Chapter 1

End-User Development: An Emerging Paradigm

HENRY LIEBERMAN[1], FABIO PATERNÓ[2], MARKUS KLANN[3]
and VOLKER WULF[4]
[1] *MIT, 20 Armes Street 305, Cambridge, Massachussets, 02139 USA, lieber@media.mit.edu*
[2] *ISTI—CNR, Via G. Moruzzi 1, 56124 Pisa, Italy, fabio.paterno@isti.cnr.it*
[3] *Fraunhofer FIT, Schloß Birlinghoven, 53754 Sankt Augustin, Germany, markus.klann@fit.fraunhofer.de*
[4] *University of Siegen, Hölderlinstr. 3, 57068 Siegen and Fraunhofer FIT, Schloß Birlinghoven, 53754 Sankt Augustin, Germany, volker.wulf@uni-siegen.de*

Abstract. We think that over the next few years, the goal of interactive systems and services will evolve from just making systems easy to use (even though that goal has not yet been completely achieved) to making systems that are easy to develop by end users. By now, most people have become familiar with the basic functionality and interfaces of computers, but they are not able to manage any programming language. Therefore, they cannot develop new applications or modify current ones according to their needs.

In order to address such challenges it is necessary a new paradigm, based on a multidisciplinary approach involving several types of expertise, such as software engineering, human-computer interaction, CSCW, which are now rather fragmented and with little interaction. The resulting methods and tools can provide results useful across many application domains, such as ERP, multi-device services (accessible through both mobile and stationary devices), and professional applications.

Key words. tailorability, end user programming, flexibility, usability

We think that over the next few years, the goal of human–computer interaction (HCI) will evolve from just making systems *easy to use* (even though that goal has not yet been completely achieved) to making systems that are *easy to develop*. By now, most people have become familiar with the basic functionality and interfaces of computers. However, developing new or modified applications that effectively support users' goals still requires considerable expertise in programming that cannot be expected from most people. Thus, one fundamental challenge for the coming years is to develop environments that allow users who do not have background in programming to develop or modify their own applications, with the ultimate aim of empowering people to flexibly employ advanced information and communication technologies.

Current trends in professional life, education, and also in leisure time are characterized by increasing change and diversity: changing work and business practices, individual qualifications and preferences, or changes in the dynamic environments in which organizations and individuals act. The diversity concerns people with different skills, knowledge, cultural background, and cognitive or physiological abilities, as well

Henry Lieberman et al. (eds.), End User Development, 1–8.
© 2006 *Springer.*

as diversity related to different tasks, contexts, and areas of work. Enhancing user participation in the initial design of systems is part of the solution. However, given that user requirements are diversified, changing, and at times hard to identify precisely, going through conventional development cycles with software professionals to keep up with evolving contexts would be too slow, time consuming, and expensive. Thus, flexibility really means that the users themselves should be able to continuously adapt the systems to their needs. End-users are generally neither skilled nor interested in adapting their systems at the same level as software professionals. However, it is very desirable to empower users to adapt systems at a level of complexity that is appropriate to their individual skills and situations. This is the main goal of EUD: empowering end-users to develop and adapt systems themselves. Some existing research partially addresses this issue, advocating casting users as the initiators of a fast, inexpensive, and tight co-evolution with the systems they are using (Arondi et al., 2002; Mørch, 2002; Wulf, 1999; see also the "Agile Programming" techniques of Beck, 1999 and Cockburn, 2002).

This insight, which developed in various fields of software engineering (SE) and HCI, has now become focused in the new research topic of end-user development (EUD). To enable systems for EUD, they must be made considerably more flexible and they must support the demanding task of EUD: they must be easy to understand, to learn, to use, and to teach. Also, users should find it easy to test and assess their EUD activities.

Though there are diverse views on what constitutes EUD, we attempt below to give a working definition of it:

EUD can be defined as a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify, or extend a software artifact.

Today, some forms of EUD have found widespread use in commercial software with some success: recording macros in word processors, setting up spreadsheets for calculations, and defining e-mail filters. While these applications only realize a fraction of EUD's potential and still suffer from many flaws, they illustrate why empowering end-users to develop the systems they are using is an important contribution to letting them become active citizens of the Information Society.

Boehm et al. (2000) predicted exponential growth of the number of end-user developers compared to the number of software professionals, underscoring the importance of research in EUD. The potential to provide EUD services over the Internet may create a shift from the conventional few-to-many distribution model of software to a many-to-many distribution model. EUD could lead to a considerable competitive advantage in adapting to dynamically changing (economic) environments by empowering end-users—in particular domain experts (Costabile et al., 2003)—to perform EUD. The increasing amount of software embedded within consumer and professional products also points to a need to promote EUD to enable effective use of these products.

On the political level EUD is important for full participation of citizens in the emerging Information Society. The Information Society is characterized by computer networks, which will becoming the leading media, integrating other traditional media within digital networks and enabling access through a variety of interaction devices ranging from small mobile phones to large flat screens. However, the creation of content

and the modification of the functionality of this network infrastructure are difficult for non-professional programmers, resulting for many sectors of society in a division of labor between those who produce and those who consume. EUD has the potential to counterbalance these effects.

The emerging research field of EUD integrates different threads of discussion from HCI, SE, computer supported cooperative work (CSCW), and artificial intelligence (AI). Concepts such as tailorability, configurability, end-user programming, usability, visual programming, natural programming, and programming by example already form a fruitful base, but they need to be better integrated, and the synergy between them more fully exploited.

We can identify two types of end-user activities from a user-centered design perspective:

1. *Parameterization or customization*. Activities that allow users to choose among alternative behaviors (or presentations or interaction mechanisms) already available in the application. Adaptive systems are those where the customization happens automatically by the system in reaction to observation the user's behavior.
2. *Program creation and modification.* Activities that imply some modification, aiming at creating from scratch or modifying an existing software artifact. Examples of these approaches are: programming by example (also called programming by demonstration), visual programming, macros, and scripting languages.

EUD more properly involves the second set of activities since with the first set the modification of software is restricted to strictly predefined options or formats. However, we often want to design for a "gentle slope" of increasing complexity to allow users to easily move up from the first to the second set of activities. Examples of activities belonging to the first type are:

Parameterization. In this commonly occurring case, the user wishes to guide a computer program by indicating how to handle several parts of the data in a different way; the difference may simply lie in associating specific computation parameters to specific parts of the data, or in applying different program functionalities to the data.

Annotation. The users write comments next to data and results in order to remember what they did, how they obtained their results, and how they could reproduce them.

Examples of activities belonging to the second type are:

Programming by example. Users provide example interactions and the system infers a routine from them (Lieberman, 2001).

Incremental programming. This is close to traditional programming, but limited to changing a small part of a program, such as a method in a class. It is easier than programming from scratch.

Model-based development. The user just provides a conceptual description of the intended activity to be supported and the system generates the corresponding interactive application (Paternò, 2001).

Extended annotation or parameterization. A new functionality is associated with the annotated data or in a cooperative environment users identify a new functionality by selecting from a set of modifications other people have carried out and stored in shared repositories.

To start looking at EUD research, let us distinguish between research on end-user participation *during the initial design phase* and research on end-user modification *during usage*. As EUD implies that design can extend beyond an initial, dedicated design phase, this is not really a sharp distinction.

Providing support during a dedicated design phase aims at better capturing and satisfying user requirements. Research in this area seeks to develop domain-specific, possibly graphical modeling languages that enable users to easily express the desired functionality (cf. Mehandjiev and Bottaci, 1996; Paternò et al., 1994; Repenning et al., 2000). Such modeling languages are considered an important means of bridging the "communication gap" between the technical view of software professionals and the domain expert view of end-users (Majhew, 1992; Paternò, 2001). In particular, work is being done on using the extension mechanisms of the unified modeling language (UML), a set of graphical representations for modeling all aspects of software systems, to create a representational format for end-users. Another complementary approach to bringing these two views closer together is the use of scenarios in development as a communicative aid.

As noted above, an initial design tends to become outdated or insufficient fairly quickly because of changing requirements. Challenging the conventional view of "design-before-use," new approaches try to establish "design-during-use" (Dittrich et al., 2002; Mehandjiev and Bottaci, 1996), leading to a process that can be termed "evolutionary application development." System changes during use might be brought about by either explicit end-user requests or automatically initiated state transitions of the system. In the first case, the system is called *adaptable*, whereas in the second, *adaptive* (Oppermann and Simm, 1994).

Such a scenario raises the need for system flexibility that allows for modifications that go well beyond simple parameterizations, while being substantially easier than (re)programming. More precisely, a system should offer a range of different modification levels with increasing complexity and power of expression. This is to ensure that users can make small changes simply, while more complicated ones will only involve a proportional increase in complexity. This property of avoiding big jumps in complexity to attain a reasonable trade-off is what is called the "gentle slope" (Dertouzos, 1997; MacLean et al., 1990; Wulf and Golombek, 2001). As an example, a system might offer three levels of complexity: First, the user can set parameters and make selections. Second, the user might compose existing components. Third, the user can extend the system by programming new components (Henderson and Kyng, 1991; Mørch, 1997; Stiemerling, 2000). Modular approaches can generally provide a gentle slope for a range of complexity by allowing successive decomposition and reconfiguration of software entities that are themselves build up from smaller components (e.g., Won et al., in this volume). The precondition for this is that a system's component structure has been

designed to be meaningful for its users, and that these users are able to easily translate changes in the application domain into corresponding changes in the component structure.

While adaptivity alone does not constitute EUD because the initiative of modifications is with the system, it is interesting to combine it with end-user driven activities. Users may want to stay in control of how systems adapt themselves and might have to supply additional information or take certain decisions to support system adaptivity. Conversely, the system might try to preselect the pertinent EUD options for a given context or choose an appropriate level of EUD for the current user and task at hand, thus enhancing EUD through adaptivity. Mixed forms of interactions where adaptive systems can support interaction but users can still take the initiative in the development process may provide interesting results, as well.

How do we make functionality for adaptation available at the user interface? First, adaptation should be unobtrusive, so as not to distract user attention from the primary task. At the same time, the cognitive load of switching from using to adapting should be as low as possible. There seems to be a consensus that adaptation should be made available as an extension to the existing user interface. A related issue is how to make users aware of existing EUD functions and how to make these functions easily accessible (e.g., Wulf and Golombek, 2001).

Another key research area deals with cooperative EUD activities, having its roots in research on CSCW. It investigates topics such as collaborative development by groups of end-users (Letondal, 2001; Mørch and Mehandjiev, 2000), privacy issues, and repositories for sharing artifacts among end-users (Kahler 2001; Wulf 1999). This research also includes recommending and awareness mechanisms for finding suitable EUD expertise as well as reusable artifacts. We should foster the building of communities where end-users can effectively share their EUD-related knowledge and artifacts with their peers (Costabile et al., 2002; Pipek and Kahler, in this volume).

Flexible software architectures are a prerequisite for enabling adaptivity. Approaches range from simple parameters, rules, and constraints to changeable descriptions of system behavior (meta-data) and component-based architectures (Won et al., in this volume). A key property of an EUD-friendly architecture is to allow for substantive changes during run-time, without having to stop and restart or rebuild the system.

Enabling end-users to substantially alter systems creates a number of obvious issues concerning correctness and consistency, security, and privacy. One approach to handling these issues is to let the system monitor and maintain a set of desired system properties during EUD activities. For example, properties like integrity and consistency can be maintained by only allowing safe operations. Nonetheless, user errors and incompleteness of information cannot be ruled out altogether (Lieberman, 2001). Users may often be able to supply missing information or correct errors if properly notified. For this reason, it may be best to adopt a mixed-initiative approach to dealing with errors (Horvitz, 1999).

Finally, another approach to improving EUD is to create languages that are more suited to non-programmers and to specifying requirements than are conventional

programming languages. In particular, domain-specific and graphical languages are being investigated (e.g., Paternò et al., 1994).

At the center of EUD are the users and their requirements (Stiemerling et al., 1997). The increasing change and diversity engendered by networked mobile and embedded devices will enable access to interactive services anywhere and anytime in diverse contexts of use. Therefore, EUD environments should support easy generation of interfaces able to adapt the device's features (e.g., Berti et al., in this volume). Also, systems are used by diverse groups of people, with varying levels of expertise, current tasks, and other factors. Systems should be able to adapt to the changing contexts and requirements under the user's control and understanding.

EUD is a socio-cultural activity, depending on place, time, and people involved. Differences in EUD practice are likely to develop for different cultures and languages. Obviously, this is of particular importance for cross-cultural collaboration. Another area where such differences are likely to show up is EUD of groupware systems, whether this EUD is done cooperatively or not. These differences may relate to who is in control of EUD activities, to the relation between individual and collaborative EUD, and to how communities of end-user developers are organized.

Embedding systems into heterogeneous environments cannot be completely achieved before use, by determining the requirements once and deriving an appropriate design. Rather, adaptation must continue as an iterative process by the hands of the users, blurring the border between use and design. A given system design embodies a certain semiotic model (Lehman, 1980) of the context of use, and that EUD allows users to adapt this model to reflect their natural evolution. Furthermore, using a system changes the users themselves, and as they change they will use the system in new ways (Carroll and Rosson, 1992; Pipek and Wulf, 1999). Systems must be designed so that they can accommodate user needs that cannot be anticipated in the requirement phase, especially those that arise because of user evolution (Costabile et al., 2003).

Being a relatively young field, EUD is yet rather diversified in terms of terminology, approaches, and subject areas. Networking within the EUD-community has started only relatively recently (Sutcliffe and Mehandjiev, 2004). One such effort was the EU-funded Network of Excellence EUD-Net,[1] bringing together leading EUD researchers and industry players from Europe. Later on, the US National Science Foundation funded end-user software engineering systems (EUSES), investigating whether it is possible to bring the benefits of rigorous SE methodologies to end-users. It is generally felt that there is a strong need for thorough empirical investigations of new EUD-approaches in real-world projects, both to solidify the theoretical groundings of EUD, and to develop more appropriate methods and tools for deploying and using EUD-systems. Further research initiatives are on the way in the 7th Framework Program of the EU as well as by single European states such as Germany.

The present book is an effort to make many important aspects of the international EUD discussion available to a broader audience. A first set of papers resulted from

[1] For more information on EUD-Net see http://giove.isti.cnr.it/eud-net.htm.

two EUD-Net events: a research workshop held in September 2002 at ISTI-CNR in Pisa, Italy, and the International Symposium on EUD held in October 2003 in Schloss Birlinghoven, Germany. Beyond these contributions, we invited some papers from other leading researchers in the field. We hope that this broad look at the emerging paradigm of EUD leads you to appreciate its diversity and potential for the future. And we look forward to having you, the reader, the "end-user" of this book, contribute what you can to the field, whether it is working on a system for EUD, or simply achieving a better understanding of how EUD might fit into your work and your life.

References

Arondi, S., Baroni, P., Fogli, D. and Mussio, P. (2002). *Supporting Co-evolution of Users and Systems by the Recognition of Interaction Patterns.* Trento, Italy: AVI.

Beck, B. (1999). *Extreme Programming Explained: Embrace Change*. Reading, MA: Addison-Wesley.

Berti, S., Paternó, F. and Santoro, C. Natural Development of Nomadic Interfaces Based on Conceptual Descriptions, in this volume.

Boehm, B.W., Abts, C., Brown, A., Chulani, S., Clark, B., Horowitz, E., Modochy, R., Reifer, D. and Steece, B. (2000). *Software Cost Estimation with COCOMO II*. Upper Saddle River, NJ: Prentice Hall PTR.

Carroll, J.M. and Rosson M.B. (1992). Getting around the task-artifact cycle: How to make claims and design by Scenario. *ACM Transactions on Information Systems* **10**(2), 181–212.

Cockburn, A. (2002). *Agile Software Development*. Reading, MA: Addison Wesley.

Costabile, M.F., Fogli, D., Fresta, G., Mussio, P. and Piccinno, A. (2002). *Computer Environments for Improving End-User Accessibility*. ERCIM Workshop "User Interfaces For All", Paris.

Costabile, M.F., Fogli, D., Fresta, G., Mussio, P. and Piccinno, A. (2003). Building environments for end-user development and tailoring. In: *IEEE Symposia on Human Centric Computing Languages and Environments*, Aukland.

Dertouzos, M. (1997). *What Will Be: How the New World of Information Will Change Our Lives*. New York: Harper-Collins.

Dittrich, Y., Eriksén, S. and Hansson, C. (2002). *PD in the Wild: Evolving Practices of Design in Use*. Malmö, Sweden: PDC.

Henderson, A. and Kyng M. (1991). *There's No Place Like Home. Continuing Design in Use*. Design at Work, Hillsdale, NJ: Lawrence Erlbaum Assoc. pp. 219–240.

Horvitz, E. (1999). Principles of mixed-initiative user interfaces. In *Proceedings ACM CHI 1999*, ACM Press, pp.159–166.

Kahler, H. (2001). *Supporting Collaborative Tailoring*. Ph.D.-Thesis. Roskilde University, Denmark, Roskilde.

Lehman, M. (1980). Programs, life cycles, and laws of software evolution. *IEEE 68*.

Letondal, C. (2001). *Programmation et interaction*. Orsay: Université de Paris XI.

Lieberman, H. (2001). *Your Wish is My Command: Programming by Example*. San Francisco: Morgan Kaufmann.

MacLean, A., Carter, K., Lövstrand, L. and Moran, T. (1990). User-tailorable systems: Pressing the issue with buttons. In: *Proceedings of the Conference on Computer Human Interaction (CHI '90), April 1–5, 1990. Seattle, Washington.* New York: ACM-Press, pp. 175–182.

Majhew, D.J. (1992). *Principles and Guideline in Software User Interface Design*. New York: Prentice Hall.

Mehandjiev, N. and Bottaci, L. (1996). User-enhanceability for organizational information systems through visual programming. In: *Advanced Information Systems Engineering: 8th International Conference*, CAiSE'96, Springer-Verlag.

Mørch, A.I. (1997). Three levels of end-user tailoring: Customization, integration, and extension. In: M. Kyng and L. Mathiassen (eds.), *Computers and Design in Context*. Cambridge, MA: The MIT Press, pp. 51–76.

Mørch, A.I. (2002). Evolutionary growth and control in user tailorable systems. In: N. Patel (ed.), *Adaptive Evolutionary Information Systems*. Hershey, PA: Idea Group Publishing, pp. 30–58.

Mørch, A.I. and Mehandjiev, N.D. (2000). Tailoring as collaboration: The mediating role of multiple representations and application units. *Computer Supported Cooperative Work* **9**(1), 75–100.

Oppermann, R. and Simm, H. (1994). Adaptability: User-initiated individualization. In: R. Oppermann (ed.), *Adaptive User Support—Ergonomic Design of Manually and Automatically Adaptable Software*. Hillsdale, New Jersey: Lawrence Erlbaum Ass.

Paternò, F. (2001). *Model-based Design and Evaluation of Interactive Applications*. London, UK: Springer Verlag.

Paternò, F., Campari, I. and Scopigno, R. (1994). The design and specification of a visual language: An example for customising geographic information systems functionalities. *Computer Graphics Forum* **13**(4), 199–210.

Pipek, V. and Kahler, H. Supporting Collaborative Tailoring, in this volume.

Pipek, V. and Wulf, V. (1999). A groupware's life. In: *Proceedings of the Sixth European Conference on Computer Supported Cooperative Work (ECSCW '99)*, Dordrecht, Kluwer, pp. 199–219.

Repenning, A., Ioannidou, A. and Zola, J. (2000). AgentSheets: End-user programmable simulations. *Journal of Artificial Societies and Social Simulation* **3**(3).

Stiemerling, O. (2000). *Component-Based Tailorability*. Ph.D. Thesis. Department of Computer Science, University of Bonn, Bonn.

Stiemerling, O., Kahler, H. and Wulf, V. (1997). How to make software softer—designing tailorable applications. In: *Proceedings of the ACM Symposium on Designing Interactive Systems (DIS 97), 18.–20.8.1997, Amsterdam (NL)*. New York: ACM-Press, pp. 365–376.

Sutcliffe, A. and Mehandjiev N. (2004). End User Development. *Special Issue of the Communications of the ACM* **47**(9), 31–66.

Won, M., Stiemerling, O. and Wulf, V. Component-based Approaches to Tailorable Systems, in this volume.

Wulf, V. (1999). "Let's see your Search-Tool!"—Collaborative use of tailored artifacts in groupware. In: *Proceedings of GROUP '99*, New York: ACM-Press, pp. 50–60.

Wulf, V. and Golombek, B. (2001). Direct activation: A concept to encourage tailoring activities. *Behaviour and Information Technology* **20**(4), 249–263.