Domain-Oriented Design Environments:

Knowledge-Based Systems for the Real World

Gerhard Fischer
Center for LifeLong Learning and Design (L^3D)
Department of Computer Science and Institute of Cognitive Science
Campus Box 430, University of Colorado
Boulder, CO 80309-0430 USA
Phone: (303) 492-1502 or (303) 492-1592 Fax: (303) 492-2844
E-mail: gerhard@cs.colorado.edu
WWW: http://www.cs.colorado.edu/~gerhard/

Abstract. Over the last ten years we have created a theoretical framework for domain-oriented design environments (DODEs), developed several prototypes and assessed them in real-world settings. DODEs are knowledge-based systems that emphasize a human-centered and domain-oriented approach. Used as intelligent support systems, they facilitate collaboration and communication among humans as well as between humans and their computational environments to create more useful and usable artifacts.

In this paper, we discuss a component architecture (the multifaceted architecture) and a process model (the seeding, evolutionary growth, reseeding model) underlying DODEs by focusing specifically on their support for evolution. We describe two of the applications developed for voice dialog and computer network design, and we discuss our experience with the DODE approach derived from real-world uses in collaboration with companies and communities of practice.

Keywords: domain-oriented design, domain-oriented design environments, end-user modification, evolution, organizational learning and organizational memory

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 REC-9553371, (2) the ARPA HCI program, Grant N66001-94-C-6038, (3) NYNEX, Science and Technology Center, and (4) U S WEST Advanced Technologies.

Table of Contents

Why Real-World Knowledge-Based Systems Have to Evolve

We live in a world characterized by evolution—i.e., by ongoing processes of development, formation, or growth in both natural [Dawkins 1987] and human-created systems [Simon 1981]. Biology tells us that complex, natural systems are not created all at once but instead must evolve over time. We are becoming increasingly aware that evolutionary processes are ubiquitous and critical for complex software systems, such as real-world knowledge-based systems, because these systems do not necessarily exist in a technological context alone but instead are embedded within dynamic human organizations.

Our research efforts over the last decade have conceptualized the design of complex software systems as an evolutionary process in which system requirements and functionality are determined through an iterative process of collaboration among multiple stakeholders [CSTB 1990; Greenbaum, Kyng 1991]. Our theoretical work builds upon on theories of knowledge [Polanyi 1966; Popper 1965], design and design processes [Rittel 1984; Simon 1981], and empirical findings providing support for the theoretical orientation [Buchanan, Shortliffe 1984; Curtis, Krasner, Iscoe 1988]. Our theories are instantiated and assessed through the initial development and evolution of a domain-oriented design environment (DODE).

In this paper we present DODEs, give a brief discussion of the theory behind them, describe the multi-faceted architecture and the process model underlying them, and discuss our experience of using them in real-world environments.

Domain-Oriented Design Environments

DODEs are software systems that support design activities within a *particular* domain. They are examples of complex software systems that need to evolve. Design within a particular domain typically involves several stakeholders whose knowledge can be elicited only within the context of a particular design problem. Different stakeholders include the developers of a DODE (environment developers), the users of a DODE (domain designers), and the people for whom the design is being created (clients). A major assumption behind our work is that to effectively support design activities, DODEs that address authentic design need to increase communication between the different stakeholders and anticipate and encourage evolution at the hands of domain designers. DODEs integrate the capture of design rationale, end-user modifiability, and increased communication between end users and system designers so that system change can occur through an evolutionary process.

Understanding the Problem Is the Problem. The predominant activity in designing complex systems is the participants teaching and instructing each other [Curtis, Krasner, Iscoe 1988; Greenbaum, Kyng 1991]. Because complex problems require more knowledge than any single person possesses, communication and collaboration among all the involved stakeholders are necessary. Domain experts understand the practice and system designers know the technology. To overcome this "symmetry of ignorance" [Rittel 1984] (i.e., none of these carriers of knowledge can guarantee that their knowledge is superior or more complete compared to other people's knowledge), as much knowledge from as many stakeholders as possible should be activated with the goal of achieving mutual education and shared understanding.

Integrating Problem Framing and Problem Solving. Design methodologists (e.g., [Rittel 1984; Schoen 1983] demonstrate with their work the strong interrelationship between problem framing and problem solving. 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.

Communication and Coordination. Because designing complex systems is an activity involving many stakeholders, communication and coordination are of crucial importance. The types of communication and coordination processes that can be differentiated are those between (1) designers and users/clients, (2) members of design teams, and (3) designers and their computational knowledge-based design environment. By emphasizing design as a collaborative activity, domain-oriented design environments support three types of collaboration: (1) collaboration between domain-oriented designers (e.g., professional kitchen designers) and clients (owners of the kitchen to be built), (2) collaboration between domain-oriented designers and design environment builders (software designers), and (3) long-term indirect collaboration among designers (creating a virtual collaboration between past, present, and future designers). Design environments provide representations that serve as "languages of doing" [Ehn 1988] and therefore help increase the "shared context" [Resnick, Levine, Teasley 1991] necessary for collaboration.

The Need for Change. Knowledge-based systems model parts of our world. Our world evolves in numerous dimensions as new artifacts appear, 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.

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 1990]. 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. 1991b]) 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 [Fischer, Girgensohn 1990], 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 1994b] 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 [Henderson, Kyng 1991].

Reinventing the Wheel. The design of knowledge-based systems is a new design discipline relative to other more established disciplines. I claim that software designers can learn a lot by studying other design disciplines such as architectural design, engineering design, organizational design, musical composition, and writing. For example, the limitations and failures of design approaches that rely on directionality, causality, and a strict separation between analysis and synthesis have been recognized in architecture for a long time [Rittel 1984]. A careful analysis of these failures could have saved knowledge engineers the effort expended in finding out that waterfall-type models can at best be an impoverished and oversimplified model of real design activities. Assessing the successes and failures of other design disciplines does not mean that they have to be taken literally (because software artifacts are different from other artifacts), but that they can be used as an initial framework for software design.

Examples of Domain-Oriented Design Environments

Over the last eight years we have created DODEs for user interface design, COBOL programming, lunar habitat design, graphics design, multimedia design, voice dialog design, and computer network design (for details, see [Fischer 1994a]). Here we briefly describe the DODEs for voice dialog and computer network design.

Example1 of a DODE: Voice Dialog Design

The Voice Dialog Design Environment (VDDE) [Repenning, Sumner 1995; Sumner 1995] is a DODE developed in collaboration with US WEST Advanced Technologies over the last four years. The objective of the VDDE system was to improve the design practice of voice dialog designers by supporting an environment in which (1) they could focus on the task rather than on the computer, (2) they were assisted in the construction of the artifact (e.g., with critics and simulations) and (3) the evolving design could serve as an "object-to-think-with" and an "object-to-talk-about" for all stakeholders.

VDDE (see Figure 1) provides a construction kit that allows designers to quickly sketch the flow of an audio interface by arranging domain-oriented design units such as voice menus and prompts into a flow chart-style representation. Designers can hear what their audio interface design sounds like by attaching

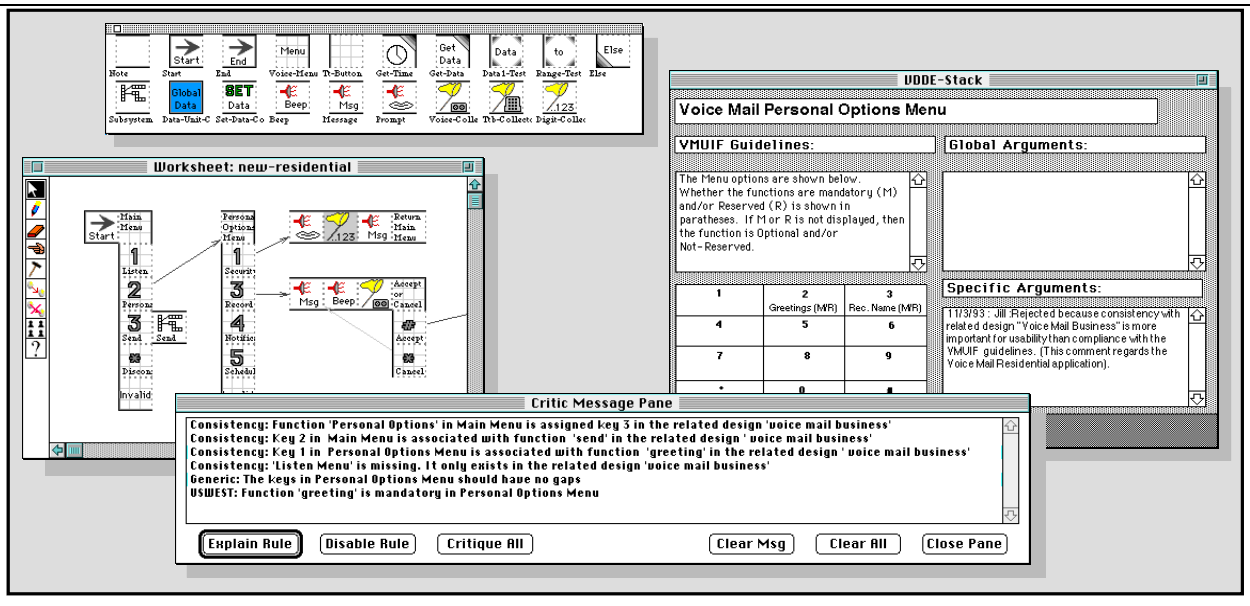

Figure 1: The Voice Dialog Design Environment

audio recordings to components in the interface and simulating the design. Computational design critics [Fischer et al. 1991a] embedded in the system watch designers' actions and comment on potentially problematic aspects of the design under construction. In the development and use of the system we observed the following evolutionary processes: Innovations in the voice dialog design domain arose both from within a particular design group and from the introduction of new design groups and new voice dialog products into the VDDE collaboration. To address these changes in the domain, we continually evolved what should be represented in VDDE and how it should be represented.

To determine the appropriate level of abstractions, we employed a system design approach that was both problem-centered and participatory. A collaborative process was followed in which voice dialog designers and system developers worked together to design and evolve domain-specific abstractions through use. VDDE evolved through repeated attempts at solving real voice dialog design problems. Overall, its "domain-orientation" was the result of analyses of existing design tools and representations, analyses of voice dialog products in the marketplace, and collaborative design sessions over the course of the project. As breakdowns in VDDE's abstractions were encountered, new design units were added or existing design units were modified or removed. In this manner, both the design units shown in the gallery (see Figure 1) and the representation of conditional actions in the worksheet underwent substantial evolution over the course of the VDDE project. The development provided strong support for the adequacy as well as the need for future development of DODEs.

In Figure 1, designers select building blocks from the gallery (top window) and arrange them in a worksheet (left window) to create a graphic representation of the audio interface design. A critiquing component analyzes the design for compliance with interface guidelines and product consistency. Possible problems are signaled in the critic message pane (lower window). The designer can select a critic message and elect to see the rationale behind the rule. The designer can also add more arguments into the hypermedia knowledge-base (right window).

In addition to developing a working system used by domain designers, careful assessment studies of all the processes and work products developed in the context of this major design effort were conducted (for details see [Sumner 1995]), resulting in the following: (a) VDDE helped to gain a deeper understanding of the strength and limitations of our component architecture and the process model described below, and (b) the system (including the substrates used, the application family of voice dialog designs, and individual artifacts) truly *evolved* over a period of four years.

Example2 of a DODE: Computer Network Design

We have developed several DODEs in the domain of computer network design [Reeves 1993; Shipman 1993; Sullivan 1994]. Similar to VDDE, these DODEs include the following domain-oriented components (see Figure 2):

- a palette containing objects of the domain (see Figure 2, (2))
- a work sheet supporting the construction of a network (see Figure 2, (3))
- a specification sheet allowing the articulation of design goals and constraints so that the system understands more about particular design situations and gives guidance and suggestions relevant to those situations (see Figure 2, (4))
- an argumentation component supporting the capture of design rationale in a WWW-based group memory system (see Figure 2, (1)), and
- a catalog of existing designs enabling design by modification rather than from scratch (see Figure 2, (5)).

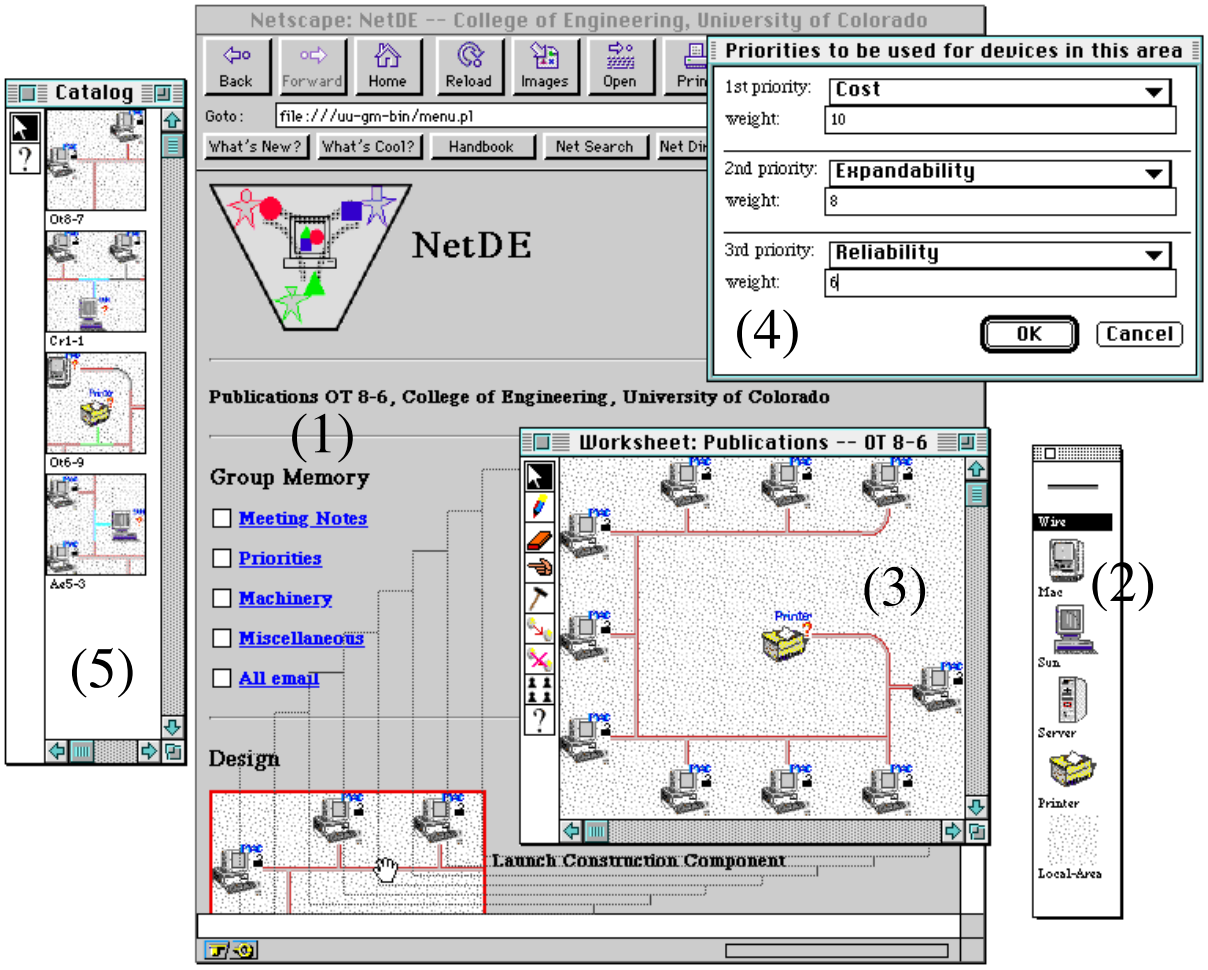

Figure 2: A DODE for Computer Network Design<u>**The Multifaceted Architecture: A Domain-Independent Architecture for DODEs**</u>

We have developed a domain-*independent* architecture for DODEs based on our theoretical and conceptual framework, which contains the essential components of DODEs. The links between the components are crucial for the synergy of the environment. Through domain-oriented instantiation, our architecture provides the foundation for design support tools and information repositories that reflect the real-world contexts of the design processes. This conceptual framework is explained in detail in [Fischer 1994a].

The architecture (see Figure 3) contains design creation tools in the form of a construction component and a specification component. The construction component is the principal medium for modeling a design. Design includes composition using elements from the palette and modification of previous design from the catalog (see Figures 1 and 2). The specification component [Fischer, Nakakoji 1991; Sullivan 1994] allows designers to describe abstract characteristics of the design they have in mind (e.g., low cost, supports email). The specification provides the system with an explicit and computationally tractable representation of the user's goals.

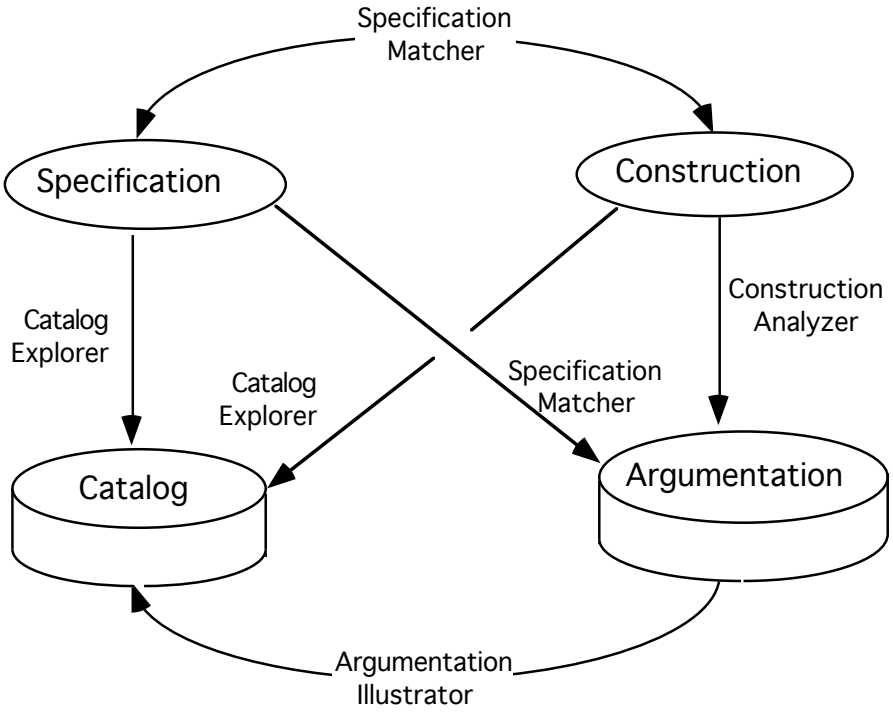

Figure 3: The Multifaceted Architecture

Design information repositories are provided in the form of argumentation [Moran, Carroll 1996] and catalogs [Kolodner 1993]. The argumentative hypermedia component contains design rationale that users can annotate and add to as it emerges during the design process. The catalog component provides a collection of previously constructed designs and is intended to support reuse and case-based reasoning.

The knowledge-based linking mechanisms integrate different facets of the DODE architecture (for details see [Fischer 1994a]):

- A specification matcher compares a specified design profile to a particular artifact design. This can be achieved by critics [Fischer et al. 1991a] that comment on the given design with respect to both stored arguments and the desired profile of the design as described in the specification.
- A construction analyzer is a critiquing system that analyzes the design construction for compliance with a set of rules that are yet another representation of domain knowledge. When such a critic fires, it provides a pointer to an entry of the argumentative hypermedia component. This entry explains the domain knowledge represented by the critic. It is left up to the designer to choose whether to modify the design in response to a critic message.
- An argumentation illustrator helps users understand the information given in the argumentation-base by finding relevant catalog examples that illustrate possibly abstract concepts.
- A catalog explorer helps users search the catalog space according to the task at hand by retrieving design examples similar to the current construction and specification situation.

Integration. The multi-faceted architectures derives its essential value from the *integration* of its components. Used individually, the components are unable to achieve their full potential. Used in combination, each component augments the values of the others, forming a synergistic whole. At each stage in the design process, the partial design embedded in the design environment serves as a stimulus to users, and suggests what they should attend to next. Links among the components of the architecture

are supported by various mechanisms (see Figure 3). The construction analyzer is a critiquing system [Fischer et al. 1991a] that provides access to relevant information in the argumentative issue base. The firing of a critic signals a breakdown to users and provides them with an entry into the exact place in the argumentative hypermedia system where the corresponding argumentation is located. The explanation given in argumentation is often highly abstract and very conceptual. Concrete design examples that match the explanation help users to understand the concept. The argumentation illustrator helps users to understand the information given in the argumentative hypermedia by finding a catalog example that illustrates the concept. The catalog explorer helps users to search the catalog space according to the task at hand. It retrieves design examples similar to the current construction situation, and orders a set of examples by their appropriateness to the current specification.

Seeding, Evolutionary Growth, Reseeding: The SER Process Model for DODEs

Design in real-world situations deals with complex, unique, uncertain, conflicted, and unstable situations of practice. Design knowledge as embedded in DODEs will never be complete because design knowledge is tacit (i.e., competent practitioners know more than they can say) [Polanyi 1966], and additional knowledge is triggered and activated by actual use situations leading to breakdowns. Because these breakdowns [Fischer 1994b; Winograd, Flores 1986] are experienced by the users and not by the developers, computational mechanisms that supporting end-user modifiability are required as an intrinsic part of a DODE.

We distinguish three intertwined levels; the interaction of these levels forms the essence of our seeding, evolutionary growth, reseeding model:

- On the *conceptual framework level*, the multifaceted, domain-independent architecture constitutes a framework for building evolvable complex software systems.
- When this architecture is instantiated in a domain (e.g., voice dialog design, computer network design), a domain-oriented design environment (representing an application family) is created on the *domain level*.
- On the artifact level, individual artifacts in the domain are developed by exploiting the information contained in the generic DODE.

Figure 4 illustrates the interplay of those three layers in the context of our seeding, evolutionary growth, reseeding model. Darker gray indicates knowledge domains close to the computer, whereas white emphasizes closeness to the design work in a domain. The figure illustrates the role of different professional groups in the evolutionary design: the *environment developer* (close to the computer) provides the domain-independent framework and instantiates it into a DODE in collaboration with the domain designers (knowledgeable domain workers who use the environment to design artifacts).

The evolution of complex systems in the context of this model can be characterized by the following major processes (details can be found in [Fischer et al. 1994]):

A **seed** will be created through a participatory design process between environment developers (software designers) and domain designers (voice dialog or network professionals). It will evolve in response to its use in new design projects because requirements fluctuate, change is ubiquitous, and design knowledge is tacit. Postulating the objective of a seed (rather then a complete domain model or a complete knowledge base) sets our approach apart from other approaches in knowledge-based systems development and emphasizes evolution as the central design concept.

Evolutionary growth takes place as domain designers use the seeded environment to undertake specific projects for clients. During these design efforts, new requirements may surface, new components may come into existence, and additional design knowledge not contained in the seed may be articulated. During the evolutionary growth phase, the environment developers are not present, thus making end-user modification a necessity rather than a luxury (at least for small-scale evolutionary changes). We have addressed this challenge with end-user modifiability [Eisenberg, Fischer 1994; Fischer, Girgensohn 1990], and end-user programming [Ambach, Perrone, Repenning 1995; Nardi 1993].

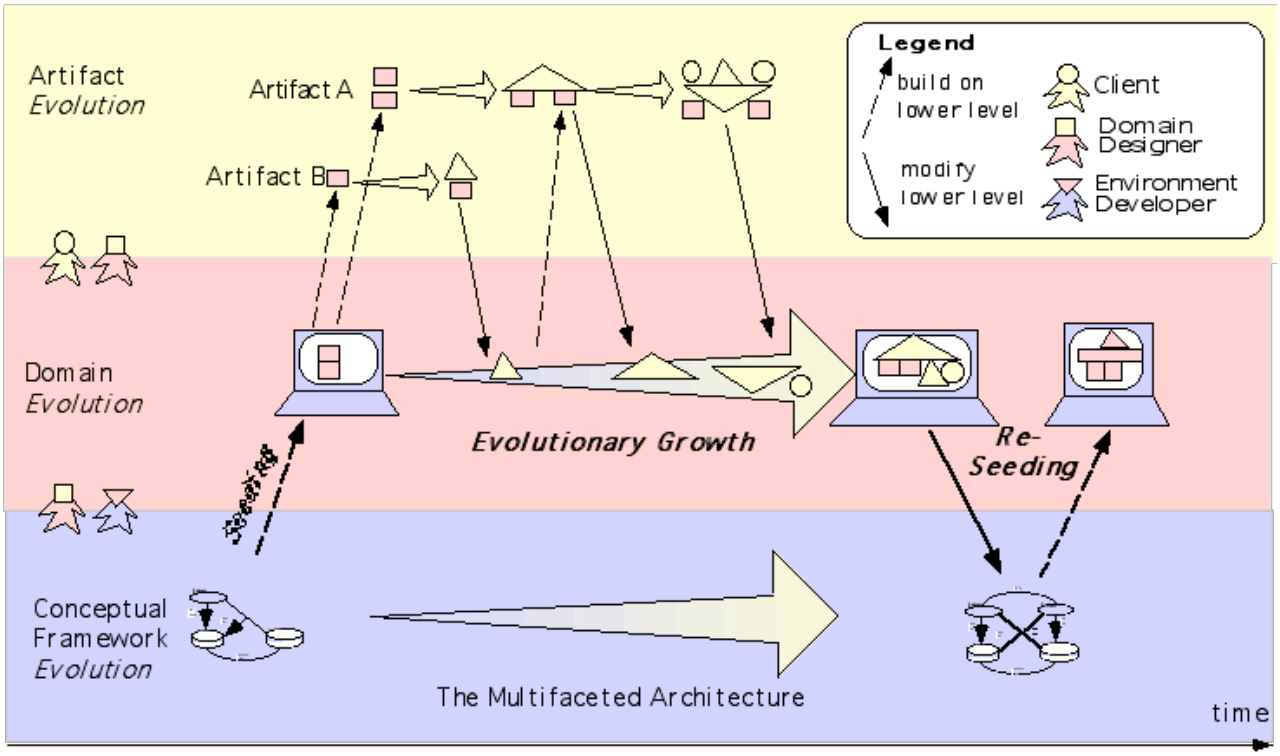

Figure 4: The SER Model: A process model for the development and evolution of DODEs

Reseeding, a deliberate effort of revision and coordination of information and functionality, brings the environment developers back to collaborate with domain designers to organize, formalize, and generalize knowledge added during the evolutionary growth phases. Organizational concerns [Grudin 1991; Terveen, Selfridge, Long 1993] play a crucial role in this phase. For example, decisions have to be made as to which of the extensions created in the context of specific design projects should be incorporated in future versions of the generic design environment. Drastic and large-scale evolutionary changes occur during the reseeding phase.

Assessment of DODEs

Knowledge-Based Systems Must Support the Integration of Working and Learning

Problems Of Skilled Domain Workers In Evolving, High Functionality, Technological Domains. Technologically oriented design fields are growing and changing at an alarming rate. Learning everything in advance in high functionality applications as embedded in knowledge-based systems is impossible because there are too many things to learn. The rapidly changing nature of available design objects (especially in all fields related to information technology such as computer network design) poses the problem that our knowledge needs to be updated constantly. The large and growing

discrepancy between the amount of potentially relevant knowledge and the amount users can know and remember makes support for learning on demand one of the most important activities. In high functionality systems "absolute experts" (in the sense of people who know everything) no longer exist [Draper 1984; Sullivan 1994]. The rapidly evolving nature of high functionality systems implies that being expert at time x does not mean one is an expert at a later time y. Systems that allow learning to take place within the context of real problem-solving situations must avoid the "production paradox" [Carroll, Rosson 1987], in which learning is inhibited by lack of time and working is inhibited by lack of knowledge. Learners must regard the time and effort invested in learning to be immediately worthwhile for the task at hand—not merely for some putative long-term gain.

Beyond Tool-Knowledge: Domain-Orientation. Domain-orientation provides a context, or grounding, so that learning can take place. Without domain-orientation, the user faces only abstract tools, and there are considerable problems when trying to apply those tools to a particular problem. Domain-oriented environments allow the user to apply newly learned knowledge in a manner that clarifies technical abstractions or ambiguities.

Supporting Designers in their Own Doing. System-building efforts in support of learning on demand [Fischer 1991] face the challenge of how a system can relinquish control of task selection yet maintain knowledge of users' goals, plans, and background knowledge. How can such systems be designed to function effectively in large solution spaces? By modeling problem domains with design environments. rather than representing solutions to individual problems, design environments support contextualized information access, which has as its goal to deliver the right knowledge in the context of a problem or service at the right moment for a human professional to consider. A partial understanding of the task at hand (as expressed by a partial specification and a partial construction) allows the system to prioritize information spaces in support of learning on demand.

Learning Embedded in the Context of Working. Embedding learning in the context of working is a promising approach for addressing such problems information overload, high functionality systems, and the rapid change of our world because: (1) it contextualizes learning by allowing it to be integrated into work rather than relegating it to a separate phase; (2) it lets learners see for themselves the usefulness of new knowledge for actual problem situations, thereby increasing the motivation for learning new things; and (3) it makes new information relevant to the task at hand, thereby leading to more informed decision making, better products, and improved performance.

Knowledge-Based Systems Must be Open, Rather Than Closed Systems

Design environments deal with complex and open-ended domains in which long-term users build extensive catalogs of personalized creative work. In contrast, non-programmable systems—systems in which users are forced to make choices by selection among fixed sets of alternatives (e.g., via menus or dialog boxes)—are rarely capable of providing users with the means for achieving their work; users' tasks eventually outstrip the capabilities provided by such systems. As a result, DODEs need means by which users can extend the functionality of their applications, building progressively more complex vocabularies and "languages of design." We have only scratched the surface of what would be possible if end users could freely program their own applications [Nardi 1993]. DODEs need be equipped with an end-user programming language This, in turn, implies certain desiderata for such an environment: interactivity, learnability, and expressiveness co-adaptivity [Mackay 1992] and malleable (adaptable and adaptive) systems [Fischer 1993] within the domain of the application. Figure 5 illustrates the duality that users of knowledge-based systems can learn from these systems but at the time need to be able to contribute new knowledge to a system.

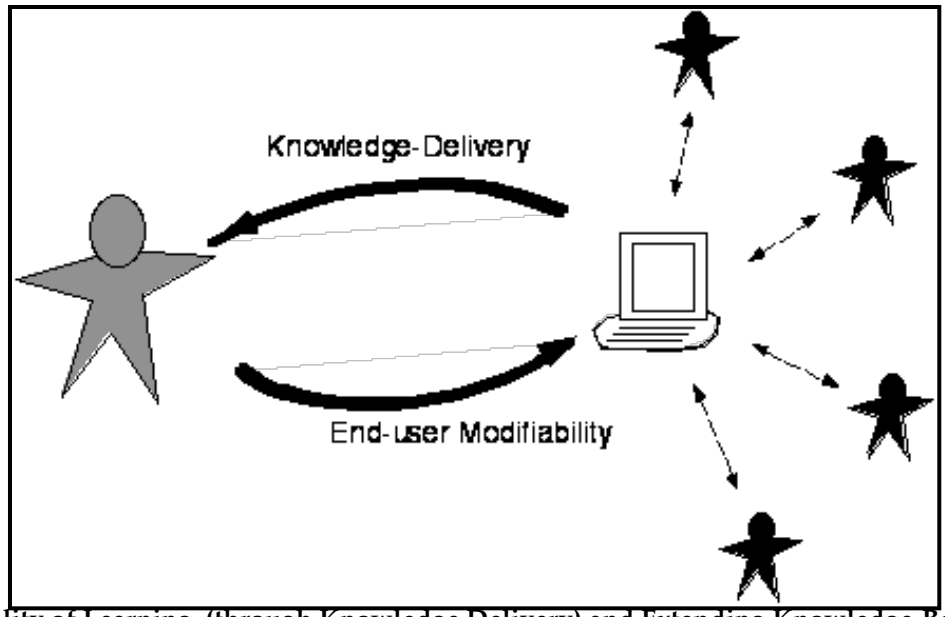

Figure 5: Duality of Learning (through Knowledge Delivery) and Extending Knowledge-Based Systems (through End-user Modifiability)

Knowledge-Based Systems Need to Evolve

Evolution of complex systems is a ubiquitous phenomenon. Design approaches based on the assumption of complete and correct requirements do not correspond to the realities of the world [CSTB 1990; Curtis, Krasner, Iscoe 1988]. Substantial costs are due to the fact that a considerable amount of essential information (such as design rationale [Fischer et al. 1991b]) is lost during development and must be reconstructed by the designers who maintain and evolve the system. Design methodologists (e.g., [Rittel 1984; Schoen 1983]) demonstrate with their work that the design of complex systems requires the integration of problem framing and problem solving, and they argue convincingly that one cannot gather information meaningfully unless one has understood the problem, but one cannot understand the problem without information about it. New requirements emerge during development because they cannot be identified until portions of the system have been designed and implemented. The conceptual structures underlying knowledge-based 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.

Evolution in DODEs. Our experience with DODEs clearly indicates that DODEs themselves as well as the artifacts created with them need to evolve. The ability of a DODE to coevolve with the artifacts created within it makes the DODE architecture the ideal candidate for creating evolvable application families. We believe that reseeding is critical to sustain evolutionary development. With design rationale captured, communication enhanced, and end-user modification available, developers have a rich source of information to evolve the system in the way users really need it.

Our research provides theoretical and empirical evidence that requirements for such systems cannot be completely specified before system development occurs. Our experience can be summarized in the following principles:

- *Software systems must evolve—they cannot be completely designed prior to use*. Design is a process that intertwines problem solving and problem framing. Software users and designers will not fully determine a system's desired functionality until that system is put to use.

- *Software systems must evolve at the hands of the users*. End users experience a system's deficiencies; subsequently, they have to play an important role in driving its evolution. Software systems need to contain mechanisms that allow end-user modification of system functionality.
- *Software systems must be designed for evolution*. Through our previous research in software design, we have discovered that systems need to be designed *a priori* for evolution. Software architectures need to be developed for software that is designed to evolve.

Experiences with DODEs

Domain Orientation: Situated Breakdowns and Design Rationale. Complex systems evolve faster if they can build on stable subsystems [Simon 1981]. Domain-oriented systems are rooted in the context of use in a domain. Although the DODE approach itself is generic, each of its applications is a particular domain-oriented system. Our emphasis on *domain-oriented* design environments acknowledges the importance of situated and contextualized communication and design rationale as the basis for *effective* evolutionary design. There is ample evidence in our work that human knowledge is tacit [Polanyi 1966] and that some of it will be activated only in actual problem situations. In early knowledge-based system-building efforts, there was a distinct knowledge acquisition phase that was assumed to lead to complete requirements — contrary to our assumption of the SER model (see Figure 3). The notion of a "seed" in the SER model emphasizes our interpretation of the initial system as a catalyst for evolution — evolution that is in turn supported by the environment itself.

End-User Modification and Programming for Communities: Evolution at the Hands of Users. Because end users experience breakdowns and insufficiencies of a DODE in their work, *they* should be able to report, react to, and resolve those problems. Mechanisms for end-user modification and programming are therefore a cornerstone of evolvable systems. At the core of our approach to evolutionary design lies the ability of end users (in our case, domain designers) to make significant changes to system functionality, and to share those modifications within a community of designers. DODEs make end-user modifications feasible because they support interaction at the domain level. We don't assume that all domain designers will be willing to make or even be interested in making, system changes, but within local communities of software use there often exist local developers and power users [Nardi 1993] who are interested in and capable of performing these tasks.

Assessment of the Multifaceted Architecture. The multifaceted architecture derives its essential value from the *integration* of its components. Used individually, the components are unable to achieve their full potential. Used in combination, each component augments the values of the others, forming a synergistic whole to support evolutionary design. At each stage in the design process, the partial design embedded in the design environment serves as a stimulus to users, focuses their attention, and enriches the "back-talk" of a design situation [Schoen 1983] by signaling breakdowns and by making task-relevant argumentation and catalog examples available [Fischer et al. 1991b].

Breakdowns occur when domain designers cannot carry out the design work with the existing DODE. Extensions and criticism drive the evolution on all three levels: Domain designers directly modify the artifacts when they build them (artifact evolution), they feed their modifications back into the environment (domain evolution), and — during a reseeding phase — even the architecture may be revised (conceptual framework evolution).

The support of DODEs for long-term, indirect communication [Fischer et al. 1992] between original developers and designers who need to evolve the generic DODE as well as the individual artifacts created within a DODE [Fischer et al. 1992] is critical and of particular importance in situations for which (1) direct communication is impossible, impractical, or undesirable; (2) communication is shared

around artifacts; or (3) designed artifacts continue to evolve over long periods of time (e.g., over months or years). DODEs provide essential mechanisms of which designers are informed within the context of their work.

Assessment of the SER Model. The SER model is motivated by how large software systems, such as Emacs, Symbolics' Genera, Unix, and the X Window System, have evolved over time. In such systems, users develop new techniques and extend the functionality of the system to solve problems that were not anticipated by the system's initial authors. New releases of the system often incorporate ideas and code produced by users. In the same way that these software systems are extensible by programmers who use them, DODEs need to be extended by domain designers who are neither interested in nor trained in the (low-level) details of computational environments.

Beyond Knowledge Acquisition. Knowledge acquisition is a crucial issue in the creation of effective information systems of all types (including expert systems, hypermedia systems, and design environments). There have been two extreme approaches: one is to input information in advance of use, typified by expert systems [Buchanan, Shortliffe 1984], and the other is to start with an empty system and allow its information base to grow and become structured as a consequence of use, characterized by initial proposals for argumentative hypertext [Moran, Carroll 1996]. Neither approach is adequate for the information needs of designers.

The "put-all-the-knowledge-in-at-the-beginning" approach fails for numerous reasons. It is inadequate for domains in which the domain knowledge undergoes rapid changes (the computer network domain being a prime example). Traditional knowledge acquisition approaches, which require domain designers to articulate their knowledge outside the context of problem solving or during an initial knowledge acquisition phase, fail to capture *tacit* knowledge.

The "just-provide-an-empty-framework" approach requires too much work of designers in the context of a specific project. The difficulties of capturing design knowledge from design projects are well known [Fischer et al. 1991b]. The act of documenting interferes with the thinking process itself, disrupting design and requiring substantial time and effort that designers would rather invest in design. Designers typically find it difficult to structure their thoughts in a given format, regardless of the format used. In addition, domain designers often lack the knowledge and the interest to formalize knowledge so it can be computationally interpreted.

The SER model explores interesting new ground between the two extremes of "put-all-the-knowledge-in-at-the-beginning" and "just-provide-an-empty-framework." Designers are more interested in their design task at hand than in maintaining the knowledge base. At the same time, important knowledge that should be captured is produced during daily design activities. Rather than expect designers to spend extra time and effort to maintain the knowledge base as they design, we provide tools to help designers record information quickly and without regard for how the information should be integrated with the seed. Knowledge base maintenance is periodically performed during the reseeding phases by environment developers and domain designers in a collaborative activity.

Conclusions

The DODE approach is not just an experience report on one system. It challenges many assumptions and provides an alternative to other existing approaches to the design of knowledge-based systems [Bobrow 1991].

The domain orientation of a design environment enriches (1) the amount of support that a knowledge-based system can provide, and (2) the shared understanding among stakeholders. Design knowledge

includes domain concepts, argumentation, case-based catalogs, and critiquing rules. The appeal of the DODE approach lies in its compatibility with an emerging methodology for design, views of the future as articulated by practicing software engineering experts, reflections about the success of the expert system approach, findings of empirical studies, and the *integration* of many recent efforts to tackle specific issues in software design (e.g., recording design rationale, supporting case-based reasoning, or creating artifact memories).

References

[Ambach, Perrone, Repenning 1995] J. Ambach, C. Perrone, and A. Repenning, Remote Exploratoriums: Combining Networking and Design Environments, Computers and Education, Vol. 24, No. 3, pp. 163-176.

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

[Buchanan, Shortliffe 1984] B.G. Buchanan, and E.H. Shortliffe, Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project, Addison-Wesley Publishing Company, Reading, MA.

[Carroll, Rosson 1987] J.M. Carroll, and M.B. Rosson, Paradox of the Active User, in J.M. Carroll (ed.), Interfacing Thought: Cognitive Aspects of Human-Computer Interaction, The MIT Press, Cambridge, MA, pp. 80-111.

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

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

[Dawkins 1987] R. Dawkins, The Blind Watchmaker, W.W. Norton and Company, New York - London.

[Draper 1984] S.W. Draper, The Nature of Expertise in UNIX, in Proceedings of INTERACT'84, IFIP Conference on Human-Computer Interaction, Elsevier Science Publishers, Amsterdam, pp. 182-186.

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

[Eisenberg, Fischer 1994] M. Eisenberg, and G. Fischer, Programmable Design Environments: Integrating End-User Programming with Domain-Oriented Assistance, in Human Factors in Computing Systems, CHI'94 Conference Proceedings (Boston, MA), pp. 431-437.

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

[Fischer 1993] G. Fischer, Shared Knowledge in Cooperative Problem-Solving Systems - Integrating Adaptive and Adaptable Components, in M. Schneider-Hufschmidt, T. Kuehme and U. Malinowski (eds.), Adaptive User Interfaces - Principles and Practice, Elsevier Science Publishers, Amsterdam, pp. 49-68.

[Fischer 1994a] G. Fischer, Domain-Oriented Design Environments, Automated Software Engineering, Vol. 1, No. 2, pp. 177-203.

[Fischer 1994b] G. Fischer, Turning Breakdowns into Opportunities for Creativity, Knowledge-Based Systems, Special Issue on Creativity and Cognition, Vol. 7, No. 4, pp. 221-232.

[Fischer, Girgensohn 1990] G. Fischer, and A. Girgensohn, End-User Modifiability in Design Environments, in Human Factors in Computing Systems, CHI'90 Conference Proceedings (Seattle, WA), New York, pp. 183-191.

[Fischer et al. 1992] G. Fischer et al., Supporting Indirect, Collaborative Design with Integrated Knowledge-Based Design Environments, Human Computer Interaction, Special Issue on Computer Supported Cooperative Work, Vol. 7, No. 3, pp. 281-314.

[Fischer et al. 1991a] G. Fischer et al., The Role of Critiquing in Cooperative Problem Solving, ACM Transactions on Information Systems, Vol. 9, No. 2, pp. 123-151.

[Fischer et al. 1991b] G. Fischer et al., Making Argumentation Serve Design, Human Computer Interaction, Vol. 6, No. 3-4, pp. 393-419.

[Fischer et al. 1994] G. Fischer et al., Seeding, Evolutionary Growth and Reseeding: Supporting Incremental Development of Design Environments, in Human Factors in Computing Systems, CHI'94 Conference Proceedings (Boston, MA), pp. 292-298.

[Fischer, Nakakoji 1991] G. Fischer, and K. Nakakoji, Making Design Objects Relevant to the Task at Hand, in Proceedings of AAAI-91, Ninth National Conference on Artificial Intelligence, AAAI Press/The MIT Press, Cambridge, MA, pp. 67-73.

[Greenbaum, Kyng 1991] J. Greenbaum, and M. Kyng, Design at Work: Cooperative Design of Computer Systems, Lawrence Erlbaum Associates, Inc., Hillsdale, NJ.

[Grudin 1991] J. Grudin, Interactive Systems: Bridging the Gaps Between Developers and Users, Computer, Vol. 24, No. 4, pp. 59-69.

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

[Kolodner 1993] J.L. Kolodner, Case-Based Reasoning, Morgan Kaufmann Publishers, Inc., San Mateo, CA.

[Mackay 1992] W.E. Mackay, Co-adaptive Systems: Users as Innovators, in CHI'92 Basic Research Symposium,

[Moran, Carroll 1996] T.P. Moran, and J.M. Carroll, Design Rationale: Concepts, Techniques, and Use, Lawrence Erlbaum Associates, Inc., Hillsdale, NJ.

[Nardi 1993] B.A. Nardi, A Small Matter of Programming, The MIT Press, Cambridge, MA.

[Polanyi 1966] M. Polanyi, The Tacit Dimension, Doubleday, Garden City, NY.

[Popper 1965] K.R. Popper, Conjectures and Refutations, Harper & Row, New York, Hagerstown, San Francisco, London.

[Reeves 1993] B.N. Reeves, Supporting Collaborative Design by Embedding Communication and History in Design Artifacts, Ph.D. Thesis, Department of Computer Science, University of Colorado at Boulder.

[Repenning, Sumner 1995] A. Repenning, and T. Sumner, Agentsheets: A Medium for Creating Domain-Oriented Visual Languages, in Computer, IEEE Computer Society, Los Alamitos, CA, pp. 17-25.

[Resnick, Levine, Teasley 1991] L.B. Resnick, J.M. Levine, and S.D. Teasley, Perspectives on Socially Shared Cognition, American Psychological Association, Washington, D.C.

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

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

[Shipman 1993] F. Shipman, Supporting Knowledge-Base Evolution with Incremental Formalization, Ph.D. Thesis, Department of Computer Science, University of Colorado at Boulder.

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

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

[Sumner 1995] T. Sumner, Designers and Their Tools: Computer Support for Domain Construction, Ph.D. Thesis, University of Colorado at Boulder.

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

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