Domain-Expert Users and their Needs of Software Development[1]

M.F. Costabile°, D. Fogli, C. Letondal+, P. Mussio*, A. Piccinno°*

° DI - Università di Bari
Via Orabona 4
Bari, Italy
[costabile, piccinno]@di.uniba.it

* DEA - Università di Brescia
Via Branze 38
Brescia, Italy
[fogli, mussio]@ing.unibs.it

+ Pasteur Institute
25, rue du Dr Roux
Paris, France
letondal@pasteur.fr

Abstract

There are several categories of end-users of computer systems, depending on their culture, skills, and types of tasks they perform. This paper recognizes the problem of 'user diversity' even among people of the same technical or scientific tradition, and focuses on the study of a specific category of end-users, that we call domain-expert users: they are professionals in some domain different from computer science, who need to use computers in their daily work. We analyse the activities they usually perform or are willing to perform with computers and we identify their real needs of carrying out activities that result in the creation or modification of software artefacts.

1 Introduction

The ever increasing spread of computer environments in the information society is determining a continuous growth of the end-users population as broad as possible. Such end-users have different needs and backgrounds, and operate in different contexts. The following definition of end-users is given in (Cypher, 1993): "A user of an application program. Typically, the term means that the person is not a computer programmer. A person who uses a computer as part of daily life or daily work, but is not interested in computers per se." It is evident that several categories of end-users can be defined, for instance depending on whether the computer system is used for work, for personal use, for pleasure, for overcoming possible disabilities, etc.

Brancheau and Brown analyse the status of what they call *end-user computing* and define it as "… the adoption and use of information technology by people outside the information system department, to develop software applications in support of organizational tasks" (Brancheau & Brown, 1993). In this survey, they primarily analysed the needs of users that are experts in a specific discipline, but not in computer science. In our experience, we have often worked with end-users that are experts in their field, that need to use computer systems for performing their work tasks, but that are not and do not want to become computer scientists. This has motivated the definition of a particular class of end-users, that we call *domain-expert users* (*d-expert* in the following): they are experts in a specific domain, not necessarily experts in computer science, who use computer environments to perform their daily tasks. They have also the responsibility for induced errors and mistakes. In this paper, we focus on such users and analyse the activities they usually perform or are willing to perform with computers. This analysis shows that d-experts have real needs of performing some programming activities that result in the creation or modification of software artefacts.

In the rest of this paper, Section 2 describes some features of d-experts. Section 3 reports about activities d-experts have the need to perform. Section 4 describes real situations in which programming needs occur and Section 5 concludes the paper.

[1] The support of EUD-Net Thematic Network, sponsored by EC, project IST-2001-37470, is acknowledged.

2 Characterizing domain-expert users

In scientific and technological domains, d-experts communicate with each other through documents, expressed in some notations, which represent abstract or concrete concepts, prescriptions, and results of activities. Recognizing users as d-experts means recognizing the importance of their notations and dialects as reasoning and communication tools. As designers, we are challenged to create virtual environments, in which users interact using a formal representation of their traditional languages and with virtual tools that recall the real ones with which users are familiar. This is a hard challenge, because d-expert communities develop in time from the experience different notations, which reflect the different technical methods, languages, goals, tasks, ways of thinking, and documentation styles.

Often, dialects arise in a community, because the notation is applied in different practical situations and environments. For example, mechanical drawings are organized according to standard rules, which are different in Europe and in USA. Explicative annotations are written in different national languages. Often the whole document (drawing and text) is organized according to guidelines developed in each single company. The correct and complete understanding of a technical drawing depends on the recognition of the original standard as well as on the understanding of the national (and also company developed) dialects. Similar cases are quite common: d-experts of a same community constitute different sub-communities depending not only on user skill, culture, knowledge, but also on specific abilities (physical/cognitive), tasks, and context. Recognizing the diversity of users calls for the ability to represent a meaning of a concept with different materialization, e.g. text, images or sound, and to associate to a same materialization a different meaning according, e.g., to the context of interaction.

An important phenomenon, often observed in Human-Computer Interaction (HCI), is that "using the system changes the users, and as they change they will use the system in new ways" (Nielsen, 1993). In turn, the designer must evolve the system to adapt it to its new usages; we called this phenomenon *co-evolution of users and systems* (Arondi et al., 2002). In (Bourguin et al., 2001) it is observed that these new uses of the system determine the evolution of the user culture and of her/his models and procedures of task evolution, while the requests from users force the evolution of the whole technology supporting interaction.

Co-evolution stems from two main sources: a) user creativity: the users may devise novel ways to exploit the system in order to satisfy some needs not considered in the specification and design phase; and b) user acquired habits: users may follow some interaction strategy to which they are (or become) accustomed; this strategy must be facilitated with respect to the initial design.

3 Activities of domain-expert users

When working with a software application, d-experts feel the need to perform various activities that may even lead to the creation or modification of software artefacts, in order to get a better support to their specific tasks, thus being considered activities of End-User Development (EUD) in accordance with the following definition: "EUD is a set of activities or techniques that allow people, who are not professional developers, at any stage to create or modify a software artefacts for their own or shared use" (EUD-Net, 2002). The need of EUD is a consequence of user diversity and user evolution discussed in the previous section. Within EUD, we may include various tailoring activities described in the literature, and reported in the following.

3.1 Tailoring activities

Tailoring activities are defined in different ways in the literature; they include adaptation, customization, end-user modification, extension, personalization, etc. These definitions partly overlap with respect to the phenomena they refer to, while often the same concepts are used to refer to different phenomena.

In (Wulf, 1999), tailorability is defined as the possibility of changing aspects of an application's functionality, during the use of an application, in a persistent way, by means of tailored artefacts; the changes may be performed by users that are local experts. Tailorability is very much related to adaptability. In (Trigg et al., 1987), a system is adaptable if an end-user "produces new system behaviour without help from programmers or designers". There are four levels for being adaptable: 1) *flexible* - objects and behaviours can be interpreted and used differently; 2) *parameterizable* - alternative behaviours can be chosen by the user; 3) *integrable* - the system can be integrated with other components, internal or external, 4) *tailorable* - users are allowed to change the system itself by building accelerators, specializing behaviour, or adding new functionality. Thus, tailoring involves the creation of new functionalities by end-users.

In (Mackay, 1991) and in (Nardi, 1993) empirical studies are reported on activities performed by end-users, and generally defined as tailoring activities. Mackay analyses how users of a UNIX software environment try to customise the system, intending as customisation the possibility of modifying software to make persistent changes. She finds that many users do not customise their applications as much as they could. This also depends on the fact that it takes too much time and deviates from other activities. Nardi conducted empirical studies on users of spreadsheets and CAD software. She found out that those users actually perform activities of end user programming, thus generating new software artefacts; these users are even able to master the formal languages embedded in these applications when they have a real motivation for doing so.

Mørch specifies three main categories of tailoring: customisation, integration, and extension (Mørch, 1997). *Customisation* usually consists of a set of preferences configurable by the user through a preference form, in which a user can set parameters for the various configuration options the application supports. *Integration* goes beyond customization by allowing users to add new functionality to an application, without accessing the underlying implementation code. Instead, users tailor an application by linking together predefined components within or across the application. *Extension* refers to the case in which the application doesn't provide, by itself or by its components, any functionality that accomplishes a specific user need, thus adding a new functionality generates a radical change in the software. In the above categorization, there are instances that cut across categorical boundaries.

3.2 Two classes of domain-expert activities

The brief overview given in the previous section shows that different meanings are associated to tailorability and adaptability. To avoid ambiguity, we propose two classes of d-expert activities:

Class 1. It includes activities that allow users, by setting some parameters, to choose among alternative behaviours (or presentations or interaction mechanisms) already available in the application; such activities are usually called parameterisation or customization or personalization.

Class 2. It includes all activities that imply some programming in any programming paradigm, thus creating or modifying a software artefact. Since we want to be as close as possible to the human, we will usually consider novel programming paradigms, such as programming by demonstration, programming with examples, visual programming, macro generation.

In the following, we provide examples of activities of both classes from experiences of participatory design workshops in two domains, biology and earth science.

Activities belonging to Class 1 are:

- *Parameterization*. It is intended as specification of unanticipated constraints in data analysis. In this situation, observed very often, the d-expert wishes to guide a computer program by indicating how to handle several parts of the data in a different way; the difference can just lay in associating specific computation parameters to specific parts of the data, or in using different models of computations available in the program. In biology, this is related to protocol design.

- *Annotation*. D-experts often write comments next to data and result files in order to remember what they did, how they obtained their results, and how they could reproduce them.

The following activities belong to Class 2:

- *Modelling from the data*. The system supporting the d-expert derives some (formal) models from observing data, e.g. in (Blackwell, 2000) a kind of regular expression is inferred from selected parts of aligned sequences, or in (Arondi et al., 2002) patterns of interactions are derived.
- *Programming by demonstration*. D-experts show examples of properties occurrences in the data and the system infers from them a (visual) function.
- *Formula languages*. This is available in spreadsheets and could be extended to other environments, such as Biok (Biology Interactive Object Kit) that is a programmable application for biologists (Letondal, 2001). The purpose of Biok is twofold: to analyze biological data such as DNA, protein sequences or multiple alignments, and to support tailorability and extensions by the end-user through an integrated programming environment.
- *Indirect interaction with application objects***.** As opposed to direct manipulation, a command language can be provided to script objects.
- *Incremental programming*. It 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.
- *Extended Annotation*. A new functionality is associated with the annotated data. This functionality can be defined by any technique previously described.

4 Examples of EUD applications

In this section we describe some situations that show the real need of environments with EUD capabilities, as emerged in our work with biologists and earth scientists.

Experience acquired at the Pasteur Institute during several years indicates that in the field of biology software for academic research there are two types of software development: 1) large scale projects, developed in important bioinformatics centres, such as the European Bioinformatics Institute; 2) local development, by biologists who know some programming language, in order to deal with daily tasks, such as managing data, analysing results, or testing scientific ideas. We are here interested in the second type of development, since it can be considered end-user development. Moreover, it is worth mentioning that many biologists do not know anything about programming, and yet feel the need of modifying the application they use to better fit their needs. Below is a list of real programming situation examples, drawn from interviews with biologists, news forum, or technical support at the Pasteur Institute. These situations occurred when working with molecular sequences, i.e., either DNA or protein sequences: *scripting*, i.e. search for a sequence pattern, then retrieve all the corresponding secondary structures in a database; *parsing*, i.e. search for the best match in a database similarity search report but relative to each subsection; *formatting*, i.e. renumber one's sequence positions from -3000 to +500 instead of 0 to 3500; *variation*, i.e. search for patterns in a sequence, except repeated ones; *finer control on the computation*, i.e. control in what order multiple sequences are compared and aligned (sequences are called aligned when, after being compared, putative corresponding bases or amino-acid letters are put together); *simple operations*, i.e. search in a DNA sequence for some characters.

Considering the domain of earth science, we worked with scientists and technicians who analyse satellite images and produce documents such as thematic maps and reports, which include photographs, graphs, etc., and textual or numeric data related to the environmental phenomena of interest. Two sub-communities of d-experts are: 1) photo-interpreters who classify, interpret, and annotate remote sensed data of glaciers; 2) service oriented clerks, who organize the interpreted images into documents to be delivered to different communities of clients. Photo-interpreters and clerks share environmental data archives, some models for their interpretation, some notations for their presentation, but also have to achieve different tasks, documented through different sub-notations and tools. Therefore, their notations can be considered two dialects of the Earth Scientist & Technologist general notation.

5 Conclusions

In this paper, we have focused on a specific category of end-users, called domain-expert users, and have analysed the activities they usually perform with computers as well as the activities they would like to perform in order to get a better support from computer systems to their daily work. The two examples in the previous section make the needs of d-experts emerge. In the case of biologists unpredictable needs may arise at any time, which require some kind of programming, even if such programming is rather simple most of the time. However, existing software does not usually provide any programming capability. Thus, the biologists have often to program everything from scratch, which is usually very difficult for them. In the case of earth science, photo-interpreters need software tools for interactive image processing and extended annotation. They interactively identify and classify parts of glacier images and associate to them an extended annotation, constituted by a textual part and a program created by selecting some computations on the basis of the observed data. In this way, photo-interpreters create new software artefacts, which are managed through user-defined widgets. Clerks will use these widgets to interact with the programs made available by the photo-interpreters to produce the required documents.
Thus, it is a challenge for us, as designers of computer systems more accessible to their end-users, to develop programming paradigms and software environments that are adequate to the needs of end-users.

6 References

Arondi, S., Baroni, P., Fogli, D., Mussio, P. (2002). Supporting co-evolution of users and systems by the recognition of Interaction Patterns. *Proc. of AVI 2002*, Trento (I), May 2002, 177-189.

Blackwell, A. (2000). See What You Need: Toward a visual Perl for end users. *Proc. of Workshop on visual languages for end-user and domain-specific programming*, September 10, 2000, Seattle, WA, USA.

Bourguin, G., Derycke, A., & Tarby, J.C. (2001). Beyond the Interface: Co-evolution inside Interactive Systems - A Proposal Founded on Activity Theory. *Proc. of IHM-HCI*.

Brancheau & Brown. (1993). The Management of End-User Computing: Status and Directions. *ACM Computing Surveys*, 25 (4), 437-482.

Cypher, A. (1993). Watch What I Do: Programming by Demonstration. The MIT Press, Cambridge.

EUD-Net. (2002). http://giove.cnuce.cnr.it/EUD-NET/pisa.htm

Letondal, C. (2001). Programmation et interaction. PhD thesis, Université de Paris XI, Orsay.

Mackay, W.E. (1991). Triggers and Barriers to Customizing Software. *Proc. CHI'90 Human Factors in Computing Systems*, New Orleans, Apr. 27 – May 2, 153-160. ACM Press.

Mørch, A. (1997). Three Levels of End-User Tailoring: Customization, Integration, and Extension. In M. Kyng & L. Mathiassen (eds.), *Computers and Design in Context*, (51-76). The MIT Press, Cambridge.

Nardi,B.(1993). A small matter of programming: perspectives on end user computing.MIT Press, Cambridge.

J. Nielsen. (1993). Usability Engineering. Academic Press, San Diego.

Trigg, R.H., Moran, T.P., & Halasz, F.G. (1987). Adaptibility and Tailorability in NoteCards. *Proc. of INTERACT '87*, Stuttgart, 723-728. Elsevier Science Publishers.

Wulf, V. (1999). “Let's see your Search-Tool!” - Collaborative use of Tailored Artifacts in Groupware. *Proc. of GROUP '99*, Phoenix, USA, Nov.14-17, 50-60. ACM-Press, New York.