Seeding, Evolutionary Growth, and Reseeding: Constructing, Capturing, and Evolving Knowledge in Domain-Oriented Design Environments

Gerhard Fischer
Center for LifeLong Learning and Design (L^{3}D)
Department of Computer Science and Institute of Cognitive Science
Campus Box 430, University of Colorado, Boulder, Colorado 80309
gerhard@cs.colorado.edu

Abstract

We live in a world characterized by evolution – that is, by ongoing processes of development, formation, and growth in both natural and human-created systems. Biology tells us that complex, natural systems are not created all at once but must instead evolve over time. We are becoming increasingly aware that evolutionary processes are ubiquitous and critical for technological innovations as well. This is particularly true for complex software systems because these systems do not necessarily exist in a technological context alone but instead are embedded within dynamic human organizations.

The Center for LifeLong Learning and Design (L^{3}D) at the University of Colorado has been involved in research on software design and other design domains for more than a decade. We understand software design as an evolutionary process in which system requirements and functionality are determined through an iterative process of collaboration among multiple stakeholders, rather than being completely specified before system development occurs. Our research focuses on the following claims about software systems embedded within dynamic human organizations: (1) they must evolve because they cannot be completely designed prior to use, (2) they must evolve to some extent at the hands of the users, and (3) they must be designed for evolution.

Our theoretical work builds upon our existing knowledge of design processes and focuses on a software process model and architecture specifically for systems that must evolve. Our theories are instantiated and assessed through the development and evolution of domain-oriented design environments (DODEs) – software systems that support design activities within particular domains and that are built specifically to evolve.

Keywords

design, domain-oriented design environments, evolution, end-user modification, knowledge construction, computer network design

Table of Contents

Note: This manuscript is a revised and extended version of the following paper — Gerhard Fischer: "Seeding, Evolutionary Growth and Reseeding: Constructing, Capturing and Evolving Knowledge in Domain-Oriented Design Environments" in A. Sutcliffe, D. Benyon and F. van Assche (eds.): *Domain Knowledge for Interactive System Design*, IFIP Series, Chapman & Hall, London, 1996, pp. 1-16.

1. Introduction

Software engineering research has been concerned historically with the transition from specification to implementation ("downstream activities") rather than with the problem of how faithfully specifications really address the problems to be solved ("upstream activities"). Many methodologies and technologies were developed to prevent *implementation disasters* (i.e., creating correct programs with respect to a given specification). The progress made to successfully reduce implementation disasters (e.g., structured programming, information hiding, etc.) allowed an equally relevant problem to surface: how to prevent *design disasters* [Lee, 1992] — meaning that a correct implementation with respect to a given specification is of little value if the specification does not adequately address the problem to be solved.

Design [Simon, 1981] in the context of our research approach refers to the broad endeavor of creating artifacts as exercised by architects, industrial designers, curriculum developers, composers, and others, rather than to a specific step in a software engineering life-cycle model (located between requirements and implementation). Domain-oriented design environments (DODEs) [Fischer, 1994] are computational environments that have been used for the design of software artifacts such as user interfaces, voice dialog systems, and Cobol programs, and have served equally well for the conceptual design of material artifacts such as kitchens, lunar habitats, and computer networks. The fundamental assumption behind our research is that DODEs will become as valuable and as ubiquitous in the future as compilers have been in the past. They will provide the design support most desirable and most needed to avoid design disasters and will serve as prototypes for other research efforts moving in the same direction, such as ARPA's research programs in domain-specific software architectures (DSSA) [Garlan & Shaw, 1994] and evolutionary design of complex systems (EDCS) [Salasin & Shrobe, 1995].

2. Domain-Oriented Design Environments

Domain-oriented systems [Fischer, 1994; Gamma et al., 1994; Sutcliffe, 1997] are rooted in the *context of use* in a domain. Whereas the *domain-oriented* design environments approach itself is generic, each of its applications is a particular domain-oriented system. Our emphasis on DODEs acknowledges the importance of situated and contextualized communication [Greenbaum & Kyng, 1991] and design rationale [Moran & Carroll, 1996] as the basis for *effective* evolutionary design. DODEs transcend the *domain modeling approach* [Prieto-Diaz & Arango, 1991] which assumes that there exists a common concept of a domain shared by all practitioners and the problem is simply to identify this model and codify it. These approaches do not pay enough attention to the situated and tacit nature of professional knowledge [Polanyi, 1966] (i.e., we know more than we can say), illustrating that domain knowledge is activated and constructed on an ongoing basis in actual problem situations. The domain construction approach underlying DODEs addresses these shortcomings by explicitly acknowledging that shared domain models do not de facto exist but are socially constructed and *evolved* [Lave, 1991] by communities of practice.

2.1 Basic Design Criteria for DODEs

DODEs emphasize a human-centered and domain-oriented approach facilitating communication about evolving systems among all stakeholders. The integration among different components of DODEs supports the co-evolution of specification and construction while allowing designers to access relevant knowledge at each stage within the software development process.

Understanding the Problem Is the Problem. The predominant activity in designing complex systems is the participants teaching and learning from each other [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. None of these carriers of knowledge can guarantee that their knowledge is superior or more complete compared to other people's knowledge. To overcome this "symmetry of ignorance" [Rittel, 1984], 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 [Rittel, 1984; Schön, 1983] demonstrate with their work the strong interrelationship between problem framing and problem solving. They argue convincingly that (1) information cannot be gathered meaningfully unless the problem is understood, 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.

Increasing the Shared Understanding Between Stakeholders. A consequence of the thin spread of application knowledge [Curtis et al., 1988] is that specification errors often occur when designers do not have sufficient application domain knowledge to interpret the customer's intentions from the requirement statements – a communication breakdown based on a lack of shared understanding. The main attribute of formal specifications is that they are "formal," which means that they are manipulable by mathematics and logic and interpretable by computers. As such, these representations are often couched in the language of the computational system. However, such representations are typically foreign and unintelligible to users and get in the way of trying to create a shared understanding among stakeholders. Domain orientation reduces the large conceptual distance between problem-domain semantics and software artifacts.

The Need for Change and Evolution. There are numerous fundamental reasons why systems cannot be done "right." Software systems model parts of our world. Our world evolves in numerous dimensions – as users have new needs, as new artifacts and technologies appear, as new knowledge is discovered, and as new ways of doing business are developed. Successful software systems need to evolve [Salasin & Shrobe, 1995]. System maintenance and enhancement need to become "first class design activities."

2.2 Architectures and Process Models for DODEs

Essential components developed in the context of our research for DODEs [Fischer, 1994] are:

A **multifaceted, domain-independent architecture,** including the following major components: (1) a construction kit providing a palette of domain building blocks; (2) an argumentative hypermedia system containing issues, answers, and arguments about the design domain and the design rationale for a specific application built within the domain; (3) a catalog consisting of a collection of prestored designs that illustrates the space of possible designs in the domain, and thereby supports reuse and case-based reasoning [Maiden & Sutcliffe, 1992]; (4) a specification component supporting the interaction among stakeholders in describing characteristics of the design they have in mind; and (5) a simulation component allowing designers to carry out "what-if" games to simulate various usage scenarios involving the artifact being designed.

Mechanisms for integration among components, including: (1) a specification matcher comparing a partial specification with a partial construction, (2) a critiquing component linking construction and argumentation and providing access to relevant information in the argumentative issue base, (3) an argumentation illustrator helping users to understand the information given in the argumentation base by finding relevant catalog examples that illustrate abstract concepts, and (4) a catalog explorer assisting users in retrieving design examples from the catalog similar to the current construction and specification situation.

A **process model** consisting of seeding, evolutionary growth, and reseeding (SER model) to account for the evolutionary nature of DODEs [Fischer et al., 1994]. The components and the integration mechanisms of the multifaceted architecture are illustrated in the context of a specific DODE, namely for computer network design, in section 3, and the SER model is discussed in section 4.

3. Evolutionary Design At Work: A Scenario From The Domain Of Computer Network Design

The following scenario illustrates how a DODE is applied to the exemplary domain of computer network design emphasizing the importance of evolution. The system described, NetDE, is a DODE for the domain of computer network design and incorporates and illustrates the following aspects of DODEs:

- Domain-oriented components that provide computer network designers the capability to easily create design artifacts.
- Features that allow the specification of design constraints and goals so that the system understands more about particular design situations and gives guidance and suggestions for designers relevant to those situations.
- Mechanisms that support the capture of design rationale and argumentation embedded within design artifacts so that they can best serve the design task.
- Mechanisms that support end-user modifiability so that the communities of practice of network designers experiencing deficiencies of NetDE can drive the evolution of the system.
- Features that increase communication between the system stakeholders.

The following scenario involves two network designers (D_1 and D_2) at the University of Colorado who have been asked to design a new network for clients within the Publications Group in the dean's office at the College of Engineering.

Evolution of Design Artifacts: Designing a New Network. D_1's clients are interested in networking ten newly purchased Macintosh Power PCs and a laser printer. Through a combination of email discussions and meetings, D_1 learns that the clients want to be able to share the printer, swap files easily, and send each other email. D_1 raises the issue of connecting to the Internet, and is told that the clients would be interested at some point, but not for the time being. It is also made clear that the clients had spent most of their budget on the computer hardware, and do not have much left over for sophisticated network services and tools.

As argued before and based on our previous work in network design, design specification and rationale come from a number of stakeholders, including network designers and clients, and are captured in different media including email and notes. To be most effective, design rationale needs to be stored in a way that allows access to it from the relevant places within a design [Fischer et al., 1996a].

D_1 invokes the NetDE system. A World-Wide Web (WWW) Browser appears on the desktop presenting a drawing of the College of Engineering. By selecting the "New Design" option, D_1 is presented an empty NetDE page that he names "Publications OT 8-6" after the office where the clients are located. The new page becomes a repository for all of the background information and rationale that D_1 has regarding the new network. This is achieved by sending all email and text files that D_1 has to the (automatically created) email address "Publications OT 8-6." NetDE insures that the WWW page immediately updates itself to show links to the received mails and files (Figure 1 (1)).

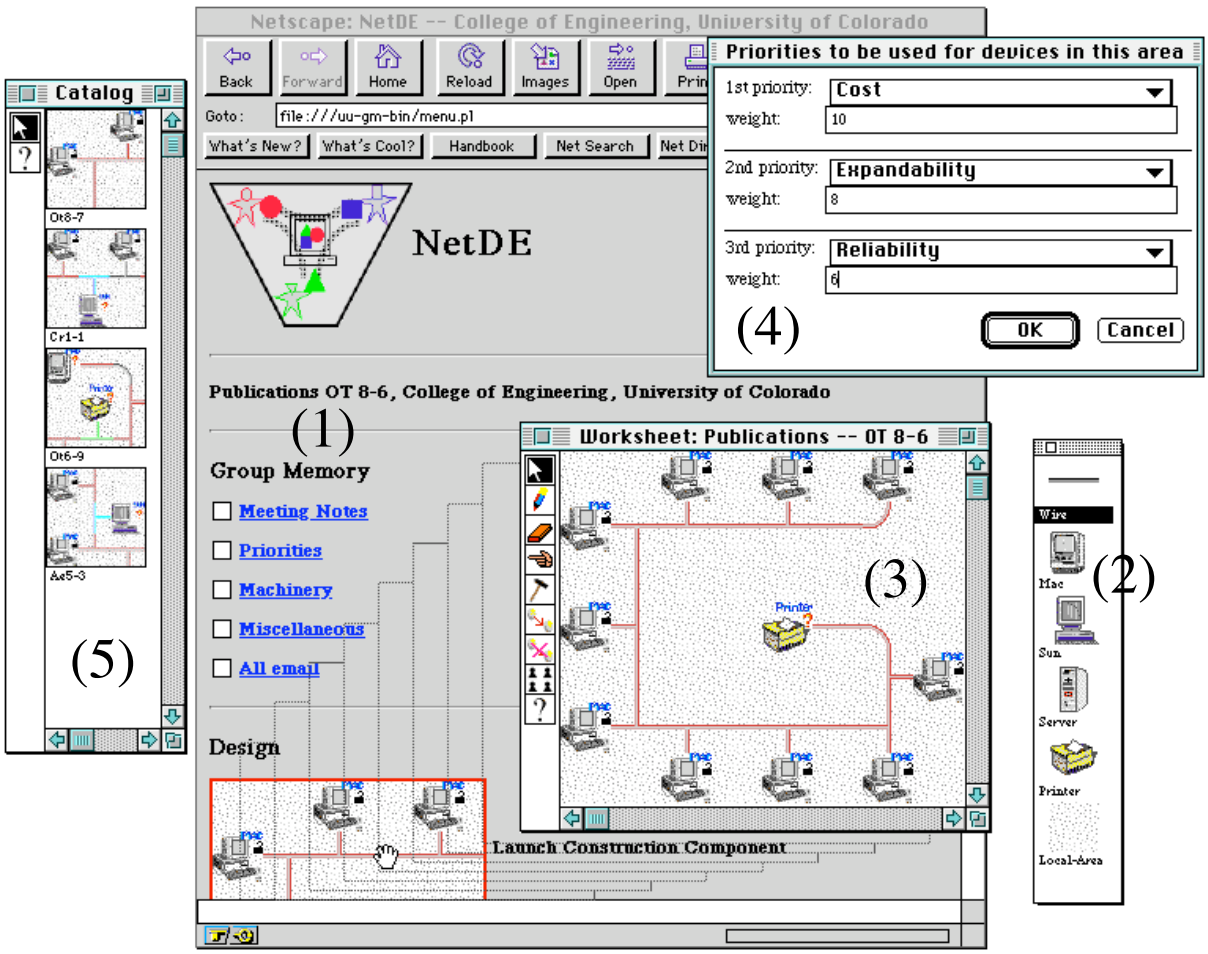

Figure 1: NetDE in Use

Selecting the "Launch Construction Component" option opens a palette of network objects (Figure 1 (2)). D_1 starts by specifying certain design constraints to the system (Figure 1 (4)). Immediately the Catalog (Figure 1 (5)) displays a selection of existing designs that have constraints similar to those specified by D_1. Selecting one of the designs represented in the Catalog moves that design into the worksheet (Figure 1 (3)) so that D_1 can modify it. D_1 changes the design to reflect the specific needs of the Publications Group.

NetDE is accessible through the World-Wide Web, so other network designers (D_2...D_n) can use it also. The existing designs represent contributions from the whole community of practice using NetDE for their work.

NetDE provides a domain-specific construction kit (the palette and the worksheet), and allows the specification of design constraints and goals. Using additional specification mechanisms, D_1 describes how the network will be used, and what kinds of networking services are desired. This is the first time D_1 has networked Macs, so he takes advantage of the NetDE critiquing feature [Fischer, 1994], which evaluates his design and compares it to the established design constraints. During evaluation, NetDE suggests the use of the EtherTalk network protocol and the PowerTalk email capabilities that come standard with Macs. D_1 agrees with this assessment because they limit the cost of the network. He finishes creating his design. The integration of specification, construction, catalog, and argumentation components is the characteristic strength of a DODE such as NetDE. These components and their interactions are critical to the "evolvability" of the system.

Evolution of NetDE. Several months pass, and Publications is interested in changing its network. D_1 is not available, so D_2 is to design the new changes. D_2 receives email from Publications indicating that their network needs have changed. They want to start publishing WWW pages and will need Internet access. They will also be using a Silicon Graphics Indy computer. They have received a substantial budget increase for their network.

First, D_2 accesses the NetDE page that describes the Publications network. She quickly reviews the current design and rationale to learn what has already occurred. She updates the design specification to reflect the fact that cost has become less important, and speed has become more important. Then she searches the NetDE palette to see if it has an icon representing the Indy. She does not find one, and realizes that it must be added. After reviewing the specs for the Indy from the Silicon Graphics Web Page, D_2 creates a new palette element for the Indy (Figure 2 (1)), and then defines its characteristics using NetDE's end user modifiability features (Figure 2 (2)). According to the company's specs, the Indy has built-in networking capabilities and understands the TCP/IP network protocol. D_2 enters this information, and the new icon appears in the palette. D_2 adds the Indy to the design, and NetDE indicates (by displaying different colored wires) that the two types of machines (Macs and the Indy) are using different network protocols. D_2 knows that Macs can understand TCP/IP protocol, so she changes the network's protocol to TCP/IP. After invoking NetDE's critiquing mechanism, D_2 receives a critiquing message indicating that the use of TCP/IP violates the easy file-sharing design constraint (Figure 2 (3)). After reading through some of the argumentation (Figure 2 (4)), D_2 learns that although file sharing is possible in TCP/IP with the Macs, it is not as easy as using EtherTalk. D_2 decides that this is not a constraint she would like to break, and asks some other network designers if there is a way to get the Indy to understand EtherTalk. D_2 learns that there is software the Indy can run to translate protocols, and she adds an annotation to the Indy object to reflect this.

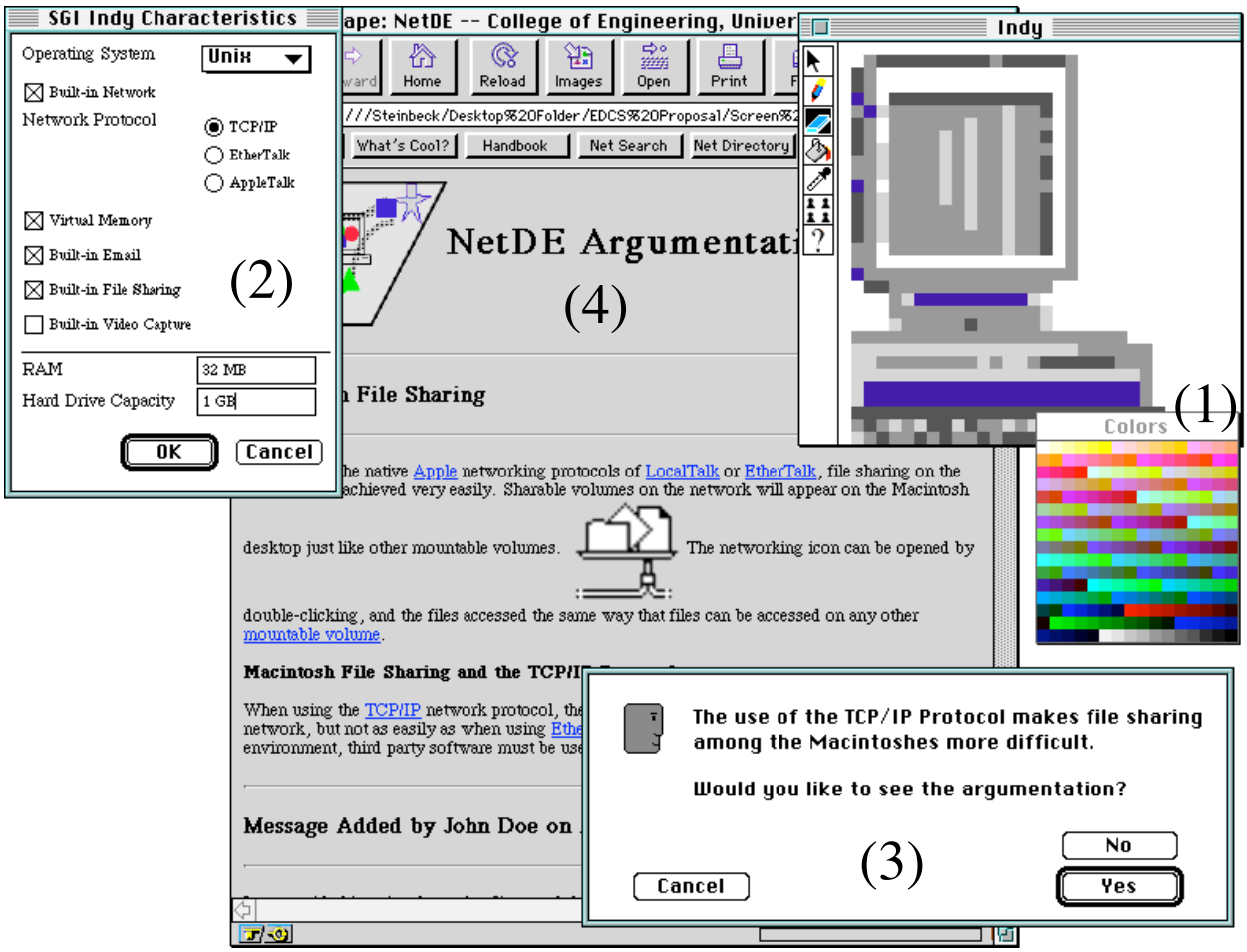

Figure 2: The Network Evolves

4. The Seeding, Evolutionary Growth, Reseeding (SER) Process Model For DODEs

4.1 The SER Model

As illustrated in the previous sections design knowledge as embedded in DODEs will never be complete because (1) design in real-world situations deals with complex, unique, uncertain, conflicted, and unstable situations of practice [Rittel, 1984]; (2) design knowledge is tacit (i.e., competent practitioners know more than they can say) [Polanyi, 1966]; and (3) additional knowledge is triggered and activated by actual use situations leading to *breakdowns*. Because these breakdowns are experienced by the users and not by the developers, computational mechanisms supporting end-user modifiability are required as an intrinsic part of a DODE.

We distinguish three intertwined levels of design activities and system developments; their interactions form the essence of our seeding, evolutionary growth, reseeding model (see Figure 3):

- On the *conceptual framework level* (lowest band in the figure), the multifaceted, *domain-independent* architecture constitutes a framework for building evolvable complex software systems.

- When this architecture is instantiated for a particular domain (e.g., computer network design), a DODE (representing an application family [Salasin & Shrobe, 1995]) is created on the *domain level* (middle band in the figure).
- Individual *artifacts* (e.g., the computer network of CU Boulder) in the domain are developed by exploiting the information contained in the DODE (highest band in the figure).

The SER model describes continual DODE evolution in terms of three *levels* (artifact, domain, and software architecture), three *phases* (seeding, evolutionary growth, and reseeding), and three *groups of stakeholders* (developers, domain designers, and clients). Figure 3 illustrates the interplay of those three layers in the context of our SER model. Darker gray indicates knowledge domains close to the computer, whereas the light band emphasizes closeness to the design work in a domain. The figure illustrates the role of different professional groups in the evolutionary design: *environment developers* (professional software designers) provide the domain-independent framework, and instantiate it into a DODE in collaboration with the *domain designers* (knowledgeable domain workers) who use the environment to design artifacts in collaboration with *clients*. In the figure, the dashed arrow from the DODE level and Artifact level indicates that artifacts are built using domain knowledge contained in the DODE (e.g., palette items, design rationale, catalog items). The fat arrows (along the time dimension) indicate that artifacts are not simply designed at one time, but instead evolve over time. The solid arrows from artifact to DODE level indicate that the design and evolution of artifacts produces new domain knowledge, which is stored in the design environment.

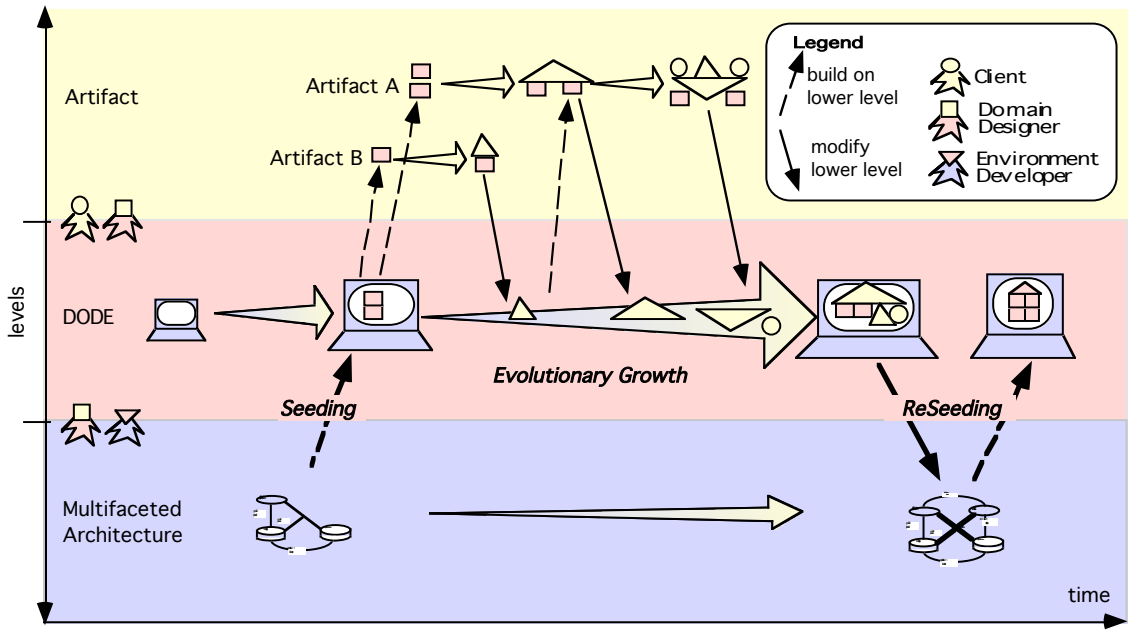

Figure 3: The SER Model: A process model for the development and evolution of DODES

The evolution of complex systems in the context of the SER model will be described below (details can be found in [Fischer et al., 1994]):

Seeding. A seed will be created through a participatory design process between environment developers and domain designers (e.g., computer 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-base systems development and emphasizes evolution as the central design concept.

The seed incorporates domain-specific knowledge into the domain-independent multifaceted architecture underlying the design environment. Seeding entails embedding as much knowledge as possible into all components of the architecture. But any amount of design knowledge embedded in DODEs will never be complete because (as argued before) real-world situations are complex, unique, uncertain, conflicted, and unstable, and knowledge is tacit, implying that additional knowledge is triggered and activated only by experiencing breakdowns in the context of specific use situations. The seed should provide a strong information base for evolution by giving users something to which to react.

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) [Fischer et al., 1996b]. New seed requirements are elicited by constructing and evaluating domain-oriented knowledge structures.

The seeding process for the NetDE DODE was driven by observations of network design sessions, prototypes of proposed system functionality, and discussions centered on the prototypes. Evaluation of the NetDE seed indicated that designers need support for communication in the form of critiques, reminders, and general comments. An important lesson we learned during the seeding of NetDE 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. In addition, information to seed NetDE was acquired from existing databases containing information about network devices, users, and the architectural layout of the building.

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 (in the scenario: access to the Internet is needed), new components may come into existence (in the scenario: a Silicon Graphics Indy computer will be used), and additional design knowledge not contained in the seed may be articulated (in the scenario: annotation to the Indy object that software exists for translating protocols). 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] and end-user programming [Nardi, 1993].

Reseeding, a deliberate effort of revision and coordination of information and functionality brings the environment developers back in to collaborate with domain designers to organize, formalize, and generalize knowledge added during the evolutionary growth phases. 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. Organizational concerns [Terveen et al., 1995] 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. Periodically, the growing information space must be structured, generalized, and formalized to increases the computational support the system is able to provide to designers [Shipman & McCall, 1994].

4.2 Illustrations of the SER Model with Examples from our Work

Collaboration and Communication Between Stakeholders as Facilitated by the SER Model. The SER model emphasizes several collaboration and communication processes among stakeholders. At the level of an individual artifact (e.g., a particular computer network evolving over many years), we have emphasized the need for long-term, indirect collaboration [Fischer et al., 1992], which takes place when an artifact functions and is repeatedly redesigned over a relatively long period of time.

Collaboration and communication between designers is not only asynchronous with respect to time and place, but it is *indirect* in the sense that groups of designers may have no opportunity to meet or communicate directly. This requires that the relevant design information such as design rationale is associated and embedded in the artifact. Long-term projects are unpredictable with regard to the team members and users who need to communicate. Support for collaboration and communication allows 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.

Evolution at the Different Levels of the SER Model. The SER model can be used to illustrate evolutionary processes within each of the three levels shown in Figure 3.

Evolution at the Conceptual Framework Level. Our work at this level started many years ago using object-oriented environments to decompose complex systems into modules and take advantage of inheritance among them. The lack of support for interaction between humans and problem domains in general object-oriented environments led to the development of domain-oriented construction kits. Although construction kits allowed us to create artifacts quickly, there was no support for evaluating the quality of an artifact [Schön, 1983]. This led to the development of critics and explanation components. Driven by our understanding of design as an argumentative process and the need to support "reflection-in-action" by making argumentation serve design, we added an argumentation component and a specification component to the framework. Accounting for the requirement that environments need to be changed by their users led to the development of end-user modification components, turning DODEs into programmable DODEs [Eisenberg & Fischer, 1994].

Evolution of the Domain. Computer network design has undergone dramatic changes over the last ten years, including new network devices, new design guidelines, new simulation support, and new design rationale. This evolution was driven by new needs and expectations of users as well as new technology (either responding to these needs and expectations or creating them). It is obvious that a DODE modeling this domain has to evolve in accordance with the evolution of the domain.

Evolution of Individual Artifacts. We have tracked the development of the computer networks within CU Boulder and the Computer Science Department specifically — and they have served us well to understand the processes associated with the evolution of an individual artifact. Analysis of the difficulties in evolving these complex artifacts over many years has been an important source for insight about our developments to capture design rationale and associate it with the artifact to support indirect, long-term collaboration.

System-Building Efforts in Support of the SER Model. We are in the process of developing several systems in support of the SER model, which are described briefly.

Group Interactive Memory Manager (GIMMe). GIMMe ([Fischer et al., 1996b] and http://www-l3d.cs.colorado.edu/~stefanie/GM_Info.html) is a World-Wide Web-based group memory system that is used for the capture and retrieval of design rationale [Moran & Carroll, 1996]. GIMMe supports the SER model by assisting users to understand the design decisions that led to the current artifact. It addresses the problem of how to capture design rationale, structure it for later reuse, and make it available to the users by letting it emerge as a by-product of normal work. This is critical because we know from empirical evidence that most design rationale systems have failed not because of the inadequacy of a computational substrate, but because they did not pay enough attention to the question, "who is the beneficiary and who has to do the work?" [Grudin, 1991].

Expectation Agents. Expectation Agents [Girgensohn et al., 1994] are computational structures that support communication between end users and developers of an interactive system during actual use situations [Henderson & Kyng, 1991]. They observe and analyze the reactions of end users to the system by not only relying on a small subset of end users but by reaching the whole (or a large subset) of the community of practice. If an Expectation Agent notices a discrepancy between actual system use and the designer's expectation, it will communicate the discrepancy to the designer. Expectation Agents are general mechanisms that support the concept of design-in-use and are used within the DODE architecture not only to notify designers when *existing* design expectations have not been met, but also when the domain designer adds *new* functionality to the DODE. Expectation Agents contribute to the SER model by supporting evolutionary growth based on discrepancies between design decisions of the designers and actual work practices.

Visual AgenTalk. Visual AgenTalk (http://www.cs.colorado.edu/~l3d/systems/agentsheets/) is an environment for developing domain-oriented end-user programming languages that are tailorable to a particular domain, promote program comprehensibility, and provide end users with control over powerful, multimodal interaction capabilities. It provides mechanisms for the creation of commands so that domain-specific languages can be developed. Conditions, actions, and rules are all graphical objects, and end users can try out their programs by dragging and dropping them onto agents in a worksheet. The ability to test programs within the context of a particular agent increases the end user's ability to comprehend Visual AgenTalk programs. In addition, Visual AgenTalk commands are powerful, providing users with access to a rich set of communication modalities including mouse clicks, timers, sound input, and animation. Visual AgenTalk supports the SER model by enabling end users to modify and program the behavior of active agents inside a simulation environment. New agent types can be created, modified, and shared, promoting evolutionary growth at the hands of end users.

Behavior Exchange. The Behavior Exchange [Repenning & Ambach, 1997] involves communities of practice in the evolution of systems by supporting two-way interaction over the World-Wide Web. It provides a forum in which developers can make their systems publicly available with the help of the World-Wide Web, so that everybody not only can look at the systems available, but also can use them to create new artifacts, and make these new artifacts available as well. Communication is encouraged and triggered by interactive exhibits. The Behavior Exchange is a contribution to change the World-Wide Web from an information distribution environment to a collaboration environment. It supports the SER model as a seed-distribution and growth-capturing mechanism. It enables developers to make their systems available to potential users. Those users create artifacts, make them publicly available, evolve the available systems, and make the added/modified functionality available to the distributed stakeholder community.

5. Assessment Of DODEs

5.1 Related Work

The Centrality of Evolution in the Design of Complex Systems. Evolution of complex systems is a ubiquitous phenomenon. Many researchers have addressed the evolutionary character of successful complex systems and of scientific endeavor. New paradigms are emerging in many fields, leading to a replacement of earlier ideas that were based on mechanistic determinism toward new models of change, indeterminance, and evolution. Knowledge in all fields should be open to critical examination, and the advance of knowledge consists of the modification of earlier knowledge. Big-step reductionism [Dawkins, 1987] cannot work as an explanation of mechanism; we can't explain a complex thing as originating in a single step, but complex things *evolve*. This is true in the physical domain, where, for example, artificial cities such as Brasilia are missing essential ingredients from natural cities such as London or Paris. "Natural" cities gain essential ingredients through their evolution — designers of "artificial" cities are unable to anticipate and create these ingredients. It is equally true for software systems for the reasons argued in this paper. A challenge for the future is to make software designers aware of essential concepts that originated and were explored in evolution, such as ontogeny, phylogeny, and punctuated equilibrium. For example, the evolution of individual artifacts (as illustrated in the upper level of Figure 3) can be described as the ontogenic development of an individual artifact, whereas the middle band of Figure 3 shows the phylogenic development of a generic species, namely a family of software systems as represented by a DODE. The development of operating systems (characterized with the SER model) illustrates the notion of a punctuated equilibrium, namely that they often go through lengthy periods where they remain unchanged, followed by brief periods of rapid change (e.g., when a new major version is released). Even though we are convinced that models from biology may be more relevant to future software systems than models from mathematics, we also have to be cautious: to follow an evolutionary approach in software design *successfully* does not imply that concepts from biological evolution should be

mimicked literally, but rather they need to be reinterpreted in the domain of software design [Basalla, 1988] — an attempt which we undertake with the SER model.

Evolution is essential in software systems because the assumption of complete requirements at any point in time is detrimental to the development of useful and usable software systems. Successful software gets changed because it offers the possibility to evolve. Lee [Lee, 1992] describes many convincing examples (including the failure of the Aegis system in the Persian Gulf) that design approaches based on the assumption of complete and correct requirements do not correspond to the realities of this world. Curtis and colleagues [Curtis et al., 1988] identified in a large-scale empirical investigation that fluctuating and conflicting requirements are critical bottlenecks in software production and quality and that a large percent of the lifecycle costs of a complex system is absorbed and needs to be dedicated to enhancements. Much of this cost is due to the fact that a considerable amount of essential information (such as design rationale [Moran & Carroll, 1996]) is lost during development and must be reconstructed by the designers who maintain and evolve the system. Development, maintenance and enhancement have to merge into cycles of an evolutionary process, making capturing of design rationale a necessity rather than a luxury.

Understanding Pitfalls Associated with Evolutionary Design. The Oregon Experiment [Alexander et al., 1975] (a housing experiment at the University of Oregon instantiating the concept of end user-driven evolution) serves as an interesting case study that end user-driven evolution is no guarantee for success. The analysis of its unsustainability indicated two major reasons: (1) there was a lack of continuity over time, and (2) professional developers and users did not collaborate, so there was a lack of synergy. These findings led us in part to postulate the need for a reseeding phase (making evolutionary development more *predictable*), in which developers and users engage in intense collaborations. With design rationale captured, communication enhanced, and end user modifiability supported, developers have a rich source of information to evolve the system in the way users really need it.

Comparison with other Domain-Oriented Approaches. DODEs transcend many existing design methodologies: (1) they reconceptualize domain modeling [Prieto-Diaz & Arango, 1991] with domain construction, and knowledge acquisition with knowledge construction [Fischer et al., 1996b]; (2) they transcend current object-oriented approaches by providing new support mechanisms for extensibility, reusability and evolvability [Fischer et al., 1995]; (3) they have contributed to paradigm shifts from the "specify-build-then-maintain" cycle to one of continuous evolution, and from developing a single application to designing domain-oriented application families; and (4) they force us to abandon closed systems in favor of open, living systems, which evolve over time and need to be sustained on an ongoing basis.

Our work in DODEs will be more tightly integrated with the theory of domain knowledge as developed by Sutcliffe and his colleagues [Maiden & Sutcliffe, 1992; Sutcliffe, 1997], who have explored the relevance of generic components of domains that can be reused in a variety of different domain. For example, many of the DODEs we have built over the last ten years [Fischer, 1994] capture knowledge about spatial relationships that could be shared among all of them. The two approaches complement each other: DODEs can give (1) more detailed advice for learning for specific problems and (2) generic components support for more widespread reuse of requirements and design knowledge.

WWW Support for the SER Model. Advances in networking have created new opportunities for communication among members of widely distributed communities. Whereas in the past, complex systems have been created by a very large effort by a small group of people, environments now coming into existence promise that complex systems (by building on a seed) can be evolved by small contributions of a (very) large community of practice. We can envision a world that encourages people to become active producers of knowledge. Examples of first steps toward creating an economy of knowledge based on community participation is beginning to take form in the software design community. Due to the contributions of developers around the world, the Java programming community has used community repositories of knowledge to produce incredible technical advances in an extremely short period of time. Gamelan (http://www.gamelan.com) was one of the first and largest of the community repositories of knowledge. The primary users of Gamelan are Java developers looking for information about what other people are doing with Java. Based on the large number of developers who contribute to the Gamelan repository and the number of people who

search for information in Gamelan, it appears that the Java community has taken a great deal of interest in using community repositories to locate information.

The Educational Object Economy (http://trp.research.apple.com/EdEconomy) provides an example that is more domain-oriented than Gamelan by providing educational support that is lacking in more global systems such as Gamelan. The Educational Object Economy is currently realized as a collection of Java objects (mostly completed applets) designed specifically for education. The target users of the Educational Object Economy are teachers wishing to use new interactive technology and developers interested in producing educational software.

5.2 Assessment of our Approach

Evolution in DODEs. Our experience with DODEs clearly indicates that DODEs themselves *and* artifacts created with them need to evolve. The ability of a DODE to co-evolve 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 to be evolved.

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. Systems must be open enough to allow "emergent behavior."

- *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. Systems must be underdesigned to support emergent new ideas. Software architectures need to be developed for software that is designed to evolve.

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. 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. Mechanisms for end-user modification and programming are, therefore, a cornerstone of evolvable systems. DODEs make end-user modifications feasible because they support interaction at the domain level. We do not assume that all domain designers will be willing or interested in making system changes, but within local communities of practice 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 [Schön, 1983] by signaling breakdowns and by making task-relevant argumentation and catalog examples available.

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 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. 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 a knowledge base. At the same time, important knowledge is produced during daily design activities that should be captured. 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.

Current Limitation and Research Issues for DODEs. There are numerous reasons that the DODE approach will not be readily accepted (for some arguments see the commentaries in [Fischer, 1994]).Software designers often feel that they need to create "universal solutions" that make everyone happy. They have difficulty sacrificing generality for increased domain-specific support. DODEs replace the clean and controllable waterfall model with a much more interactive situation in which control is distributed among all stakeholders. Several of our DODEs have been used in real work environments [Girgensohn et al., 1995; Sumner, 1995] and their impact, strengths and weaknesses have been carefully analyzed [Sumner et al., 1997].

Creating seeds for a variety of different domains will require substantial resources and the willingness of people from different disciplines to collaborate. In order to make the creation of a large number of domain-specific environments economically feasible, powerful substrates are needed (such as Agentsheets, Expectation Agents, Visual Agentalk, Behavior Exchange, and Gimme, mentioned earlier). The necessity of investing in long-term benefits must be taken seriously. Designers who do the work (e.g., provide design rationale) without directly benefiting from their efforts must be rewarded [Grudin, 1991]. Evolving seeds over time will require more involvement of users, and a willingness to acquire additional and different qualifications, as well as different organizational commitments [Nardi, 1993].

As 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, such as critics, and support mechanisms for learning on demand are needed to address these problems.

6. Conclusions

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, 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, creating artifact memories). We are further encouraged by the excitement and widespread interest in DODEs and the numerous prototypes being constructed, used, and evaluated in the last few years.

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. Jim Ambach and Ernesto Arias collaborated with the author to establish parts of the conceptual framework and to develop the scenario. The author also would like to thank the colleagues who provided important feedback and commentaries to his earlier article in *Automated Software Engineering*, 1(2). These commentaries served as an important input for the future development of our work. The work was further stimulated by many discussions with Alistair Sutcliffe over the last two years.

The research was supported by (1) the National Science Foundation, Grants REC-9631396 and IRI-9711951; (2) NYNEX Science and Technology Center, White Plains; (3) Software Research Associates, Tokyo, Japan; (4) PFU, Tokyo, Japan; and (5) Daimler-Benz Research, Ulm, Germany.

References

Alexander, C., Silverstein, M., Angel, S., Ishikawa, S., & Abrams, D. (1975) *The Oregon Experiment*, Oxford University Press, New York, NY.

Basalla, G. (1988) *The Evolution of Technology*, Cambridge University Press, New York.

Curtis, B., Krasner, H., & Iscoe, N. (1988) "A Field Study of the Software Design Process for Large Systems," *Communications of the ACM*, 31(11), pp. 1268-1287.

Dawkins, R. (1987) *The Blind Watchmaker*, W.W. Norton and Company, New York - London.

Eisenberg, M., Fischer, G. (1994) "Programmable Design Environments: Integrating End-User Programming with Domain-Oriented Assistance," *Human Factors in Computing Systems, CHI'94*, pp. 431-437.

Fischer, G. (1994) "Domain-Oriented Design Environments," *Automated Software Engineering*, 1(2), pp. 177-203.

Fischer, G., Grudin, J., Lemke, A. C., McCall, R., Ostwald, J., Reeves, B. N., & Shipman, F. (1992) "Supporting Indirect, Collaborative Design with Integrated Knowledge-Based Design Environments," *Human Computer Interaction, Special Issue on Computer Supported Cooperative Work*, 7(3), pp. 281-314.

Fischer, G., Lemke, A. C., McCall, R., & Morch, A. (1996a) "Making Argumentation Serve Design," In T. Moran & J. Carrol (eds.), *Design Rationale: Concepts, Techniques, and Use*, Lawrence Erlbaum and Associates, Mahwah, NJ, pp. 267-293.

Fischer, G., Lindstaedt, S., Ostwald, J., Schneider, K., & Smith, J. (1996b) "Informing System Design Through Organizational Learning," *International Conference on Learning Sciences (ICLS'96)*, pp. 52-59.

Fischer, G., McCall, R., Ostwald, J., Reeves, B., & Shipman, F. (1994) "Seeding, Evolutionary Growth and Reseeding: Supporting Incremental Development of Design Environments," *Human Factors in Computing Systems (CHI'94)*, pp. 292-298.

Fischer, G., Redmiles, D., Williams, L., Puhr, G., Aoki, A., & Nakakoji, K. (1995) "Beyond Object-Oriented Development: Where Current Object-Oriented Approaches Fall Short," *Human-Computer Interaction*, 10(1), pp. 79-119.

Gamma, E., Johnson, R., Helm, R., & Vlissides, J. (1994) *Design Patterns - Elements of Reusable Object-Oriented Systems*, Addison-Wesley,

Garlan, D., Shaw, M. (1994) *An Introduction to Software Architecture* , (Technical Report No. CMU/SEI-94-TR-21). Software Engineering Institute.

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

Girgensohn, A., Zimmerman, B., Lee, A., Burns, B., & Atwood, M. (1995) "Dyanmic Forms: An Enhanced Interaction Abstraction Based on Forms," *Fifth International Conference on Human-Computer Interaction (Interact '95)*, pp. 362-367.

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

Grudin, J. (1991) "Interactive Systems: Bridging the Gaps Between Developers and Users," *Computer*, 24(4), pp. 59-69.

Henderson, A., Kyng, M. (1991) "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, Inc., Hillsdale, NJ, pp. 219-240.

Lave, J. (1991) "Situated Learning in Communities of Practice," In L. Resnick, J. Levine, & S. Teasley (eds.), *Perspectives on Socially Shared Cognition*, American Psychological Association, Washington, DC, pp. 63-82.

Lee, L. (1992) *The Day The Phones Stopped*, Donald I. Fine, Inc., New York.

Maiden, N. A. M., Sutcliffe, A. G. (1992) "Exploiting Reusable Specifications through Analogy," *Communications of the ACM*, 35(4), pp. 55-64.

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

Nardi, B. A. (1993) *A Small Matter of Programming*, The MIT Press, Cambridge, MA.

Polanyi, M. (1966) *The Tacit Dimension*, Doubleday, Garden City, NY.

Prieto-Diaz, R., Arango, G. (1991) *Domain Analysis and Software Systems Modeling*, IEEE Computer Society Press, Los Alamitos, CA.

Repenning, A., Ambach, J. (1997) "The Agentsheets Behavior Exchange: Supporting Social Behavior Processing," *Computer-Human Interaction (CHI '97)*, pp. 26-27 (Extended Abstracts).

Rittel, H. (1984) "Second-Generation Design Methods," In N. Cross (eds.), *Developments in Design Methodology*, John Wiley & Sons, New York, pp. 317-327.

Salasin, J., Shrobe, H. (1995) "Evolutionary Design of Complex Software (EDCS)," *Software Engineering Notes*, 20(5), pp. 18-22.

Schön, D. A. (1983) *The Reflective Practitioner: How Professionals Think in Action*, Basic Books, New York.

Shipman, F., McCall, R. (1994) "Supporting Knowledge-Base Evolution with Incremental Formalization," In *Human Factors in Computing Systems, INTERCHI'94 Conference Proceedings*, pp. 285-291.

Simon, H. A. (1981) *The Sciences of the Artificial*, The MIT Press, Cambridge, MA.

Sumner, T. (1995) *Designers and Their Tools: Computer Support for Domain Construction*, Ph.D., Department of Computer Science, University of Colorado at Boulder.

Sumner, T., Bonnardel, N., & Kallak, B. H. (1997) "The Cognitive Ergonomics of Knowledge-Based Design Support Systems," *Human Factors in Computing Systems, CHI'97*, pp. 83-90.

Sutcliffe, A. (1997) "Using Domain Knowledge in Interactive System Design," In (submitted for publication):

Terveen, L. G., Selfridge, P. G., & Long, D. M. (1995) "Living Design Memory: Framework, Implementation, Lessons Learned," *Human-Computer Interaction*, 10, pp. 1-37.