91]
A. Henderson, M. Kyng, *There's No Place Like Home: Continuing Design in Use*, in J. Greenbaum, M. Kyng (eds.), *Design at Work: Cooperative Design of Computer Systems*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1991, pp. 219-240, ch. 11.

[Hill 89]
W.C. Hill, *The Mind at AI: Horseless Carriage to Clock*, AI Magazine, Vol. 10, No. 2, Summer 1989, pp. 29-41.

[Nakakoji 93]
K. Nakakoji, *Increasing Shared Understanding of a Design Task Between Designers and Design Environments: The Role of a Specification Component*, Unpublished Ph.D. Dissertation, Department of Computer Science, University of Colorado, 1993, Also available as TechReport CU-CS-651-93.

[Norman 93]
D.A. Norman, *Things That Make Us Smart*, Addison-Wesley Publishing Company, Reading, MA, 1993.

[Reeves 93]
B.N. Reeves, *Supporting Collaborative Design by Embedding Communication and History in Design Artifacts*, Ph.D. Dissertation CU-CS-694-93, Department of Computer Science, University of Colorado, Boulder, CO, 1993.

[Reeves, Shipman 92]
B.N. Reeves, F. Shipman, *Supporting Communication between Designers with Artifact-Centered Evolving Information Spaces*, Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'92), ACM, New York, November 1992, pp. 394-401.

[Resnick 91]
L.B. Resnick, *Shared Cognition: Thinking as Social Practice*, in L.B. Resnick, J.M. Levine, S.D. Teasley (eds.), *Perspectives on Socially Shared Cognition*, American Psychological Association, Washington, D.C., 1991, pp. 1-20, ch. 1.

[Rittel 84]
H.W.J. Rittel, *Second-Generation Design Methods*, in N. Cross (ed.), *Developments in Design Methodology*, John Wiley & Sons, New York, 1984, pp. 317-327.

[Schoen 83]
D.A. Schoen, *The Reflective Practitioner: How Professionals Think in Action*, Basic Books, New York, 1983.

[Shipman 93]
F. Shipman, *Supporting Knowledge-Base Evolution with Incremental Formalization*, Ph.D. Dissertation, Department of Computer Science, University of Colorado, Boulder, CO, 1993, Also available as TechReport CU-CS-658-93.

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

[Stahl 93]
G. Stahl, *Interpretation in Design: The Problem of Tacit and Explicit Understanding in Computer Support of Cooperative Design*, Ph.D. Dissertation, Department of Computer Science, University of Colorado, Boulder, CO, 1993.

[Stefik 86]
M.J. Stefik, *The Next Knowledge Medium*, AI Magazine, Vol. 7, No. 1, Spring 1986, pp. 34-46.

[Suchman 87]
L.A. Suchman, *Plans and Situated Actions*, Cambridge University Press, Cambridge, UK, 1987.

[Sullivan 94]
J. Sullivan, *A Proactive Computational Approach for Learning While Working*, Unpublished Ph.D. Dissertation, Department of Computer Science, University of Colorado, 1994.

[Terveen 95]
L.G. Terveen, *An Overview of Human-Computer Collaboration*, Knowledge-Based Systems Journal, Vol. 8, No. 2-3, April-June 1995, pp. 67-81.

[Terveen, Selfridge, Long 93]
L.G. Terveen, P.G. Selfridge, M.D. Long, *From Folklore to Living Design Memory*, Human Factors in Computing Systems, INTERCHI'93 Conference Proceedings, ACM, April 1993, pp. 15-22.

[Thomas, Kellogg 89]
J.C. Thomas, W.A. Kellogg, *Minimizing Ecological Gaps in Interface Design*, IEEE Software, Vol. 6, January 1989, pp. 78-86.