University of Colorado at Boulder

Center for LifeLong Learning and Design (L3D)

Department of Computer Science

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

Evolutionary Design of Open, Complex Systems

Gerhard Fischer

Center for LifeLong Learning and Design (L3D)
Department of Computer Science and Institute of Cognitive Science
University of Colorado
Campus Box 430
Boulder, CO 80309-0430
tel: (303) 492-1502 fax: (303) 492-2844
gerhard@cs.colorado.edu

International Symposium on Science of Computing and Technology of Software, Software Engineer's Association, December 1996, pp. 13-25

Evolutionary Design of Open, Complex Systems

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, Colorado 80309
gerhard@cs.colorado.edu

Abstract. The major philosophical foundation for software technology of the future is to think about, engage in, develop architecture, processes and support environments for the evolutionary design of open, complex 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^3D) 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 be completely specified before system development occurs. Our research focuses on the following claims: software systems (1) must evolve because they cannot be completely designed prior to use, (2) must evolve to some extent at the hands of the users, and (3) 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 can 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.

software process model and architecture specifically for systems that can 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.

Introduction

Software engineering research has been historically concerned 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. 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, etc., rather than to a specific step in a software engineering life-cycle model (located between requirements and implementation). Domain-oriented design environments (DODEs) [Fischer 1994a] 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 [Winograd 1966]. 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 specific software architectures (DSSA) and evolutionary design of complex systems (EDCS) [Salasin, Shrobe 1995].

Evolutionary Design

The Need for Change and Evolution. There are numerous fundamental reasons why systems cannot be done "right." Software systems model parts of our world [Ehn 1988]. 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 [CSTB 1990; Salasin, Shrobe 1995]. System maintenance and enhancement need to become "first class design activities."

Domain-oriented design environments (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.

construction while allowing designers to access relevant knowledge at each stage within the software development process.

A Scenario From The Domain Of Computer Network Design. The following scenario illustrates how a DODE affects the exemplary domain of computer network design. In the scenario [Fischer, Ambach, Arias 1995] we emphasize 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 (by building on several earlier version of DODEs for computer network design [Fischer et al. 1992; Reeves 1993; Shipman 1993; Sullivan 1994]):

- 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.

This 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.

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)).

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 so that D_1 can modify it. D_1 changes the design to reflect the specific needs of the Publications Group.

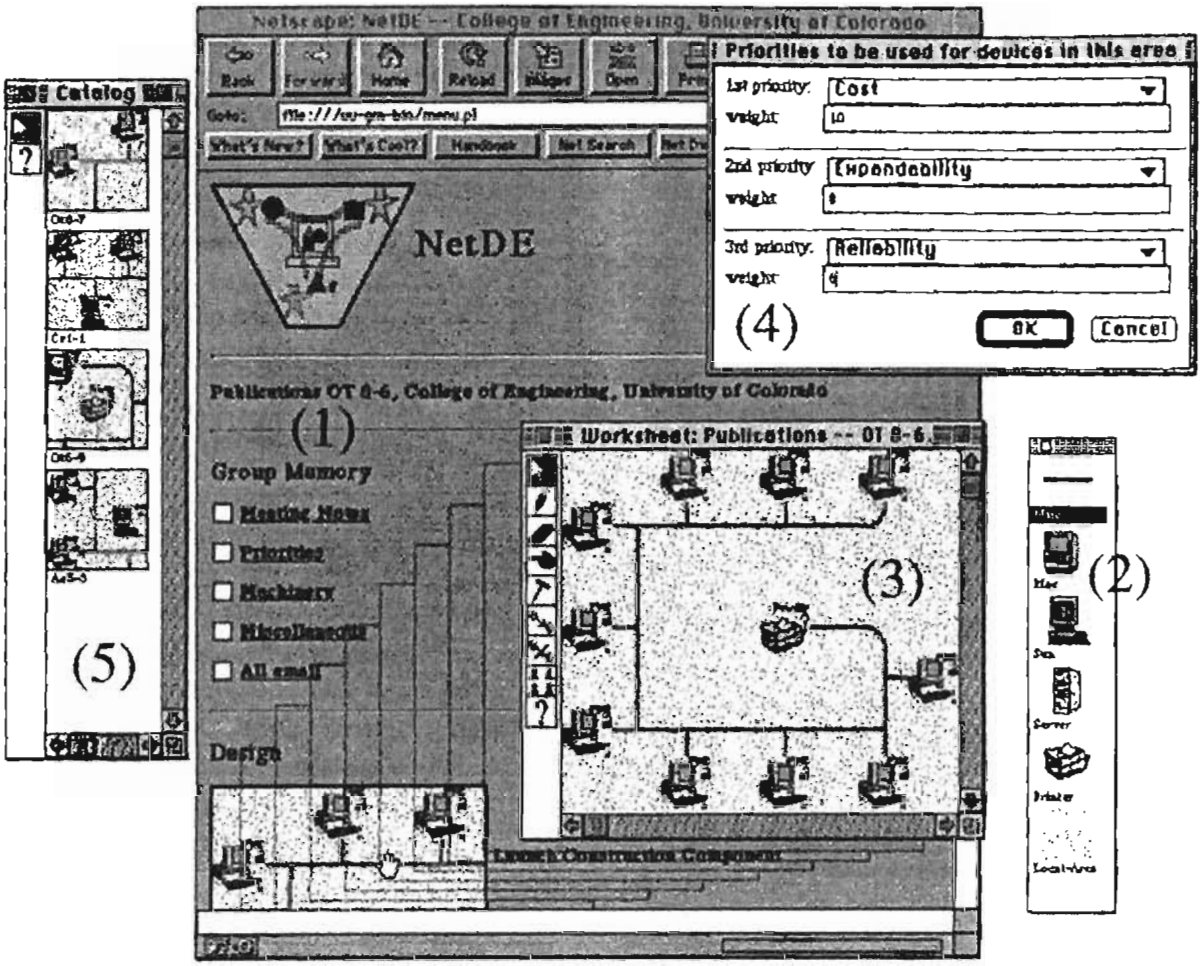

Figure 1: NetDE in Use

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 mechanism (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 et al. 1991a], 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. 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, D2 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 is no longer as important, and that 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, D2 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. D2 enters this information, and the new icon appears in the palette. D2 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. D2 knows that Macs can understand TCP/IP protocol, so she changes the network's protocol to TCP/IP.

machines (Macs and the Indy) are using different network protocols. D2 knows that Macs can understand TCP/IP protocol, so she changes the network's protocol to TCP/IP. After invoking NetDE's critiquing mechanism, D2 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)), D2 learns that although file sharing is possible in TCP/IP with the Macs, it is not as easy as using EtherTalk. D2 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. D2 learns that there is software the Indy can run to translate protocols, and she adds an annotation to the Indy object to reflect this. A critiquing component [Fischer et al. 1991a] is important in linking design rationale and argumentation [Fischer et al. 1991b; Schön 1983] to the designed artifact as well as for pointing out potential breakdowns to the designer.

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

As illustrated in the previous 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 [Fischer 1994b]. Because these breakdowns 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 of supporting end-user modifiability are required as an intrinsic part of a DODE.

We distinguish three intertwined levels of artifacts; their interaction forms the essence of our seeding, evolutionary growth, reseeding model (see Figure 3):

- On the *conceptual framework level*, 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*.

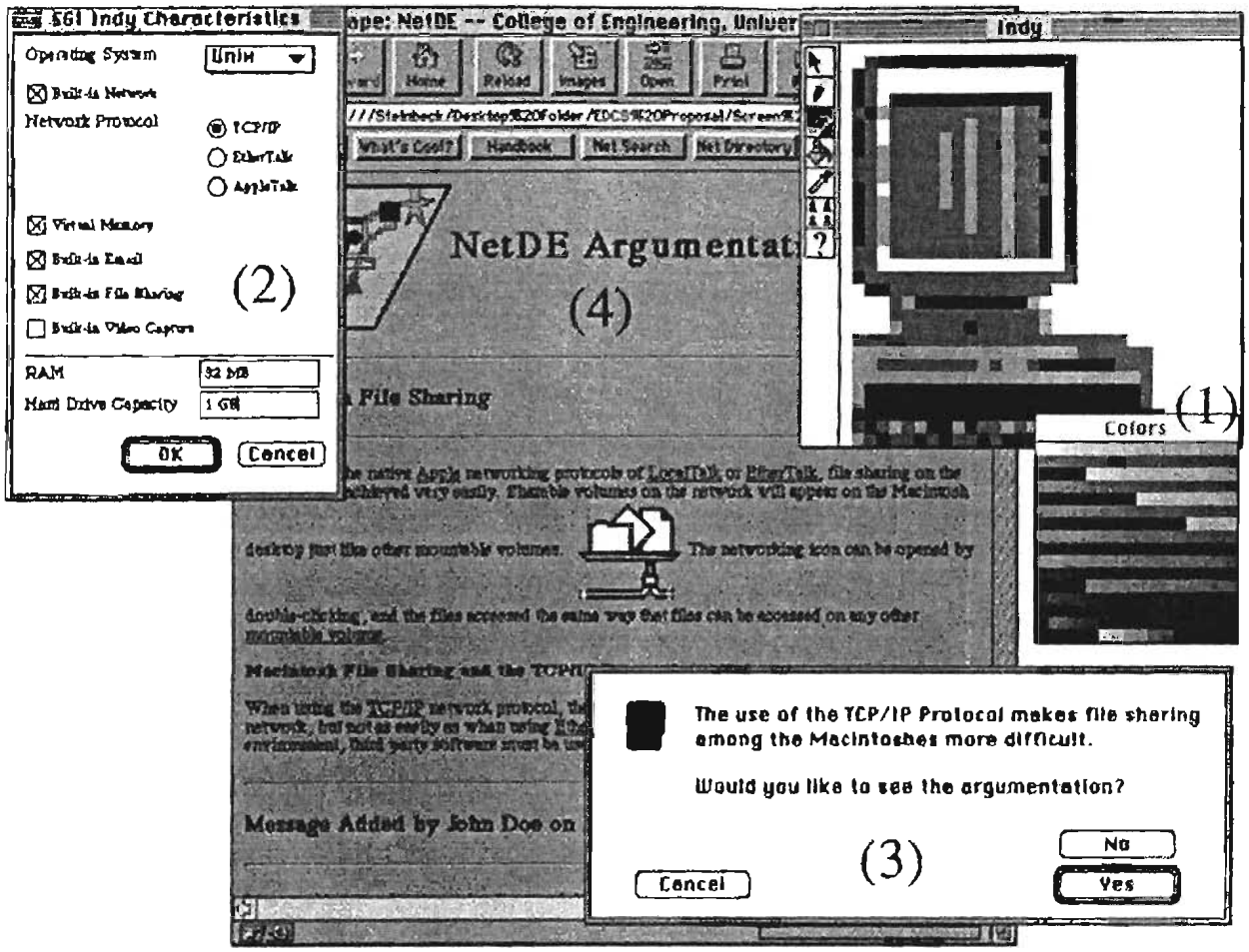

Figure 2: The Network Evolves

- Individual *artifacts* in the domain are developed by exploiting the information contained in the DODE.

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 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* (professional software designer) 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 n collaboration with clients. 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

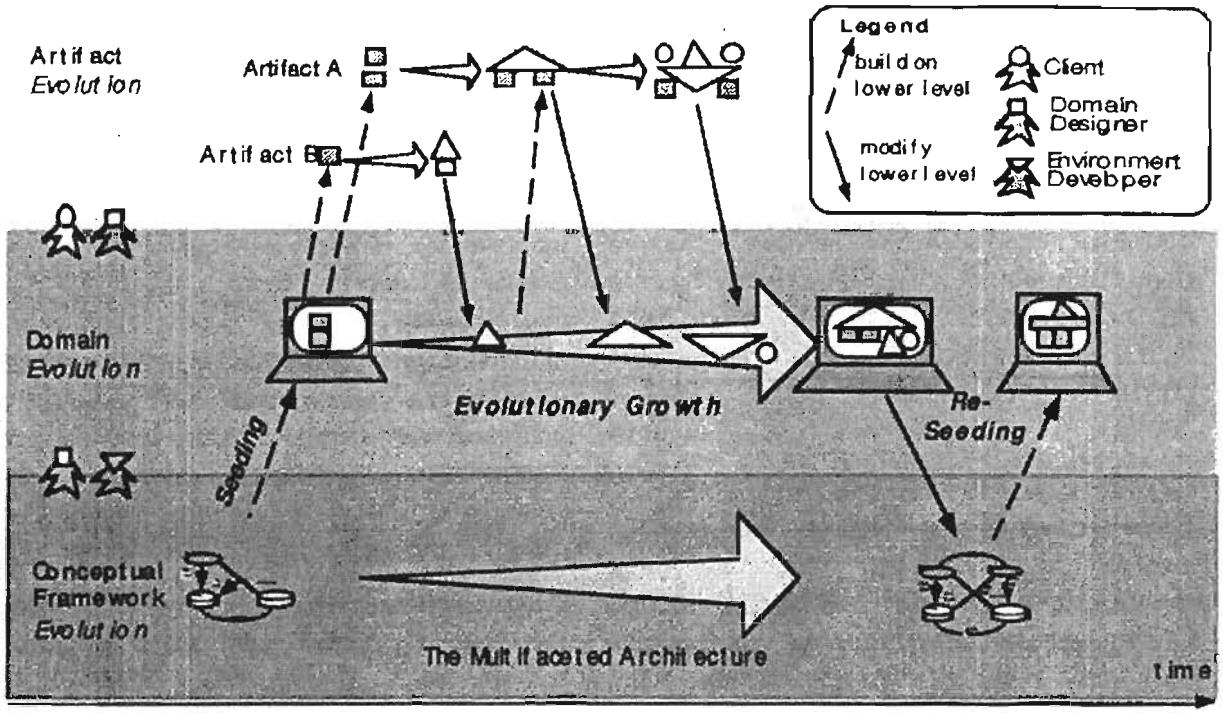

Figure 3: The SER Model: A process model for the development and evolution of DODEs

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 react to.

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) [Ostwald 1996]. 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 [Fischer et al. 1992]. 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; Fischer, Girgensohn 1990], and end-user programming [Ambach, Perrone, Repenning 1995; 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 [Grudin 1991; Grudin 1994; 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. 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].

Collaboration and Communication Between Stakeholders as Facilitated by the SER Model. The SER model emphasizes several collaboration and communication processes between stakeholders (e.g., during the seeding phase [Ostwald 1996], during evolutionary growth [Ambach, Perrone, Repenning 1995], and during reseeding [Shipman, McCall 1994]). 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, 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 [Fischer et al. 1992]. This requires that the relevant design information such as design rationale is associated and embedded in the artifact [Reeves 1993]. 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.

Illustrations of the SER Model with Examples from our Work. 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 [Fischer, Lemke 1988]. While construction kits allowed us to create artifacts quickly, the "back-talk" of the artifact themselves was insufficient [Norman 1986; 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 [Fischer et al. 1991b]. 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]

users led to the development of end-user modification components turning DODEs into programmable DODEs [Eisenberg, Fischer 1994]

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 artifacts, 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 [Reeves 1993; Shipman 1993] – and they have served us well to understand the processes associated with the evolution of an individual artifact. Analyzing the difficulties to evolve 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 [Fischer et al. 1992].

Assessment Of 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 source of information to evolve the system in the way users really need it to be evolved.

Our research provides theoretical and 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" [Brown, Duguid, Haviland 1994].

- *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.

Domain Orientation. 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. While the DODE approach itself is generic, each of its applications is a particular domain-oriented system. Our emphasis on DODEs demonstrated the importance of situated and contextualized communication and design rationale as the basis for *effective* evolutionary design.

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 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.

Conclusions:The Centrality Of Evolution In The Design Of Open, Complex Systems

Complex systems 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 [Arias 1995]. "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 which originated and were explored in evolution such as ontogeny, phylogeny and punctuated equilibrium [Dennett 1995]. 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 shows the phylogenic development of a generic species, namely a family of software systems as represented by a DODE. The development of operating systems (which as argued before can be 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). To follow an evolutionary approach in software design *successfully* does not only imply that concepts from biological evolution should be mimicked literally[Basalla 1988]. Rather they need to be reinterpreted in the domain of software design — an attempt which we undertake with the SER model.

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. 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, (4) Software Research Associates, and (5) PFU.

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.

[Arias 1995] E.G. Arias, Designing in a Design Community: Insights and Challenges, DIS '95 - Symposium on Designing Interactive Systems.: Processes, Practices, Methods and Techniques, pp. 259-263.

[Basalla 1988] G. Basalla, The Evolution of Technology, Cambridge University Press, New York.

[Brown, Duguid, Haviland 1994] J.S. Brown, P. Duguid, and S. Haviland,

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

[Norman 1986] D.A. Norman, Cognitive Engineering, in D.A. Norman and S.W. Draper (eds.), User Centered System Design, New Perspectives on Human-Computer Interaction, Lawrence Erlbaum Associates, Inc., Hillsdale, NJ, pp. 31-62.

[Ostwald 1996] J. Ostwald, Knowledge Construction in Software Development: The Evolving Artifact Approach, Ph.D. Thesis, Department of Computer Science, University of Colorado at Boulder.

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

[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.

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

[Salasin, Shrobe 1995] J. Salasin, and H. Shrobe, Evolutionary Design of Complex Software (EDCS), Software Engineering Notes, Vol. 20, No. 5, pp. 18-22.

[Schön 1983] D.A. Schön, 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.

[Shipman, McCall 1994] F. Shipman, and R. McCall, Supporting Knowledge-Base Evolution with Incremental Formalization, in Human Factors in Computing Systems, INTERCHI'94 Conference Proceedings, pp. 285-291.

[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.

[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 1966] T. Winograd, Bringing Design to Software, ACM Press and Addison-Wesley, New York.