Advanced Visual Systems Supporting Unwitting EUD

Maria Francesca Costabile*, Piero Mussio°, Loredana Parasiliti Provenza°, Antonio Piccinno*

*Dipartimento di Informatica, Università di Bari, ITALY
°DICO, Università di Milano, ITALY
{costabile, piccinno}@di.uniba.it, {mussio, parasiliti}@dico.unimi.it

ABSTRACT

The ever increasing use of interactive software systems and the evolution of the World Wide Web into the so-called Web 2.0 determines the rise of new roles for users, who evolve from information consumers to information producers. The distinction between users and designers becomes fuzzy. Users are increasingly involved in the design and development of the tools they use, thus users and developers are not anymore two mutually exclusive groups of people. In this paper types of users that are between pure end users and software developers are analyzed. Some users take a very active role in shaping software tools to their needs, but they do it without being aware of programming, they are *unwitting* programmers who need appropriate development techniques and environments. A meta-design participatory approach for supporting unwitting end-user development through advanced visual systems is briefly discussed.

Categories and Subject Descriptors

H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces – *Asynchronous interaction, Organizational design, Theory and models*

General Terms

Design, Experimentation, Human Factors, Theory.

Keywords

End user, end-user development, user classification.

1. INTRODUCTION

Users and developers have been traditionally considered two distinct communities: users are the “owners” of the problems and developers are those who implement software systems for supporting users to solve the problems. Nowadays, with the widespread use of software systems and the evolution of the World Wide Web toward the so called Web 2.0, more and more people do not only use software but also get involved in designing and developing software. Such users deploy various pre-programming and programming activities, ranging from simple parameter customization to variation and assembling of components, creating simulations and games. In this paper, we analyze different types of users that are between pure end users and professional software developers. A particular type of end users refers to those that are very active in shaping software tools to their needs without being aware of programming. Similarly to what it is done in [7] for children, we call them *unwitting* programmers and we will explain why in Section 2. We then discuss how the meta-design participatory approach described in [3] [4] is refined for designing advanced visual systems that can support the activities of unwitting programmers.

2. UNWITTING PROGRAMMERS

More and more end users are required to perform various activities that push them to possibly modify and/or create software artefacts in various ways: from simple customization to changing software functionalities. End-User Development (EUD) [6] [11] and End Users Software Engineering [9] are definitions that refer to these activities. Such end users are neither pure end users, nor software professionals: some of them may have certain software development skills but surely they are not interested in software per se. They might develop software to solve specific problems they own [3]. According to the activities they perform with software systems, the following types of users are distinguished:

1. End users who perform simple customization activities, such as changing colours, selecting or creating toolbars to be visualized, selecting items to be shown in a toolbar, etc., in order to adapt the software environment to their habits.
2. End users who write macros to automate some repetitive operations. These possibilities are permitted in spreadsheets.
3. End users who develop web applications, i.e., people with modest levels of experience in Web development, or in computing activities; they have possibly taken a course in HTML and create Web sites. They might be experts in a certain domain, but have poor knowledge in computer science. This type of users has been studied in [8], where an exploratory study of the application and consequences of informal design planning by such users is reported.
4. Developers using domain-specific languages, i.e., professionals in a particular domain (not in computer science) using a specific language to write programs for solving problems of their own domain. Examples are mathematicians, physicists and engineers using MatLab™, but also biologists working with systems like Biok (Biology Interactive Object Kit) [5], which extends formula languages available in spreadsheets and supports tailorability and extensions by end users through an integrated programming environment.
5. Data-intensive researchers who intensively manage stored data and create computer programs, so that they can be considered almost professional software developers, even if they do not have any software engineering background. They use general-purpose

programming languages, such as C and C++, for developing software for their own research goals. They may have attended courses on particular programming languages, but they have no formal training in software engineering. Consequently, they do not pay attention to software qualities such as security, code readability, maintenance, as software engineers usually do. Such end users are called "professional end user developers" in [10], intending with this term people who work in highly technical, knowledge-rich domains and develop software in order to further their professional goals. Examples of these end users are space scientists, research scientists, financial mathematicians.

End users and the activities they usually perform or are willing to perform with computers were previously analyzed in [1] [2] [12] (also mentioned in the introductory chapter of [6]). Two classes of end-user activities were identified, depending on whether their activities imply creating or modifying a software artefact (Class 2) or not (Class 1). More specifically, Class 1 includes activities that allow users to choose among alternative behaviours (or presentations or interaction mechanisms) already available in the application by setting some parameters; such activities are usually called parameterisation or customization or personalization. Class 2 includes all activities that imply some programming in any programming paradigm, thus creating or modifying a software artefact. End users belonging to Class 1 actually correspond to "end users that perform customization" (type 1 in the above classification), while users belonging to Class 2 include the following three categories: end users who write macros (type 2 in the above classification), Web contents developers (type 3), developers using domain-specific languages (type 4).

Fischer and Ye also analyse different types of end users, proposing a spectrum of software-related activities [12]. At the right end of the spectrum, there are the *software professionals*, i.e., software engineers that develop software systems that are used by people other than themselves. At the opposite side (left side) are the *pure end users* that passively use software systems to accomplish their daily task. End users in the middle of the spectrum are "people who have certain development skills but are not interested in software per se. They do not develop software for other people; rather they are developing software to solve specific problems that they own" [12].

We have experience of working with various end users, primarily people who are experts in a specific discipline, such as geology, mechanical engineering, medicine, etc., who are not expert at all in computer science, nor willing to be, and use computer systems for their daily work activities [4]. They do not have any development skill and do not want to be constrained by formalisms unfamiliar to their culture. They want software environments that are easily accessible and usable, that they can "tailor" to their needs, tasks and habits. They do not know what programming means, even if they use software applications that allow them to create or modify software artefacts, but they do this without being aware of programming. In this sense, they are similar to the children analyzed in [7], defined *unwitting end user programmers*. Indeed, children playing with a computer game are required reasonably sophisticated programming, but this is embedded in an intrinsically motivated activity that is perceived as something easy and fun to perform; in their opinion, it cannot be programming, which is generally perceived very difficult.

Petre and Blackwell observe that programming is not the children's goal; playing, constructing and deconstructing is their goal. They also say: "Children have always appropriated, reconfigured and customized their toys. Papert was motivated by such behaviour in the philosophy of 'constructionism' that motivate the Logo programming language" [7]. Children have used Lego, Meccano, and similar toys for doing construction. Computer-based authoring tools allow children to construct interactive simulations, animations, and games, in a manner that places a lot of emphasis on construction; thus they program by construction not by algorithm development, their intention is to play and enjoy, they are not aware of programming. Similarly, adult end users want to manipulate and tailor objects in their software environments in order to create new configurations and designs. They want to do this as part of their own activities that they are highly motivated to perform, not being aware that they are programming.

Going back to the end user classification discussed above, it is evident that programming in an unwitting way is a characteristics of several categories of those end users. Moreover, from the considerations derived in [7], useful indications for the design of software systems for adult unwitted programmers can be extracted. Specifically, children learn and do programming by composition while they tinker or play. They learn by trying things out. Similarly to many adults, when they use a new environment, they do not read tutorials, but go straight to the example or ask friends. Children enjoy a lot communicating with friends and possibly performing collaborative activities, often conducted remotely by exchanging artefacts online. The Software Shaping Workshop (SSW) design methodology we have developed, based on a meta-design approach, includes features that comply with these indications [4]. We will show with a case study how it permits the design of advanced visual systems that support unwitting end-user development.

3. A META-DESIGN APPROACH

The aim of the *Software Shaping Workshop* (SSW) methodology presented in [1] [3] [4] is to create software environments that support end users in performing their activities of interest, but also allow them to tailor these environments to better adapt them to their needs, and even to create or modify software artifacts. The latter are defined as activities of End-User Development (EUD) [6] [11]. To permit EUD activities, we have to consider a two-phase process, the first phase being designing the design environment (meta-design phase), the second one being designing the applications using the design environment. In this way, we distinguish between the design (or "shaping") work that is done by different types of end users (second phase), and the process of creating suitable environments and tools that can be applied by end users for their own design/shaping purposes.

Traditionally, the life cycle of interactive software system distinguishes between design time and use time. At *design time*, system developers (with or without user participation) create environments and tools for the *world as imagined* by them to anticipate users' needs and objectives. At *use time*, users use the system in the *world as experienced* [12]. Existing design frameworks are based on the assumption that major design activities end at a certain point after which the system enters use time. Our approach bridges these two stages into a unique *"design-in-use" continuum* that permits the creation of *open and continuously evolvable systems*, which can be collaboratively extended and redesigned at use time *by users and user communities*. Moreover, meta-designed software systems not only provide the technical means for users to customize and extend the systems but also provide social and technical mechanisms to facilitate user participation and collaboration during the design activities.

Participatory Design is an approach originated in Scandinavian, which consists of the participation of end users in the design process, not only as an experimental subject but as active members of the design team. In this way, end users are not passive participant whose involvement is entirely governed be the designer. The rationale is that users are expert in the work context and a design can be affective if these experts are allowed to actively contribute to the design. Moreover, the adoption of a new system is liable to change the work context and organizational processes and it can only be accepted if these changes are acceptable to the user. The traditional Scandinavian approach stresses the involvement of end users in the design process, but it does not indicate when this involvement stops. The SSW methodology adopts a Participatory Design approach that it is not over with the release of the software, but continues throughout the whole software life cycle. Users are provided with software environments to perform design activities even at run-time.

The basic idea of this methodology is that an interactive system must be designed as a network of different software environments (also called Software Shaping Workshops), each one specific for a community of users of that system. Each workshop allows its users to interact through a visual language familiar to their culture and skills, since it reflects and empowers the users' traditional notations and system of signs. Such workshops enable domain experts, as well as HCI experts, to create and modify software artefacts by direct manipulation of visual elements.

We briefly refer to a case study to better explain our approach. The case arises from the collaboration with CIDD ("Consorzio Italiano Distribuzione Dolciaria"), a consortium of about thirty small and medium Italian companies and about twenty big partner companies, operating in the confectionery field. The consortium's main business is the confectionery distribution in the whole Italy. It provides its associated companies with a number of services, such as price lists, discounts, order management, etc. We collaborated with CIDD in order to create its Web portal of services. In CIDD, there is a chairman who is the person formally responsible for the CIDD activities according to the consortium statute. The main role with reference to the portal is played by the sales manager, who manages all the consortium activities and defines the services for the associated companies. Such companies purchase products from the partner companies and, through the portal, communicate with their customers.

The sales manager wants to tailor the software environments to be used by the associate companies, maintaining a consistent style of documentation and interaction. On its side, each company wants to define the environments to be used by their customers. This is a typical case in which the meta-design approach of the SSW methodology can be successful. The Web portal is then developed as a network of different workshops, each one specific for a community of users, where users can perform the EUD activities they require. The sales manager will work in a meta-environment

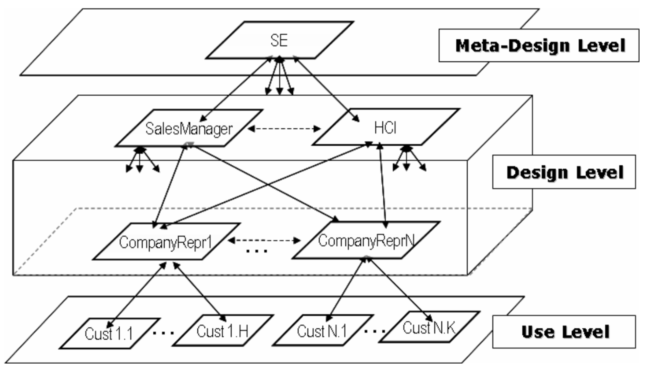

Figure 1. The SSW network in the case of CIDD.

through which he can design the environments to be used by the associated companies, and the companies' representatives will use their environments to tailor the tools to be used by their customers.

During the field study we carried out for requirements analysis, we identified five different types of users based on the roles they have in the consortium:

– *power users*: they are able to see, modify and delete portal contents, define access rules to the portal and even design workshops for the CIDD companies; the role of the power user is played by the sales manager and his secretary, who works on his behalf;
– *associated companies*: their representatives can access contracts, catalogues, promotions, competitions, place orders and design/tailor the workshops for their customers;
– *registered guests*: they are the customers of the associated companies and can see some contents according to the access rules defined by the power user;
– *Unregistered guests*: any user who can see the portal home page when browsing on the web.

The portal allows its users to access and exchange information cooperating through the Web according to the different access rules in force. It is evident that there is a variety of users with different roles and accesses. They are experts in a specific discipline, but not in computer science. They need to use the portal for performing their work tasks, but they are not and do not want to become computer scientists. When the power users, or the representatives of associated companies, modify and update the CIDD portal, they actually program, but they are not aware of this, also because they do not use conventional programming that would be too unfamiliar to their culture and skills, but they compose new software artefacts by construction, similarly to the children's program construction described in [7]. CIDD users work with the available tools through direct manipulation techniques, so that they do not perceive they are creating or modifying software artefacts, they are simply carrying out their work activities. In other words, they are unwitting end-user developers.

The variety of roles and accesses requires different workshops for accessing and managing the different information. The identified SSW network in the case study of CIDD is represented in Figure 1. At the meta-design level (the top level) there is a SSW ("SE" in the figure) used by the software engineers to shape the tools to be

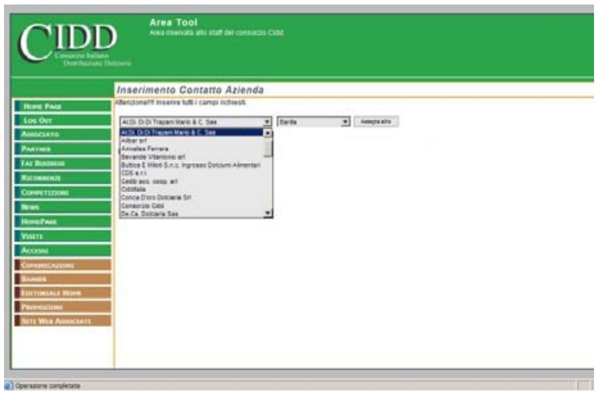

Figure 2. A screen shot of the SalesManager workshop in which is generating services for associated companies

used at the other levels and to participate in the design, implementation, and validation activities of all the SSWs in the network. At the design level, there is a workshop for HCI experts, a workshop for the sales manager and a number of workshops for representatives of associated companies (“AssocRep1”, ..., “AssocRepN” in Figure 1). The latter are used by representatives of associate companies to create and modify the application workshops devoted to their customers (“Cust1.1”,..., “Cust1.H”,.., “CustN.1”,..., “CustN.K”). At the use level, there are workshops used by company customers (“Cust1.1”.. “Cust1.H”, .., “CustN.1”,.. “CustN.K”) for their consortium activities. On the whole, both meta-design and design levels include all the SSWs that support the design team in their activities of participatory design.

In the *SalesManager* workshop, the sales manager finds tools that allow him to design the system workshops for associate company representatives (“AssocRep1”,..., “AssocRepN” in Figure 1). This workshop is shown in Figure 2. Let us suppose that the sales manager wants to design the system workshop to be used by the representatives of an associated company, providing it with some services. He designs this workshop by direct manipulation. Specifically, he selects the company from a drop-down list available in the central area of his workshop (the list is shown in the central area in Figure 2); he also selects a service he wants to provide from another drop-down list (available on the right of the previous one, not open in Figure 2) and clicks on the “Assign” button (the latter on the right in Figure 2) to actually define that service for that company workshop. He does this for all services he wants to provide. The result of this activity is the workshop for representatives of the associated company, presenting the assigned services, that lack of space prevents us to show.

4. CONCLUSIONS

This paper has discussed the actors involved in system development, ranging from pure end users to software professionals. We propose a framework for classifying the different actors, focusing primarily on those that are very active in shaping software tools, without being aware of programming. Such unwitting programmers require new software environments, new tools, new programming approaches able to comply with their needs. The meta-design approach briefly outlined in the paper is able to provide software environments to support unwitting programmers.

5. ACKNOWLEDGMENTS

This work was supported by the Italian MIUR and by EU and Regione Puglia under grant DIPIS. We thank the CIDD consortium and Nicola Claudio Cellamare for their collaboration related to development of the CIDD portal.

6. REFERENCES

[1] Costabile, M. F., Fogli, D., Fresta, G., Mussio, P., and Piccinno, A. 2003. Building environments for end-user development and tailoring. Proc. HCC 2003 (Auckland, New Zealand October 28-31, 2003). 31-38.

[2] Costabile, M.F., Fogli, D., Letondal, C., Mussio, P., and Piccinno, A. 2003. Domain-Expert Users and their Needs of Software Development. Proc. HCII 2003 (Crete, Greece, June 22-27, 2003). Lawrence Erlbaum Associates, 532-536.

[3] Costabile, M.F., Fogli, D., Mussio, P., and Piccinno, A. 2006 End-User Development: the Software Shaping Workshop Approach, in Lieberman, H., Paternò, F., and Wulf, V. (Eds), End User Development. Springer, Dordrecht, The Netherlands, 183-205.

[4] Costabile, M.F., Fogli, D., Mussio, P., and Piccinno, A. 2007. Visual Interactive Systems for End-User Development: a Model-based Design Methodology. IEEE Trans. on SMC - Part A: Systems and Humans 37, 6, 1029 - 1046.

[5] Letondal, C. 2006. Participatory Programming: Developing Programmable Bioinformatics Tools for End-Users, in Lieberman, H., Paternò, F., and Wulf, V. (Eds), End User Development. Springer, Dordrecht, The Netherlands, 207-242.

[6] Lieberman, H., Paternò, F., and Wulf, V. (Eds) 2006. End User Development. Springer, Dordrecht, The Netherlands.

[7] Petre, M. and Blackwell, A. F. 2007. Children as Unwitting End-User Programmers. Proc. VL/HCC 2007 (Coeur d'Alène, USA, Sep. 23-27, 2007). 239-242.

[8] Rosson, M. B., Sinha, H., Bhattacharya, M., and Zhao, D. 2007. Design Planning in End-User Web Development. Proc. VL/HCC 2007 (Coeur d'Alène, USA, Sep. 23-27, 2007). 189-196.

[9] Scaffidi, C., Shaw, M., and Myers, B. (Eds) 2005. Proc. 1st workshop on End-user software engineering. (St. Louis, Missouri, May 21 - 21, 2005), 1-5.

[10] Segal, J. 2007. Some Problems of Professional End User Developers. Proc. VL/HCC 2007 (Coeur d'Alène, Idaho, USA, Sep. 23-27, 2007). 111-118.

[11] Sutcliffe, A., Mehandjiev, M. (Eds.) Special Issue on End-User Development. CACM, 47, 9, 31-66.

[12] Ye, Y. and Fischer, G. 2007. Designing for Participation in Socio-Technical Software Systems. Proc. HCII 2007 (Beijing, China, Jul. 22-27, 2007). LNCS,Springer, 312-321.