The future success of EUD depends on creating tools that end users are motivated to learn and use in daily work practices.

META-DESIGN: A MANIFESTO FOR END-USER DEVELOPMENT

By G. Fischer, E. Giaccardi, Y. Ye, A.G. Sutcliffe, *and* N. Mehandjiev

END-USER DEVELOPMENT (EUD) ACTIVITIES RANGE from customization to component configuration and programming. Office software, such as the ubiquitous spreadsheet, provides customization facilities, while the growth of the Web has added impetus to end-user scripting for interactive functions in Web sites. In scientific and engineering domains, end users frequently develop complex systems with standard programming languages such as C++ and Java. However, only a minority of users adapt commercial off-the-shelf (COTS) software products. Indeed, composing systems from reusable components, such as enterprise resource planing (ERP) systems, defeats most end users who resort to expensive and scarce expert developers for implementation.

Therefore, EUD is only a partial success story. Here, we argue the spread of EUD depends on a fine balance between user motivation, effective tools, and management support. We explore that balance and investigate a future approach to EUD—meta-design—that proposes a vision in which design, learning, and development become part of everyday working practice.

Designing language for user-computer communication poses a conflict between complexity and power. More complex languages can address a wider range of problems but impose an increased learning burden on the user. Text-based languages tend to be more complex because the syntax and lexicon (terminology) must be learned from scratch, as with any human language. Consequently, languages designed specifically for end users represent the programmable world as graphical metaphors containing agents that can be instructed to behave by condition-action rules. The aim is to reduce the cognitive burden of learning by shrinking the conceptual distance between actions in the real world and programming.

A key trade-off in EUD languages is between their scope of application and learning costs, as illustrated in Figure 1. In the *high cost, high scope* cell are traditional programming languages, Java and C++, employed by highly motivated end users particularly in scientific domains. At the convergence of this cell and the high scope, lower cost cell are the majority of current EUD languages that have evolved as simplified versions of full programming languages, for example, Web scripting languages. The *low scope, high cost* cell is occupied by only a small number of domain-specific programming languages developed to address the requirements in complex engineering domains, such as device controllers. These languages impose a considerable learning burden, but are worth it for improving efficiency over a general-purpose language. The *low cost, low scope* cell contains domain-specific EUD languages that lower the learning burden but at the price of addressing only a specific application area. In this cell, EUD languages merge with the customization of COTS software packages so the act of programming is reduced to entering parameters in a form-filling dialogue. Closer to the higher scope boundary are macro languages that extend the office-style applications, for example, formulae for Excel spreadsheets, and database query languages. Finally, the *high scope, low cost* cell is the EUD ideal, although still largely unattained. The current state-of-the-art EUD environments provide graphical worlds to create programmable agents that still impose a learning burden of instructing agents with condition-action rules, and designing agent models.

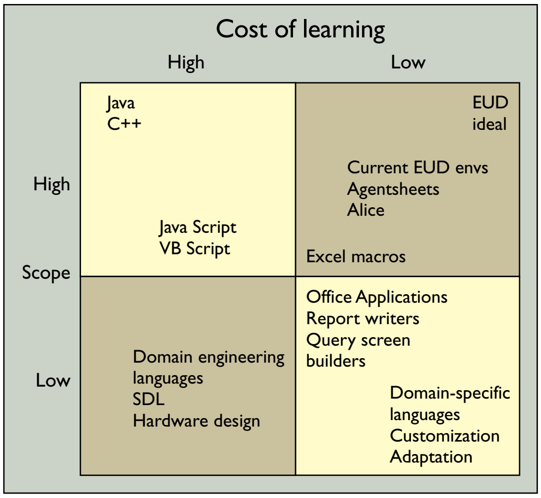

Figure 1. Cost-scope trade-offs in EUD tools.

Active EUD environments attempt to infer programs as instructions from user manipulations of agent worlds. The graphical agent worlds must still be designed, but once present, programming by example [5] can infer instructions from the users' actions; for example, in a robot game the user demonstrates an agent bumping into a wall followed by reversing two steps and changing direction. The system infers the condition-action rule of detect-a-collision followed by the appropriate reverse-and-change-direction response. This approach reduces learning by semiautomatic rule acquisition, but the downside is the learning system can make mistakes. The learning styles range from more complete inference to direct instruction, where the system learns only when given a command. Direct instruction requires the user to anticipate all the possible rules and learning situations, while the complete inference approach is limited by the system's domain knowledge. Developing the model is the difficult part and therein lies the real challenge for end-user design—abstract conceptual thinking. Complex domains require sophisticated analysis and modeling skills; programming is only part of an end-user developer's needs.

The goal for EUD tools is to reduce the learning burden while providing powerful facilities to address a wide range of problems. Given that some learning burden will always be present, tools must motivate their users. We propose a meta-design approach [4], where users are motivated to learn by examples and demonstrations of working systems to show them what is achievable.

Managerial and Social Perspective

EUD is a long-standing concern within organizations. Managerial issues, illustrated in Figure 2, are based on previous surveys of end-user computing [2, 9] and our more recent investigation into the task-organizational fit of EUD technology [6]. Do-it-yourself development is a balance of benefits and cost. *User motivators* are empowerment from being able to complete a job more effectively, speed of development, flexibility and local control so programming can be on demand. Another benefit is eliminating potential miscommunications of requirements to specialist software engineers, thus avoiding the frustration with perceived poor service from the IS department. Success stories can create motivational capital to help users over the hump of learning until actual benefits arrive in the form of working applications. User motivation should be encouraged during the early stages of adoption by management support, training, and task forces to spread best practice and expertise. This counteracts user costs such as selecting appropriate technology, installing and learning it, programming, and debugging.

A number of context and management issues influence the balance between costs and benefits. For example, EUD can be dangerous in safety-critical domains where software must be reliable and accurate. User costs can be significantly influenced by the scale and complexity of the domain, so safer, less complex domains should be selected for EUD. Changeability of the domain can be a motivator for EUD adoption, since end users can respond to rapidly evolving requirements more quickly than traditional development; however, rapid change can lead to throw-away software and lost development effort. Management issues include risks associated with EUD, perceived by IT management to create unreliable and unmanageable software. Other risks are inaccurate information and security with increased exposure to hacking attacks.

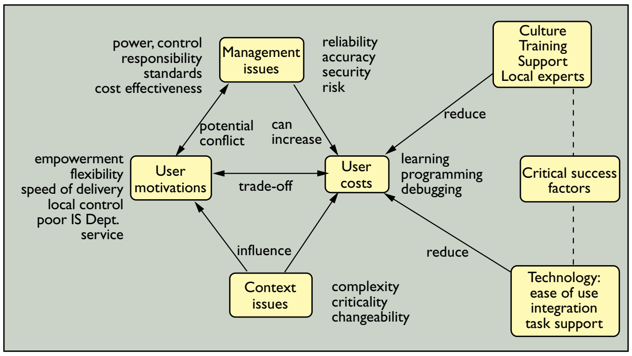

Figure 2. Relationships between social and managerial issues in EUD.

The conflict between IT management and end users over power, authority, and control of IT systems may be a productive force for change or it may lead to disruption, mistrust, and failure. It can be argued that enforcing standards and controlling end users leads to more cost-efficient development and less waste from unreliable software. However, rigid top-down control may only cause resentment among end users. The control-power conflict between users and IT management will not evaporate; but constructive support and training fosters success, encourages responsibility, and enables management to control by leadership.

Critical success factors for EUD depend on the domain. In a culture of high end-user motivation and low managerial influence—a common situation in scientific and engineering domains, educational applications, and interactive art—success is simply a matter of users taking development into their own hands, often using standard programming languages. However, in most business domains, training, technical, and management support are vital for helping EUD flourish. A culture of cooperation shares the responsibility for developing accurate and effective solutions. Local experts among the end-user community spread expertise and advice, although power users can be prone to migrating to the wrong side of the "us" and "them" (IT department) fence [7, 8]. Technology should provide easy integration with other information systems and optimized support for EUD tasks. Progress in the technology area is still necessary to unlock the true potential for EUD.

The set of EUD critical success factors suggests the need for a socio-technical approach to increase user motivation and decrease cognitive and organizational costs. Such an approach suggests a future technological framework with tools for discovery-led design to balance learning costs with results-driven motivation. We propose meta-design, which is an evolution of domain-oriented design environments (DODEs) [3] as a vision in which design, learning, and development become everyday working practice.

Meta-design

Meta-design characterizes objectives, techniques, and processes for creating new media and environments allowing "owners of problems" (that is, end users) to act as designers. A fundamental objective of meta-design is to create socio-technical environments that empower users to engage actively in the continuous development of systems rather than being restricted to the use of existing systems.

In all design processes, two basic stages can be differentiated: *design time* and *use time*. At design time, system developers (with or without user involvement) create environments and tools. In conventional design they create complete systems. Because the needs, objectives, and situational contexts of users can only be anticipated at design time, users often find the system unfit for their tasks at use time, thus requiring modification of existing systems. To accommodate unexpected issues at use time, systems must be "underdesigned" at design time. Underdesign represents a fundamental shift in the approach to the creation of systems, but it does not mean less work or fewer demands on the design team. Instead of designers aiming at designing complete solutions for users at design time, underdesign aims to provide social and technical instruments for the owners of problems to create the solutions themselves at use time. Within the overall approach of meta-design, underdesign is a defining activity to create design spaces for others.

Meta-design extends the traditional notion of system development to include users in an ongoing process as co-designers, not only at design time but throughout the entire existence of the system. A necessary, although not sufficient, condition for meta-design is that software systems include advanced features permitting users to create complex customizations and extensions. Rather than presenting users with closed systems, meta-design provides them with opportunities, tools, and social structures to extend the system to fit their needs. Meta-design shares some important objectives with user-centered and participatory design, but it transcends these objectives by changing the processes by which systems and content are designed. Meta-design shifts control from designers to users and empowers users to create and contribute their own visions and objectives. Meta-design promotes "designing the design process" to a first-class activity, so that creating the technical and social conditions for broad participation in design activities becomes as important as creating the artifact itself. It creates the enabling conditions for collaborative design in which all participants, not just skilled computer professionals, incrementally acquire ownership of problems and contribute actively to their solutions.

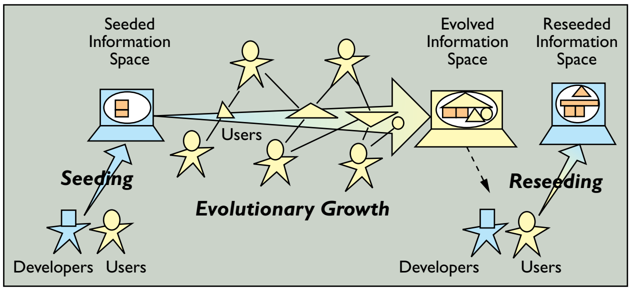

Figure 3. The seeding, evolutionary growth, and reseeding process model.

To support meta-design, we have developed the *seeding, evolutionary growth*, and *reseeding (SER)* process model. As illustrated in Figure 3, SER is a descriptive and prescriptive model for large evolving systems and information repositories, postulating that systems that evolve over a sustained time span must continually alternate between periods of activity, unplanned evolution, and periods of deliberate (re)structuring and enhancement. The SER model encourages designers to conceptualize their activity as meta-design, thereby supporting users as designers rather than restricting them to passive consumers.

To demonstrate the broad applicability and power of meta-design, we have applied the framework in a number of different application areas, including three briefly mentioned here:

Social creativity. Complex design problems require more knowledge than any single person can possess, and the knowledge relevant to a problem is often distributed among stakeholders from different perspectives and backgrounds. The solution of complex design problems requires social creativity in which all stakeholders reach a shared understanding by contributing their different points of view and knowledge. We have applied the meta-design approach in the creation of augmented reality environments in urban planning [1]. The tools themselves are not solutions to any particular problem, but provide the socio-technical environment for stakeholders to become informed participants. The immediate and visual feedback facilitates the creation of a shared understanding leading to new insights, new ideas, and new artifacts as a result of collaboration.

Open source. Open source development is an activity in which a community of software developers collaboratively constructs systems to help solve problems of shared interest and for mutual benefit. The original designers of an open source system do not provide a complete solution that addresses all problems of potential users; they provide a seed that can be evolved by users at use time. The ability to change source code, the technological means of sharing changes over the Internet, and the spontaneous social support among community members are the enabling conditions for collaborative construction of software. Software is changed from a fixed entity produced and controlled by a closed group of designers to an open effort that allows a community to design collaboratively following the framework provided by the SER process model. The success of open source systems exemplifies meta-design by openly embracing users as co-designers by releasing incomplete code; actively soliciting and incorporating user contributions; strategically sharing the control over original designers and users by granting users direct access to source code; aggressively promoting mutual learning among community members through mailing lists; and deliberately fostering a reward and recognition structure that motivates active participation by explicitly acknowledging and promoting contributors [10].

Open source projects based on meta-design have a lower cost for each user because the development cost is distributed among a large number of participants and individual contributions are shared.

Interactive art. Conceptualized as meta-design, interactive art [4] focuses on participation and collaboration as forms of co-creation, in which users become co-developers of artwork. The original seed design establishes a context in which users can actively produce new content and meaning through a process of mutual interaction and evolutionary growth. By putting the tools rather than the object of design in the hands of users, interactive art seeds collaboration between the participants (both technical and human) and sees this interaction as the real object of creative production. Hence meta-design creates interactive systems that define the conditions for interaction. Meta-design environments not only allow users to create content, but also modify the behavior and components of the system at use time through interaction (see A-Volve; www.iamas.ac.jp/~christa/). The initial seed is often developed by a community of artists, and can be adjusted and improved according to the talk-back deriving from the continuing experience of using the creative environment as in SITO, (www.sito.org), a virtual community of artists-participants. Interaction and evolution occur both at the level of the development of materials and at the level of the creation, elaboration and completion of collective artworks. Interactive art emphasizes different objectives compared to traditional design approaches, including cultural shifts from following guidelines and rules to learning from exceptions and negotiations, content to context of design, changing focus from design objects to process, and from working with representation to the act of construction.

Conclusion

To evolve, EUD development needs technologies that foster collaboration between communities of end-user designers and users and managers, while increasing motivation and reducing cognitive and organizational costs. Meta-design provides a pathway to transform development as coding—a discrete computing activity—into design of artifacts as part of the users' work (or leisure) practice.

Meta-design puts owners of problems in charge of creating open, evolvable systems that address the limitations associated with closed systems. Open systems allow significant modifications when the need arises and the evolution takes place through modifications by the owners of problems as a major design activity. Meta-design is more than a technical problem; it must address the challenges of creating new mindsets, new sources of creativity, cultural changes, and innovative societies. It has the potential to create a culture in which all participants in collaborative design processes can express themselves and engage in personally meaningful activities. C

References

1. Arias, E.G., Eden, H., Fischer, G., Gorman, A., and Scharff, E. Transcending the individual human mind: Creating shared understanding through collaborative design. *ACM Trans. on Computer-Human Interaction 7*, 1 (2000). ACM, NY, 84–113.
2. Brancheau, J.C., and Brown, C.V. The management of end user computing: Status and directions. *ACM Computing Surveys 25,* 4. ACM, NY, (1993), 437–482.
3. Fischer, G. Domain-oriented design environments. *Automated Software Engineering, 1,* 2 (1994), 177–203.
4. Fischer, G., and Giaccardi, E. Meta-design: A framework for the future of end user development. *End User Development: Empowering People to Flexibly Employ Advanced Information and Communication Technology*. H. Lieberman, F. Paternò, and V. Wulf, Eds. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2004, in press.
5. Lieberman, H., Ed. *Your Wish Is My Command: Programming By Example.* Morgan Kaufmann, San Francisco, 2001.
6. Mehandjiev, N., Sutcliffe, A.G., and Lee, D. Organisational views of end user development. *End User Development: Empowering People to Flexibly Employ Advanced Information and Communication Technology.* H. Lieberman, F. Paternò, and V. Wulf, Eds. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2004, in press.
7. Mumford, E., and Henshall, D. *A Participative Approach to Computer System Design.* Associated Business Press, London, 1979.
8. National Research Council. *Beyond Productivity: Information Technology, Innovation and Creativity.* National Academy Press, Washington, D.C., 2003.
9. Powell, A., and Moore, J.E. The focus of research in end user computing: Where have we come since the 1980s? *Journal of End User Computing 14,* 1 (2002), 3–22.
10. Ye, Y., and Kishida, K. Toward an understanding of the motivation of open source software developers. In *Proceedings of the 25th International Conference on Software Engineering* (Portland OR, 2003) ACM, NY, 419–429.

G. Fischer (gerhard@cs.colorado.edu) is a professor in the Department of Computer Science and director of the Center for Lifelong Learning and Design at the University of Colorado at Boulder.
E. Giaccardi (giaccard@cs.colorado.edu) is a research associate in the Center for Lifelong Learning and Design in the Department of Computer Science at the University of Colorado at Boulder.
Y. Ye (yuwen@cs.colorado.edu) is a research associate in the Center for Lifelong Learning and Design in the Department of Computer Science at the University of Colorado at Boulder.
A.G. Sutcliffe (a.sutcliffe@co.umist.ac.uk) is a professor of systems engineering and director of the Centre for Human Computer Interface Design in the School of Informatics at the University of Manchester, U.K.
N. Mehandjiev (mehandjiev@acm.org) is a senior lecturer in the School of Informatics at the University of Manchester, U.K.