Meta-Design—Design for Designers

Gerhard Fischer
Center for LifeLong Learning and Design (L^{3}D)
Department of Computer Science and
Institute of Cognitive Science
University of Colorado
Boulder, CO 80309-0430 USA
gerhard@cs.colorado.edu

Eric Scharff
Center for LifeLong Learning and Design (L^{3}D)
Department of Computer Science and
Institute of Cognitive Science
University of Colorado
Boulder, CO 80309-0430 USA
scharffe@cs.colorado.edu

ABSTRACT
One fundamental challenge for the design of the interactive systems of the future is to invent and design environments and cultures in which humans can express themselves and engage in personally meaningful activities. Unfortunately, a large number of new media are designed from a perspective of viewing and treating humans primarily as consumers. The possibility for humans to be and act as designers (in cases in which they desire to do so) should be accessible not only to a small group of "high-tech scribes," but rather to all interested individuals and groups. Meta-design characterizes activities, processes, and objectives to create new media and environments that allow users to act as designers and be creative.

In this paper we discuss problems addressed by our research on meta-design, provide a conceptual framework for meta-design, and illustrate our developments in the context of a particular system, the Envisionment and Discovery Collaboratory.

Keywords
Consumer and designer mindsets; designing "out of the box," domain-oriented design environments; open evolvable systems; Open Source; seeding, evolutionary growth, reseeding model; impact of new media on design; underdesigned systems.

1. INTRODUCTION
Cultures are substantially defined by their media and their tools for thinking, working, learning, and collaborating. A large number of new media are designed to see humans only as consumers. Television is the most obvious medium that promotes this mindset and behavior [33] and contributes to the degeneration of humans into "couch potatoes"—individuals for whom passive consumption dominates, and activity, both physical and intellectual, is limited [13].

Unfortunately, a consumer mindset does not remain limited to television; in many cases it is a model dominating our culture. In our educational institutions learners are often treated as consumers, which creates a mindset of consumerism for the rest of their lives [15, 23]. Citizens often feel left out of decisions by policy makers, denying them opportunities to take an active role. Computational media have the unique potential to let people be designers or to assist them to incrementally become designers. Unfortunately, most current computational environments do not allow users to act as contributors and designers [12].

A Chinese proverb says: "If you give a fish to a human, you will feed him for a day—if you give someone a fishing rod, you will feed him for life." This saying can be extended by arguing that "if we can provide someone with the knowledge, the skill, and the tools for making a fishing rod, we can feed the whole community." *Meta-design* characterizes activities, processes, and objectives to create new media and environments that allow users to act as designers and be creative. This can be compared with the objective in art that focuses on the artist as the facilitator of the creative experience for users. In our work, we have explored a set of concepts and ideas for meta-design that are summarized in Figure 1.

This paper first presents problems at a technical and cultural level that are addressed by a meta-design approach. Then a conceptual framework for meta-design is outlined, including a brief characterization of meta-design tools and environments and a process model supporting meta-design. The Envisionment and Discovery Collaboratory (EDC) serves as an example for a system instantiating designing "out of the box" and illustrating the needs and possibilities for meta-design approaches. A brief assessment section based on the work so far articulates some of the major challenges for the future.

2. PROBLEMS

2.1 Shortcoming of Closed Systems
Closed systems typically create a sharp separation between the creation and use of the system. Providing functionality of collaborative human-computer systems that is fixed when the system is created has important implications on how it will be used. However, designing a system that can sufficiently anticipate all possible uses in advance (that is, when the system is created) is an impossible task [20, 42]. An important attribute of real software systems is that 40 to 60 percent of a system's cost over its lifetime is spent after the original system design is finished [7]. Sustaining the usefulness of

Concept	Implications
convivial tools	allow users to invest the world with their meaning and to use tools for the accomplishment of a purpose they have chosen [24]
domain-orientation	bring task to the forefront; provide time on task [11]
open, evolvable systems	put owners of problems in charge [18]
underdesigned systems	create seeds and constructs for design elaboration at use time [14]
collaborative work practices	support design communities and the emergence of power users [29]

Figure 1: Concepts of Meta-Design

software systems differs from the traditional concept of "maintenance" because beyond repairing defects and fixing bugs, most of the efforts (an estimated 75 percent of the overall maintenance efforts) involve enhancement activities. The needs for enhancements are noticed in most cases by skilled domain workers using these systems rather than by the system designers.

In the domain of urban planning, SimCity [27] provides an interesting example of a closed system that exemplifies some of the problems encountered when attempting to design with these systems. Although SimCity provides some superficial kinds of modifications (such as changing the appearance of buildings in the city), most aspects of the simulation environment have been determined by the designers. For example, the only way to reduce crime in a simulated city is to add more police stations. It is impossible to explore other solutions, such as increasing social services. The functionality of the system was fixed when the system was created, so exploring concepts that were never conceived by the system designers is difficult. Because of its closed nature, SimCity may make be a good tool for education or entertainment, but it is inadequate for actual city planning tasks. To support city planning, a system such as SimCity would need to be extended to represent the various kinds of situations encountered in urban planning design tasks.

Of course, it is possible to consider adding functionality to SimCity while maintaining its "closed" status, such as creating a new version. This does not address the fundamental problem with this approach, which is that activities and issues may arise that cannot be represented by the system. When closed systems lack the ability to evolve so that they can be modified to address unanticipated issues, they will inevitably be unable to cope with change and the possibly unlimited extensions that might arise in the design process.

Open, evolvable systems address the limitations often associated with closed systems. Open systems allow significant modifications when the need arises. The evolution that takes place through modifications is a "first class design activity." That is, it is important not only to allow people to design within a domain, but to be able to design modifications to the current realization of the domain when necessary. The need for open, evolvable systems was eloquently advocated by Nardi [29]: *"We have only scratched the surface of what would be possible if end users could freely program their own applications.... As has been shown time and again, no matter how much designers and programmers try to anticipate and provide for what users will need, the effort always falls short because it is impossible to know in advance what may be needed... End users should have the ability to create customizations, extensions, and applications....* {p. 3}"

In purely technical domains, *Open Source Software* [30] has emerged as an interesting example of successful open systems. Powerful tools such as the Linux operating system and the Apache Web server have become both useful and reliable in large part because of the evolutionary contributions of a large community of motivated developers. In Open Source Software, the source code for a computer system is available to a broad group of individuals, often anyone interested in obtaining it. Because developers can modify the source directly, they can make changes to the systems and, assuming they have the adequate motivation and technical competency, they can extend a system to fit their unique problems. Open Source Systems have also been encouraged by the development of the Internet and the Web, making it much easier for widely distributed communities to share their extensions. Open Source Software gives developers the power they need to extend systems. Creating open systems for non technical domains is an important challenge. Open systems have been successful when technically oriented people create technically oriented software, but it is also important to support users not motivated simply by technology in domains that do not involve the creation of technical artifacts.

2.2 The Consumer Mindset

Supporting people in taking an active role in the design processes that shape their lives implies that people wish to engage in this activity. Having a desire to become involved in the design activities that shape your life, ranging from participating in neighborhood planning groups to becoming active in knowledge sharing communities, requires having the motivation to take part in such activities. One of the critical preconditions for this motivation is a cultural mindset in which participation plays a major role. In short, taking an active role in design needs to be supported by the creation of a "designer mindset."

Although new technologies have the potential to move beyond the paradigm of consumption, many new technologies have adopted a similar role. Even the Web, which has been a successful medium in allowing anyone to distribute their own content, is based on a consumer mindset. Typically, someone will create a Web site and publish (or broadcast) this site to the world. People browsing the Web can receive this content, but they can rarely change the content provided. Personalized information has become more common recently, giving viewers more control over the information presented, but users still consume most information, and producing new information is limited. It is not surprising that a term such as "Web surfing" is similar to the television-oriented "channel surfing."

To create designer mindsets, one of the major roles for new media and new technologies is not to deliver predigested information to individuals, but to provide the opportunity and resources for social debate, discussion, and collaborative knowledge construction. In many design activities, learning cannot be restricted to finding knowledge that is "out there." For most design problems (ranging from urban design to

graphics design and software design) that we have studied over many years, the knowledge to understand, frame, and solve problems does not exist; rather it is constructed and evolved during the process of solving these problems, exploiting the power of the "symmetry of ignorance" [39] and "breakdowns" [17, 40]. From this perspective, *access* to existing information and knowledge (often seen as the major advance of new media) is a very limiting concept [2, 5, 32].

3. CONCEPTUAL FRAMEWORKS FOR META-DESIGN

Our belief is that providing the opportunity for people to become designers is both important and rewarding. People have incredible capabilities when they adopt a designer role, and, under the right circumstances, people want to be and act as designers. In this context, "design" is a broad notion that involves activities in which a person wishes to act as an active participant and contributor in personally meaningful activities. Of course, not everyone wants to be a designer, we are not designers all the time and in all contexts. However, when people have the need and desire to participate in a design process, we must provide contexts in which they can be designers. Illich [24] (sharing this premise) has articulated the need for convivial tools and systems, which he characterized as follows: "convivial tools allow users to invest the world with *their* meaning, to enrich the environment with the fruits of *their* vision and to use them for the accomplishment of a purpose *they have chosen*" (emphasis added). Convivial systems encourage users to be actively engaged in generating creative extensions to the artifacts given to them and have the potential to break down the strict counterproductive barriers between consumers and designers. What are the challenges involved in designing tools that support people in their design activities?

3.1 Design Time and Use Time

The need for meta-design is founded on these observations: design problems in the real world require *open systems* that users can modify and evolve. Because problems cannot be completely anticipated at *design time* (when the system is developed), users at *use time* will discover mismatches between their problems and the support a system provides.

One of the fundamental problems of system design is how to write software for millions of users (at design time), while making it work as if it were designed for each individual user (who is known only at use time) [16]. Figure 2 differentiates between two stages in the design and use of a system. At design time, developers create systems, and they have to make decisions for users for situational contexts and for tasks that these designers can only anticipate. For print media, a fixed context is decided at design time, whereas for computational media, the behavior of a system at use time can take advantage of contextual factors (such as the background knowledge of a user, the specific goals and objectives of a user, the work context, etc.) *known only at use time*. The fundamental difference is that computational media have interpretive power: they can analyze the artifacts created by users and the interaction patterns between users and system, and they can support users in their articulation of additional contextual factors.

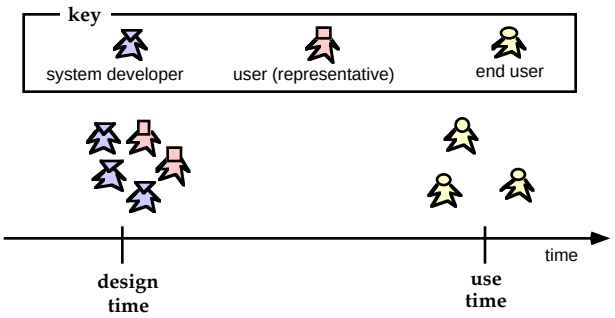

Figure 2: Design and Use time

3.2 The Spectrum of Meta-Design Tools

It is fair to wonder why current interactive programming environments, such as Lisp, Logo, and Smalltalk, are not ideal for supporting this meta-design. After all, these tools provide the ultimate level of openness and flexibility. For example, Squeak [25] is an Open Source [30] implementation of Smalltalk written entirely in itself. As a general-purpose programming language, it is capable of representing any problem that computers can be used to solve. As an open system, any user can change any aspect of the system if necessary, allowing for unlimited extension.

Although systems such as this are useful as computational substrates, by themselves they are insufficient for meta-design. The essential problem with these systems is that they provide the incorrect level of representation for most problems. Expressing a problem and designing a solution in these systems requires creating a mapping from the context of the problem to the core constructs provided by the programming language and its supporting library.

On the other side of the spectrum, domain-specific tools such as SimCity provide extensive support for certain problem contexts. SimCity 3000, the latest in the SimCity series, has mechanisms to construct and manipulate many parameters of a city. Users can manipulate zoning (determining the purpose for which a specific plot of land should be used), finance, transportation, electricity, water, even sanitation (sewage and garbage disposal). However, as we discussed previously, the ability to extend these environments is often limited. SimCity provides some mechanisms for change with the SimCity Urban Renewal Kit (SCURK, now known as the Building Architect Tool, BAT). SCURK and BAT allow users to change the appearance of buildings, but these changes are limited to on-screen appearance. The behavior and semantics of the buildings remains unchanged. If the system provides all the necessary capabilities to design an artifact, then a closed domain-specific tool may provide adequate support for certain design activities but not others. Even minor incremental changes may not be possible in these systems.

Many systems fall between these two extremes. Domain-oriented design environments (DODEs; described in detail below) are tools that allow users to construct artifacts within the confines of a specific domain. Users can construct new designs, modify existing ones, and extend the underlying domain framework. Microsoft Word provides a great deal of support for creating and manipulating documents. It also provides mechanisms for extending the existing functionality of the system. *AgentSheets* [36, 37] is an end-user simulation programming tool that combines the concepts of autonomous computational entities (agents) and the metaphor of the grid

(spreadsheets). Users create and program agents that interact within a grid-based environment. Thus, AgentSheets provides a high degree of programmability while attempting to remain domain-neutral, being capable of representing tasks that can be modeled using a discrete grid.

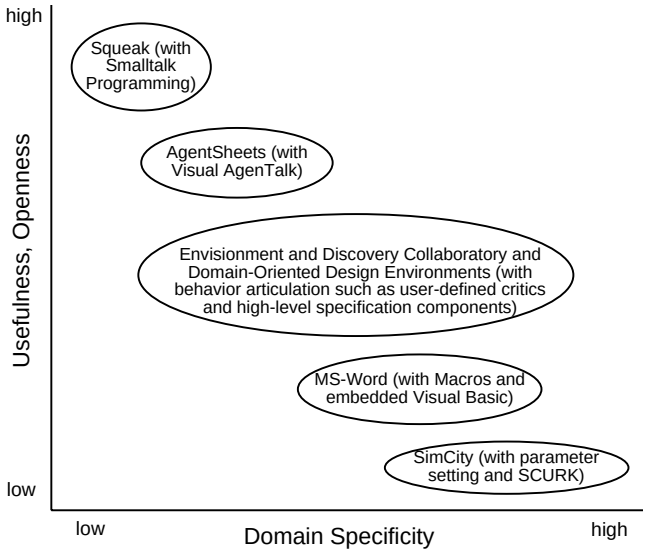

Figure 3: Spectrum of Design Tools

Figure 3 briefly summarizes the spectrum of tools that can be used for meta-design (similar classifications can be found in [26], [29], and [37]). Increasing domain specificity allows environments to provide greater support in solving problems within a given context, but limits the scope of the context that can be explored. Increasing the facilities for extension allow users to modify an environment to fit a new context, but extending the environment involves an ever-increasing gap between the context in which the design is taking place and the ability of a system to represent that context. It is apparent from analyzing the spectrum of design tools that no single point on the spectrum is ideal for all contexts. Therefore, being able to move smoothly along the spectrum is an important aspect of supporting meta-design.

3.3 The Seeding, Evolutionary Growth, Reseeding (SER) Model

In the context in which designers can create and extend a system over time, it is valuable to understand how a system is transformed over time. We have developed the *seeding, evolutionary growth, reseeding (SER) model* to help explain how meta-design systems can be understood [14]. The SER model helps explains how open systems can develop over time.

In the SER model, system developers and users (see Figure 2) develop an initial *seed*, which is a first attempt at creating a tool to support work within a specific domain. An important aspect of this seed is that it is designed to be extended. Because it is impossible to capture any design activity completely, the seed must be able to grow through use. In this way, a seed can be initially *underdesigned* [4], meaning that at design time the environment designers do not create final solutions but rather design spaces that can be changed and modified by domain designers at use time. As the seed is used for real design activity, it goes through a period of *evolutionary growth* in which the designers make incremental modifications to the system over time. Because evolution happens at the hands of the users, there must be mechanisms that allow users to make necessary changes. This implies that a system is open, so that modifications are possible, and that there are extension facilities that make extension capability available to designers without making a significant leap from domain work.

Eventually, it will become necessary to do a significant reconceptualization of the system, or *reseeding*. There are many reasons why reseeding is necessary, including the possibility that some incremental changes may point out fundamental limitations in the seed, managing and combining many incremental changes may be difficult, and some incremental changes may make future changes more difficult. Reseeding is a complex process by which a group of people must take stake in the current system, synthesize the current state of the system, and reconceptualize the system. The result of the reseeding process is a new system that can serve as the basis for future evolution. The cycle of evolution and reseeding continues as there are people actively using the system to solve problems.

The SER model can be used to understand how meta-design activity can take place. Open Source Software development follow the patterns detailed in the SER model. When designing and understanding Open systems, an understanding of the lifecycle of a system can be informative in determining what is important, and is thus a rubric for meta-design. A seed must provide the facility for people to design artifacts within a domain, but must be able to evolve over time. The system evolves as new situations reveal themselves and the system is extended or refined to handle these new situations. Incremental evolution cannot happen forever and it will be necessary to reconceptualize a system to create some order to the emergent changes and to facilitate future extension to the system.

Decentralizing evolution [38] is an important goal, but it also may present serious difficulties. When people make incremental modifications in their own contexts, these changes may not work with the changes made by others in different contexts, making sharing difficult. Keeping track of a large community so that different people don't come up with similar changes that are incompatible with each other is difficult. In Open Source Software, there is typically a centralized authoritative version of a system. Contributions to this core version are managed by a gate-keeper, either an individual or group. This centralized integration helps reduce the likelihood of incompatible changes. Determining whether an incremental change should become a "core" part of a system must be decided by some group of people, raising an issue about who controls a system. Open Source projects tend to have one project leader (or a small group of leaders) who have the ultimate say over what is a "central" part of a system. One might assume that this would lead to a totalitarian control over evolution, but this is rarely observed. Often a core group of motivated individuals will discuss changes. If they are pleased with the extensions (and adopt the extension themselves) there is a good chance that they will become a core part of the system. Sometimes, new situations will reveal situations in which incremental modifications are impossible, requiring a paradigm shift. In all of these cases, the community must make a conscious effort to reseed a system to address these major concerns. Determining who should be involved in

this reseeding and how it should take place is an important and sensitive issue.

A closer analysis of the SER framework also points out some challenges that must be faced in understanding meta-design activity. In a closed system, the evolution and reseeding of the system is not a core activity in the system. Effectively supporting both of these activities is difficult. If we want to design things, extending the substrate in order to design what we want is not likely to be the highest priority for designers. If the extension process is sufficiently off-task, time-consuming, and difficult, designers are unlikely to be motivated to make modifications. Reseeding may prove to be an expensive and time consuming process in which individuals or groups spend effort in redesigning or reconceptualizing the system. Again, if this task is sufficiently time-consuming or off-task, people will be unwilling to participate, but without their participation a system may reach a point at which it is no longer sustainable. In Open Source projects, evolution and reseeding occur naturally because there is large overlap between the community of developers and users. Tools change because the members of the community both extend and use the tools in their daily work. It is harder to see examples of successful systems where all the users are not developers. In meta-design, it is valuable for users to engage in design activities, but not all these activities necessarily require software development. Supporting meta-design in communities where the design of software itself is not the core activity is an important challenge.

3.4 Macros and Embedded Languages

Users are most likely to become designers when they are creating personally and socially meaningful artifacts. Meta-design activity that is embedded in the context of creating artifacts has the potential to capture changes as they are encountered in actual work. The meta-design capabilities of Microsoft Word provide an excellent example of this. We have written macros to help fix common typing errors (such as transposing two characters) and insert email into documents by erasing the superfluous line breaks inserted by most email programs. These extensions represent the nature of incremental modifications that meta-design must support. Both were motivated by authentic, frequently occurring problems that Word does not handle well. The extensions themselves are created partially through direct manipulation (through Word's macro recording system) and partially by learning Visual Basic for Applications, the extension language embedded in Word.

These examples also point out the challenges that systems supporting meta-design must face. Our empirical observations and studies have demonstrated that meta-design requires more than just technical facilities. Users will not realize a need to extend a system until an activity in which they engage illustrates a limitation of the system, and one that the user is sufficiently motivated to overcome by abandoning the current task to make the necessary modification. Extending open systems will not take place within the first few days or weeks of using them, but requires the long-term use of systems by owners of problems. We do not expect all users to become end-user programmers or to be interested in making radical changes to systems. Their contributions will depend on the perceived benefit of contributing, which involves the effort needed to make changes and the utility received for effecting changes. Few users take advantage of the end-user modifiability components provided by environments such as Microsoft Word, and even fewer users engage in exchanging their extensions with others. It is very easy to make a macro that you cannot share with others because it (invisibly) depends on some local situation that isn't true on others' systems. Tracking how a modification has changed over time is extremely challenging. Other communities (such as the Open Source community [34] and Web-based community of practice [10]) are better success examples to be analyzed for meta-design.

3.5 Domain-Oriented Design Environments

The most promising way to provide opportunities for a "designer mindset" [13] is to allow learners and workers to engage in design activities by creating environments supporting them in making external artifacts that they can reflect upon and share with others. Over the last ten years we have built a large number of different domain-oriented design environments [11] that support collaborative activity within a specific domain. DODEs support mechanisms such as constructing artifacts relevant to a specific domain, critiquing these constructions, accessing catalogs of existing designs, linking to contextualized argumentation, and extending with end-user modifiability capabilities. DODEs can be used not only to instruct and assist novice designers, but also to support designers at all levels of expertise.

DODEs support meta-design by allowing users to work on tasks within a specific domain, rather than working on pre-defined or fixed tasks. Critics [17] help uncover breakdowns, by activating relevant information and providing a context for extension. They also demonstrate challenges that make meta-design (and the SER model) feasible and workable. DODEs need to be extended by domain designers (end-users with respect to computational media) who are neither interested in nor trained in the (low-level) details of computational environments. Domain designers are more interested in their design task at hand than in maintaining and evolving knowledge repositories per se. At the same time, important knowledge is produced during daily design activities that should be captured.

Our work on *programmable design environments* [8, 19] is an attempt to increase the meta-design components of design environments. Programmable design environments are based on the objective to make software more "soft": that is, they empower end-users to act as designers by changing and extending the behavior of a given application without substantial reprogramming. Rather than just providing illusory and selective power, they give domain designers the expressive range needed to augment, personalize, and rethink existing systems.

3.6 Related Work

The necessity to overcome the limitations of closed systems and to empower users to become designers has been recognized not only as a desirable goal, but as a necessity in many situations: effective design involves a co-evolution of artifacts with practice. Alexander [1] identified two cultures in design: *self-conscious* and *unself-conscious*. In self-conscious design, the construction of a solution is governed by explicitly represented rules and principles requiring the anticipation of the solution at design time (see Figure 2). In *unself-conscious culture of design*, users will experience breakdowns by recognizing "bad fit" at use time. The

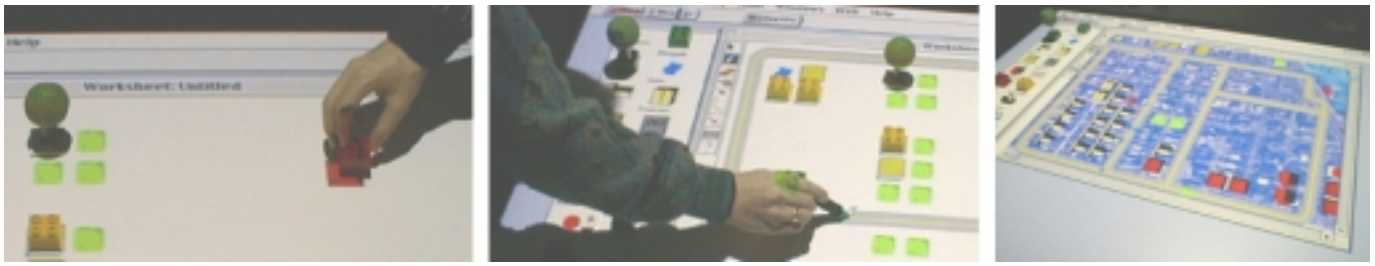

Figure 4: The EDC Environment

knowledge leading to these breakdowns is "tacit" [42], and therefore difficult for users to communicate (at design time) to those who design the artifact. Meta-design allows users to extend the results of self-conscious design activities at design time with unself-conscious design at use time. These observations drawn from the domain of architecture have parallels to the design of interactive systems—because buildings will be modified many times, they should be designed with unanticipated future changes in mind [4].

In the design of interactive systems, the Button system [26] represents an early attempt to create support for meta-design activities and it started with the same premise as the work presented here, namely, that *"it is impossible to design systems which are appropriate for all users and all situations."* Another similarity consisted in emphasizing that to encourage consumers to become designers needs more than technology: it requires a *culture* within which users feel in control and have a designer mindset.

Nardi [29], in her book "A Small Matter of Programming — Perspectives on End User Computing," provides an ethnographic perspective on social and cognitive dimensions of computer use (using spreadsheets and CAD systems as examples). Over the years, we have been especially intrigued by her observation of collaborative work practices that develop around high-functionality applications [16], leading to *power users* and *local developers*. These emerging professional roles make meta-design approaches more feasible because they indicate that design cultures can develop without requiring every user to become a sophisticated designer in all domains.

Henderson and Kyng [22] provide a convincing argument that *"design as a process is tightly coupled to use and continues during the use of the system."* The Problem section above provides evidence that this is a necessity rather than just a luxury, and the SER model provides a conceptual framework of how this might happen, specifically in the context of domain-oriented design environments. The Open Source movement represents a success model of design-in-use by computationally sophisticated communities.

4. THE ENVISIONMENT AND DISCOVERY COLLABORATORY

The Envisionment and Discovery Collaboratory [2] is a conceptual framework for supporting collaborative design activities. The EDC combines physical and computational representations to support both face-to-face and distributed collaboration. A primary goal of the EDC is to explore complex problems by encouraging users to engage in design activities.

At the heart of the EDC are two tightly coupled and highly interactive components that facilitate group interaction. Based on the theoretical paradigm of supporting reflection-in-action [40], these components are called *action* and *reflection* space. In the action space, users manipulate a shared, tangible representation of a problem being constructed. In the reflection space, information relevant to the problem is collected, presented, and extended.

Figure 4 gives a few glimpses of the EDC environment by demonstrating some uses of the environment in the context of urban transportation planning. The core activity in this specific EDC application is the design of alternatives to the current mass-transportation system. Groups of users meet around a horizontal touch-sensitive interactive surface that serves as the EDC action space. In the leftmost pane of Figure 4, we can see how activity takes place around the action space. Users place physical objects that represent aspects of context being described. In this example, a user places a red brick (representing a shopping center) on the board. Other objects include the tree in the upper left (representing a park) and the yellow square in the lower left (representing a home). The touch-sensitive computational whiteboard serves as the interface between physical and computational representations. As users place objects, the underlying computational model is simultaneously updated and projected onto the horizontal surface. This tangible representation helps users to become designers by simplifying the construction process. Users can model a neighborhood quickly and easily by placing appropriate objects on the board. Since the representation is shared, groups of users can work together face-to-face to collaboratively model a specific problem of mutual interest.

The second pane shows a more advanced stage in the construction. Here, a user "draws" a road into the neighborhood. As the user draws, the computer representation makes dynamic changes based on what the users do. The road bends when the user draws a right-angle turn, and the system creates an intersection when two roads cross. This is a simple example of the importance of an active computational representation. The computer model aids design by managing constraints, performing simulations and calculations, and visualizing the ramifications of decisions. The computer simulations enhance the design process by increasing the "talkback" of a model [28], modeling the situation and trying to highlight salient features in the context of the current problem.

In the third pane, the map that serves as a backdrop for the model was retrieved from a database of aerial photographs. Also retrieved from the database was a computer model of the same location. When users ask to model a particular location, this previous computer model and aerial photograph are combined and presented in the action space. This provides a prime example of how information relevant to a problem can be activated and integrated into a situation. Since design problems can potentially activate a huge amount of information, presenting relevant information in a contextualized way helps the users solve the particular design task in question. The tight coupling of the two spaces—actions in the action space can trigger new

information in the reflection space, information in the reflection space can be manipulated in the action space— is an important feature.

The interactive experience of the EDC is more than the sum of its parts, making a static, textual description of the dynamic environment quite difficult. A more comprehensive demonstration of the system is available on our Web site [3]. Exploring this example domain in greater detail helps to explain how the various pieces fit together. As the stakeholders manipulate the shared representation, the computational substrate performs various analyses of the current construction. It can simulate phenomena, such as buses traveling along their routes and people boarding and leaving the buses at the bus stops, and visualize this through the computational display projected onto the action space. Computational mechanisms can analyze the current construction and activate information that might be relevant to the current problem. For example, if people choose to take their cars instead of waiting for the buses, the system can flag this as an issue that the participants should address and activate information related to that problem.

It is instructive to look in detail at some of the features of the EDC that exemplify meta-design principles.

Conviviality. An important design principle behind EDC applications is creating situations in which people wish to participate. We directly support face-to-face collaboration because the participation of various stakeholders is a critical element in framing the problem [40]. For example, a neighborhood transportation issue cannot be adequately discussed without the input of neighbors. The tangible representation (utilizing the physical objects) gives a shared space with boundary objects [41] which are understandable, modifiable and extensible by all stakeholders. By focusing on problems that are important to a community, the EDC encourages people to shape the decisions that affect their daily lives. Through our initial explorations of some domains, we have observed that supporting authentic problems is difficult. This is an example of the challenge of underdesigning the seed for a system. Supporting personally meaningful activities requires creating an initial model that makes the realizations of these problems possible. It is not enough to provide physical objects and simulations that can model a single problem; the tools must be able to help construct new problems, requiring adequate domain support. Determining what amount of support is enough and how the community can help define the important features are difficult questions.

Domain-orientation. Like any other domain-specific system, individual EDC applications support discussion around a class of related problems. The example illustrates the domain of urban planning and mass transportation, which we have explored in some depth. Other domains we have explored include flood mitigation and designing spaces for learning activities (from rooms to buildings; for details, see [3]).

Domain specificity in the EDC comes in the form of physical objects, simulations, and supporting information, which are all designed for the domain in question. In the urban transportation domain, there are physical objects that represent appropriate objects such as houses, parks, schools, and shopping centers. The computational simulations can model phenomena such as bus routes and traffic flow. Supporting information includes local newspaper articles, relevant maps, and information about trade-offs (such as individual car traffic versus public transportation).

By focusing on *collaborative* representations, the domain-specific elements take on a different meanings than they might in another domain-oriented system. The construction is collaborative so there should be some shared understanding among the people constructing a problem. Face-to-face interaction is meant to help make people's tacit assumptions explicit. The relationship between personally meaningful and shared domain-specific representations remains an interesting open issue.

Open systems. To successfully model problems that are meaningful to individuals and communities, there must be opportunities for people to express themselves within the system. The EDC provides different avenues that allow people to contribute in different ways. Collaborative constructions are made quickly with the physical objects, supporting rapid creation of new situations and manipulation of existing ones. Some domain-specific features (such as surveys) can help specify information about a problem. In the urban transportation domain, neighbors can fill out a survey indicating their transportation preferences, and this information can be used to influence the behavior of the computational simulation.

The computational substrates themselves are also designed so that they can be modified when necessary. Our first EDC models have used the AgentSheets simulation environment [36] because of its extension capabilities. Users can add new objects to a simulation. The Visual AgenTalk programming language provides a interactive graphical means to change the behavior and interaction between objects. For example, Visual AgenTalk can help specify if two objects are too close together, and flag a problem if this is true. Information about a problem can be captured and extended by means of dynamic information spaces. For example, DynaSites [31] is a system for building dynamic Web sites, including features such as threaded discussions and extensible glossaries. In one EDC application, issues raised in the computer simulation were linked to threaded discussions in DynaSites. This created a link between face-to-face discussions and the persistent discussions that can be captured in an information space.

Adequate technical infrastructure that makes change possible is important but only part of what is necessary. Although changing many different aspects of the system is possible, not all changes are equally easy. Users can quickly create new situations with physical languages, but changing the mechanics of the simulation by programming in Visual AgenTalk requires much more time and understanding of the system. Although such programming is likely easier than re-programming a model in Lisp or Smalltalk, it still requires a knowledge of the system that a user may not have (or wish to acquire.) Like many systems [26], the EDC aims to provide a gradual transition between different kinds of changes, and provide new opportunities for extension when the need arises. Supporting these changes in the context of EDC activity is challenging. It is unlikely that all changes will be equally easy (or time-consuming), so the process of dealing with changes must be built in to the whole problem-solving process. Creating this culture where changes are a part of the process is a worthwhile goal.

5. ASSESSMENT

An important technical challenge for *meta-design environments* is to capture the informal, situated problem-solving episodes that real people generate in solving real problems. Formal processes have difficulties anticipating or capturing such episodes. Following, we will briefly articulate some of our experiences attempting to support meta-design.

Meta-Design is Hard. Unfortunately, the potential for conviviality exists in many current computer systems only in principle. Many users perceive computer systems as unfriendly and uncooperative, and they view their use as too time consuming; they spend more time fighting the computer than solving their problems. Many users depend on specialists ("high-tech scribes") for help, and despite the fact that they deal with "soft"ware, they do not experience software as "soft" (i.e., the behavior of a system cannot be changed without reprogramming it substantially). The world of computing is separated into a population of elite scribes who can act as designers and a much larger population of intellectually disenfranchised computerphobes who are *forced* into a consumer role.

Beyond Binary Choices. In spite of our arguing for the desirability for humans to be designers [13], it should be stated explicitly that there is nothing wrong with being a consumer. We can learn and enjoy many things in a consumer role (e.g., listening to a lecture, watching a tennis match, or attending a concert). It is a mistake to assume that being a consumer or being a designer has to be a *binary choice*. It is rather a continuum, ranging from passive consumer, to active consumer, to end-user, to user, to power-user [29], to domain designer, to system designer, all the way to meta-designer (see Figure 5, illustrating this fine-grain division of labor among software users). Problems occur when someone wants to be a designer but is forced to be a consumer, or when being a consumer becomes a universal habit and mindset that dominates a human life completely.

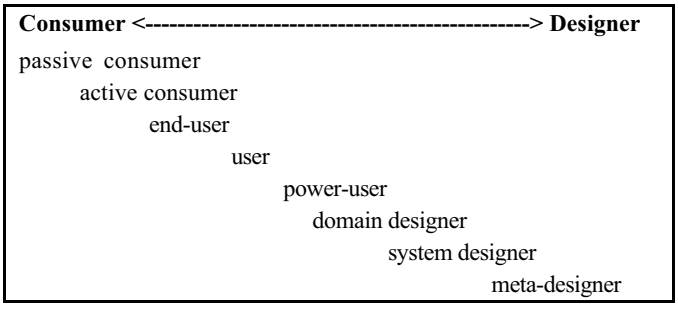

Figure 5: Multiple Roles in the Consumer/Designer Spectrum

Extending Meta-Design to Design for Design Communities. Design (as exemplified by the EDC) is a domain requiring people to think, work, and learn in conjunction or partnership with others and with the help of culturally provided tools and artifacts. A fundamental future challenge for meta-design is to create environments that not only support individual users as designers, but support design communities. Although creative designers are often thought of as working in isolation, the role of interaction and collaboration with other individuals is critical [9]. The predominant activity in designing complex systems is that participants teach and instruct each other [20]. Because complex problems require more knowledge than any single person possesses, it is necessary that all involved stakeholders participate, communicate, and collaborate with each other. Project complexity forces large and heterogeneous groups to work together on projects over long periods of time. Designers generally have a limited awareness and understanding of how the work of other designers within the project—or in similar projects—is relevant to their own part of the design task. The large and growing discrepancy between the amount of such relevant knowledge and the amount any one designer can possibly remember imposes a limit on progress in design. Overcoming this limit is a central challenge for developers of systems that support collaborative design. An specific objective in our current work on meta-design is to enrich our environments by putting more knowledge into the world in the form of *externalizations, oeuvres, and sharable artifacts* [6].

Motivation and Rewards. An important nontechnical challenge for meta-design is to take motivation seriously. There must be an incentive to create *social capital* [35] by rewarding stakeholders for being good citizens by contributing and receiving knowledge as a member of a community. Computational support mechanisms are necessary prerequisites, but not sufficient conditions to motivate people to become part of a "design culture." People must be motivated and rewarded for investing time and effort to become knowledgeable enough to act as designers [21]. These rewards may include (1) feeling in control (i.e., independent from "high-tech scribes"), (2) being able to solve or contribute to the solution of a problem, (3) mastering a tool in greater depth, (4) making an ego-satisfying contribution to a group, (5) and enjoying the feeling of good citizenship to a community [35].

6. CONCLUSIONS

The true contribution of computational media might be to allow all of us to take on or incrementally grow into a designer role in areas that we consider personally meaningful and important.

Meta-design is impossible in communities in which most members regard themselves as consumers. Consumers must evolve into power-users [29] and co-developers [22] who use artifacts and at the same time modify and extend them. A strict separation between these two groups is undesirable and unproductive. One of the major potentials of information technology is giving people the option to become designers by changing and enhancing a software system. One of the major contributions that information technology can lend to the world is to deeply understand and exploit the potential of the malleable nature of software.

Individuals acting as designers must acquire a *new mindset*—they are no longer passive receivers of knowledge, but instead are active researchers, constructors, and communicators of knowledge. Knowledge is no longer handed down from above, but instead is constructed collaboratively in the contexts of work. The foremost objective of meta-design is empowering humans (albeit not all of them, not at all times, not in all contexts) to be and act as designers.

7. ACKNOWLEDGMENTS

The authors would like to thank the members of the Center for LifeLong Learning & Design at the University of Colorado, who have made major contributions to the conceptual framework and systems described in this paper. We are especially grateful to Ernesto Arias, Hal Eden and Andrew Gorman, our co-developers of the EDC over the last five years. The research was supported by (1) the National Science

Foundation, Grants REC-9631396 and IRI-9711951; (2) Software Research Associates, Tokyo, Japan; (3) PFU, Tokyo, Japan; and (4) the Coleman Foundation, San Jose, CA.

8. REFERENCES

1. Alexander, C. *The Synthesis of Form*. Cambridge, MA: Harvard University Press (1964).

2. Arias, E.G., H. Eden, G. Fischer, A. Gorman, and E. Scharff. *Beyond Access: Informed Participation and Empowerment*. In Proceedings of the Computer Supported Collaborative Learning (CSCL '99) Conference, Stanford: pp. 20-32 (1999).

3. Arias, E.G., H. Eden, G. Fischer, A. Gorman, and E. Scharff. *Envisionment and Discovery Collaboratory (EDC) Website* at http://www.cs.colorado.edu/~l3d/systems/EDC/

4. Brand, S. *How Buildings Learn: What Happens After They're Built*. New York: Penguin Books (1995).

5. Brown, J.S., P. Duguid, and S. Haviland. *Toward Informed Participation: Six Scenarios in Search of Democracy in the Information Age.* The Aspen Institute Quarterly. 6(4): pp. 49-73 (1994).

6. Bruner, J. *The Culture of Education*. Cambridge, MA: Harvard University Press (1996).

7. Computer Science Technology Board. *Scaling Up: A Research Agenda for Software Engineering.* Communications of the ACM. 33(3): pp. 281-293 (1990).

8. Eisenberg, M. and G. Fischer. *Programmable Design Environments: Integrating End-User Programming with Domain-Oriented Assistance.* In *Human Factors in Computing Systems, CHI'94 (Boston, MA)*. New York: ACM, pp. 431-437 (1994).

9. Engelbart, D.C. *Toward Augmenting the Human Intellect and Boosting our Collective IQ.* Communications of the ACM. 38(8): pp. 30-33 (1995).

10. Experts-Exchange. *Experts-Exchange Web Site* at http://www.experts-exchange.com

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

12. Fischer, G. *Putting the Owners of Problems in Charge with Domain-Oriented Design Environments.* In *User-Centered Requirements for Software Engineering Environments,* D. Gilmore, R. Winder, and F. Detienne (Eds.). Heidelberg: Springer Verlag, pp. 297-306 (1994).

13. Fischer, G. *Beyond 'Couch Potatoes': From Consumers to Designers.* In *1998 Asia-Pacific Computer and Human Interaction, APCHI'98,* IEEE (Ed.). IEEE Computer Society, pp. 2-9 (1998).

14. Fischer, G. *Seeding, Evolutionary Growth and Reseeding: Constructing, Capturing and Evolving Knowledge in Domain-Oriented Design Environments.* Automated Software Engineering. 5(4): pp. 447-464 (1998).

15. Fischer, G. *Lifelong Learning: Changing Mindsets*. In *7th International Conference on Computers in Education on "New Human Abilities for the Networked Society" (ICCE'99, Chiba, Japan),* G. Cumming, T. Okamoto, and L. Gomez (Eds.). Omaha: IOS Press, pp. 21-30 (1999).

16. Fischer, G. *User Modeling in Human-Computer Interaction.* User Modeling and User-Adapted Interaction, Dordrecht, The Netherlands: Kluwer Academic Publishers. (to appear)(2000).

17. Fischer, G., K. Nakakoji, J. Ostwald, G. Stahl, and T. Sumner. *Embedding Critics in Design Environments.* In *Readings in Intelligent User Interfaces,* M.T. Maybury and W. Wahlster (Eds.). San Francisco: Morgan Kaufmann, pp. 537-561 (1998).

18. Fischer, G. and E. Scharff. *Learning Technologies in Support of Self-Directed Learning.* Journal of Interactive Media in Education. 1998(4)(1998).

19. Girgensohn, A., *End-User Modifiability in Knowledge-Based Design Environments,* in *Department of Computer Science*, Boulder, CO: University of Colorado at Boulder, pp. 190 (1992).

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

21. Grudin, J. *Groupware and Social Dynamics: Eight Challenges for Developers.* Communications of the ACM. 37(1): pp. 92-105 (1994).

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

23. Illich, I. *Deschooling Society*. New York: Harper and Row (1971).

24. Illich, I. *Tools for Conviviality*. New York: Harper and Row (1973).

25. Ingalls, D., T. Kaehler, J. Maloney, S. Wallace, and A. Kay. *Back to the Future: The Story of Squeak, a Practical Smalltalk Written in Itself*. In Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA), Atlanta, GA: ACM, pp. 318-326 (1997).

26. MacLean, A., K. Carter, L. Lovstrand, and T. Moran. *User-Tailorable Systems: Pressing the Issues with Buttons.* In *Proceedings of CHI'90 Conference on Human Factors in Computing Systems,* J. Carrasco and J. Whiteside (Eds.). New York: ACM, pp. 175-182 (1990).

27. Maxis. *SimCity 3000* at http://www.simcity.com

28. Nakakoji, K., Y. Yamamoto, T. Suzuki, S. Takada, and M. Gross. *From Critiquing to Representational Talkback: Computer Support for Revealing Features in Design.* Knowledge-Based Systems Journal. 11(7-8): pp. 457-468 (1998).

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

30. O'Reilly, T. *Lessons from Open Source Software Development.* Communications of the ACM. 42(4): pp. 33-37 (1999).

31. Ostwald, J. *DynaSites* at http://www.cs.colorado.edu/~ostwald/dynasites.html

32. PCAST. *Report to the President on the Use of Technology to Strengthen K-12 Education in the United States* at http://www.whitehouse.gov/WH/EOP/OSTP/NSTC/PCAST/k-12ed.html.

33. Postman, N. *Amusing Ourselves to Death—Public Discourse in the Age of Show Business*. New York: Penguin Books (1985).

34. Raymond, E.S. *The Cathedral and the Bazaar* at http://www.tuxedo.org/~esr/writings/cathedral-bazaar/

35. Raymond, E.S. *Homesteading the Noosphere* at http://www.tuxedo.org/~esr/writings/homesteading/

36. Repenning, A. *AgentSheets Web Site* at http://www.agentsheets.com/

37. Repenning, A., A. Ioannidou, and J. Phillips. *Collaborative Use & Design of Interactive Systems*. In Proceedings of the Computer Supported Collaborative Learning (CSCL '99) Conference, Stanford: pp. 475-487 (1999).

38. Resnick, M., *Beyond the Centralized Mindset: Explorations in Massively-Parallel Microworld,* in *Department of Computer Science*, Cambridge, MA: Massachusetts Institute of Technology, pp. 176 (1992).

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

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

41. Wenger, E. *Communities of Practice — Learning, Meaning, and Identity*. Cambridge, England: Cambridge University Press (1998).

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