Chapter 9

End-User Development: The Software Shaping Workshop Approach

MARIA FRANCESCA COSTABILE[1], DANIELA FOGLI[2], PIERO MUSSIO[3] and ANTONIO PICCINNO[4]
[1]*Dipartimento di Informatica, Università di Bari, Bari, Italy, costabile@di.uniba.it*
[2]*Dipartimento di Elettronica per l'Automazione, Università di Brescia, Brescia, Italy, fogli@ing.unibs.it*
[3]*Dipartimento di Informatica e Comunicazione, Università di Milano, Milano, Italy, mussio@dico.unimi.it*
[4]*Dipartimento di Informatica, Università di Bari, Bari, Italy, piccinno@di.uniba.it*

Abstract. In the Information Society, end-users keep increasing very fast in number, as well as in their demand with respect to the activities they would like to perform with computer environments, without being obliged to become computer specialists. There is a great request to provide end-users with powerful and flexible environments, tailorable to the culture, skills, and needs of a very diverse end-user population. In this chapter, we discuss a framework for End-User Development and present our methodology for designing software environments that support the activities of a particular class of end-users, called domain-expert users, with the objective of making their work with the computer easier. Such environments are called Software Shaping Workshops, in analogy to artisan workshops: they provide users only with the necessary tools that allow them to accomplish their specific activities by properly shaping software artifacts without being lost in virtual space.

Key words. end-user development, domain expert, user diversity, gain, co-evolution, implicit information, tacit knowledge, user notation, HCI model.

1. Introduction

In the Information Society, new computer technologies have created the potential to overcome the traditional division between users and the individuals responsible for developing, operating, and maintaining systems. Organizational, business, and commercial technologies increasingly require information technologies to be placed directly in the hands of technicians, clerks, analysts, and managers (Brancheau and Brown, 1993). Cypher (1993) defines end-users as people who use a computer application as part of their daily life or daily work, but are 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. The end-user population is not uniform, but divided in non-mutually exclusive communities characterized by different goals, tasks, and activities. Even these communities cannot be considered uniform, because they include people with different cultural, educational, training, and employment background, who are

Henry Lieberman et al. (eds.), End User Development, 183–205.

novices or experts in the use of the computer, the very young and the elderly and those with different types of (dis)abilities. End-users operate in various interactive contexts and scenarios of use, they want to exploit computer systems to improve their work, but they often complain about the difficulties in the use of such systems.

Brancheau and Brown (1993) describe *end-user computing* as ". . . the adoption and use of information technology by people outside the information system department, to develop software applications in support of organizational tasks". The organization in which such people work requires them to perform end-user computing and to assume the responsibility of the results of this activity. In (Brancheau and Brown, 1993), the authors primarily analyze the needs of users who are experts in a specific discipline, but not in computer science. Our experience is focused on this kind of user, such as medical doctors, mechanical engineers, geologists, etc. This has motivated our definition of a particular class of end-users, that we call *domain-expert users* (or *d-experts* for short): they are experts in a specific domain, not necessarily experts in computer science, who use computer environments to perform their daily tasks. In the literature, other authors address the needs of *domain experts* (Borchers, 2001; Fischer et al., 2001). Such end-users have the responsibility for possible errors and mistakes, even those generated by wrong or inappropriate use of the software.

Indeed, one fundamental challenge for the next few years is to develop environments that allow people without a particular background in programming to develop and tailor their own applications, still maintaining the congruence within the different evolved instances of the system. Over the next few years, we will be moving from *easy-to-use* to *easy-to-develop-and-tailor* interactive software systems. We foresee the active participation of end-users in the software development process. In this perspective, tasks that are traditionally performed by professional software developers are transferred to the users, who need to be specifically supported in performing these tasks. Active user participation in the software development process can range from providing information about requirements, use cases and tasks, including participatory design (Schuler and Namioka, 1993), to end-user computing (Nardi, 1993). Companies producing software for the mass market are slowly moving in this direction; examples are the adaptive menus in MS Word™ or some programming-by-example techniques in MS Excel™. However, we are still a long way from their systematic adoption.

In this chapter, we first analyze the activities domain-expert users usually perform or are willing to perform with computers. These people reason and communicate with each other through documents, expressed by notations, which represent abstract or concrete concepts, prescriptions, and results of activities. Often, dialects arise in a community, because the notation is used in different practical situations and environments. For example, mechanical drawings are organized according to rules, which are different in Europe and in the USA. D-experts often complain about the systems they use, they feel frustrated because of the difficulties they encounter interacting with them. Moreover, d-experts feel the need to perform various activities that may even lead to the creation or modification of software artifacts, in order to obtain a better support for their specific tasks, which are therefore considered End-User Development (EUD) activities. Indeed the definition provided by EUD-Net says that "EUD is 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 or modify a software artifact" (EUD-net Thematic Network).

In this chapter we discuss a framework for EUD based on Software Shaping Workshops (SSWs), which are software environments that aim at supporting the activities of domain-expert users, with the objective of easing the way these users work with computers. In this framework, d-experts play two main roles: (1) performing their working tasks, and (2) participating in the development of the workshops, as representatives of the workshop users. As we explain in Section 5, in both roles d-experts perform EUD activities but are required neither to write codes, nor to know any programming language. D-experts interact with the system through visual languages, computerized versions of their traditional languages and tools. Thus, they can program with the feeling of manipulating the objects of interest in a way similar to what they do in the real world.

The chapter is organized as follows. Section 2 discusses the major reasons behind the difficulties in Human–Computer Interaction (HCI). Section 3 proposes a classification of EUD activities that domain-expert users need to perform. SSWs are then presented in Section 4: they aim at supporting users in their interaction with computers and in performing EUD activities. To provide an example of how d-experts work with SSWs, a case study in a medical domain is presented in Section 5. Section 6 reports a comparison with related works. Finally, Section 7 concludes the chapter.

2. Phenomena Affecting the Human–Computer Interaction Process

Several phenomena affecting the HCI process have emerged in the use of interactive systems. They have been observed, studied, and reported in the current literature, often separately and from different points of view, typically from the points of view of Usability Engineering, Software Engineering, and Information System Development. We present them from a unified, systemic point of view, framing them in the model of HCI which we have developed within the Pictorial Computing Laboratory (PCL) (Bottoni et al., 1999). Our aim is to understand their influence on the HCI process and to derive an approach for system design and development, which tries to overcome the hurdles these phenomena create and to exploit the possibilities they offer.

2.1. A MODEL OF THE HCI PROCESS

In this chapter, we capitalize on the model of the HCI process and on the theory of visual sentences developed by the PCL (Bottoni et al., 1999). In the PCL approach, HCI is modeled as a *syndetic* process (Barnard et al., 2000), i.e., a process in which systems of different nature (the cognitive human—the "mechanical" machine) cooperate to achieve a task. From this point of view, HCI is a process in which the user and the computer communicate by materializing and interpreting a sequence of messages at successive instants in time. If we only consider WIMP (Windows, Icons, Menus, Pointers) interaction (Dix et al., 1998), the messages exchanged are the whole images which appear on the screen display of a computer and include text, icons, graphs, and

pictures. Two interpretations of each image on the screen and of each action arise in the interaction: one performed by the user performing the task, depending on his/her role in the task, as well as on his/her culture, experience and skills and the second internal to the system, associating the image with a computational meaning, as determined by the programs implemented in the system (Bottoni et al., 1999). From this point of view, the PCL model reflects a "computer semiotics" approach (Andersen, 2001), in that it "analyzes computer systems and their context of use under a specific perspective, namely as signs that users interpret to mean something" (Andersen, 1992). The HCI process is viewed as a sequence of cycles: the human detects the image on the screen, derives the message meaning, decides what to do next and manifests his/her intention by an activity performed by operating on the input devices of the system; the system perceives these operations as a stream of input events, interprets them with reference to the image on the screen, computes the response to human activity and materializes the results on the screen, so that they can be perceived and interpreted by the human. In theory, this cycle is repeated until the human decides that the process has to be stopped, either because the task has been achieved or has failed.

2.2. THE PHENOMENA

In our opinion, the major phenomena that affect the HCI process are: the communicational gap between designers and users (Majhew, 1992); the grain induced by tools (Dix et al., 1998); the co-evolution of system and users (Arondi et al., 2002; Bourguin et al., 2001; Carroll and Rosson, 1992; Nielsen, 1993); the availability of implicit information (Mussio, 2004) and tacit knowledge (Polanyi, 1967).

- *Communicational gap between designers and users.* The PCL model highlights the existence of two interpretations of each image on the screen and of each action performed to modify it. The first interpretation is performed by the user, the second by the system. The interpretation performed by the system reflects the designer understanding of the task considered, implemented in the programs that control the machine. Between designers and users there is a communicational gap due to their different cultural backgrounds. They adopt different approaches to abstraction, since, for instance, they may have different notions about the details that can be abridged. Moreover, users reason heuristically rather than algorithmically, using examples and analogies rather than deductive abstract tools, documenting activities, prescriptions, and results through their own developed notations, articulating their activities according to their traditional tools rather than computerized ones. On the whole, users and designers possess distinct types of knowledge and follow different approaches and reasoning strategies to modeling, performing, and documenting the tasks to be carried out in a given application domain. Interactive systems usually reflect the culture, skill, and articulatory abilities of the designers. Users often find hurdles in mapping features of the interactive system into their specific culture, skill, and articulatory abilities.
- *Grain*. Every tool is suited to specific strategies in performing a given task. Users are induced by the tool to follow strategies that are apparently easily executable,

but that may be non-optimal. This is called "grain" in (Dix et al., 1998), i.e., the tendency to push the users towards certain behaviors. Interactive systems tend to impose their grain on users' resolution strategies, a grain often not amenable to the users' reasoning and possibly even misleading for them.

– *User diversity*. As highlighted in the introduction, users do not belong to a uniform population, but constitute communities, characterized by different cultures, goals, and tasks. As a consequence, specialized user dialects grow in each user community, which develop particular abilities, knowledge, and notations. User diversity arises even within the same community, depending not only on user skill, culture, and knowledge, but also on specific abilities (physical and/or cognitive), tasks, and the context of the activity. If, during system design, this phenomenon is not taken into account, some users may be forced to adopt specific dialects related with the domain, but different from their own and possibly not fully understandable, thus making the interaction process difficult.
– *Co-evolution of systems and users*. It is well known that "using the system changes the users, and as they change they will use the system in new ways" (Nielsen, 1993). These new uses of the system make the working environment and organization evolve and force the designers to adapt the system to the evolved user, organization, and environment (Bourguin et al., 2001). This phenomenon is called co-evolution of system, environment, and users. Designers are traditionally in charge of managing the evolution of the system. This activity is made more difficult by the communicational gap.
– *Implicit information.* When adopting user defined notations, a relevant part of the information carried by the system is embedded in its visual organization and shape materialization. We call this part of the information carried by the system 'implicit information'. For example, in the documents of scientific communities, the use of bold characters and specific styles indicates the parts of the documents—paper title, abstract, section titles—which synthesize its meaning (Borchers, 2001). Strips of images, for example illustrating procedures or sequences of actions to be performed, are organized according to the reading habits of the expected reader: from left to right for Western readers, from right to left for Eastern ones. Furthermore, some icons, textual words, or images may be meaningful only to the experts in some discipline: for example, icons representing cells in a liver simulation may have a specific meaning only for hepatologists (Mussio et al., 1991), while a X-ray may be meaningful to physicians but not to other experts.
– *Tacit knowledge*. Implicit information is significant only to users who possess the knowledge to interpret it. Most of this knowledge is not explicit and codified but is tacit, namely it is knowledge that users possess and currently use to carry out tasks and to solve problems, but that they are unable to express in verbal terms and that they may even be unaware of. It is a common experience that in many application fields users exploit mainly their tacit knowledge, since they are often more able to do than to explain what they do. Tacit knowledge is related to the specific work domain and it is also exploited by users to interpret the messages from the software system. User notations let users exploit their tacit knowledge

and allow the system constructed in these notations to incorporate it as a part of the implicit information.

2.3. SOME OBSERVATIONS CONCERNING THE USER

When the system imposes task execution strategies, which are alien to users, it becomes a *fence* that forces users to follow unfamiliar reasoning strategies and to adopt inefficient procedures. In order to design a system that meets users' needs and expectations, we must take into account the following observations:

1. The notations developed by the user communities from their working practice are not defined according to computer science formalisms, but they are concrete and situated in the specific context, in that they are based on icons, symbols, and words that resemble and schematize the tools and the entities which are to be used in the working environment. Such notations emerge from users' practical experience in their specific activity domain. They highlight the kind of information users consider important for achieving their tasks, even at the expense of obscuring other kinds and facilitate the problem solving strategies, adopted in the specific user community.
2. Software systems are in general designed without taking explicitly into account the problem of implicit information, user articulatory skills, and tacit knowledge. The systems produced can therefore be interpreted with high cognitive costs.
3. Implicit information and tacit knowledge need an externalizing process, which translates them into a form intelligible to a computer system. Implicit information must be conveyed by the layout and appearance of the systems, in order to be exploited by users in performing their work. The final aim is the creation of interactive software systems that the users may correctly perceive and work with.
4. A system acceptable to its users should have a gentle slope of complexity: this means it should avoid big steps in complexity and keep a reasonable trade-off between ease-of-use and functional complexity. Systems might offer users, for example, different levels of complexity in performing EUD activities, ranging from simply setting parameters to integrating existing components and extending the system by developing new components (EUD-Net Thematic Network; Myers et al., 2003; 1992). The system should then evolve with the users (co-evolution) (Arondi et al., 2002), thus offering them new functionalities only when needed.

Starting with these observations, we base our methodology for designing interactive software systems on three principles: (i) the language in which the interaction with systems is expressed must be based on notations traditionally adopted in the domain (this also supports the system designers in identifying the grain problems and in defining their solutions); (ii) systems must present only the tools necessary to perform the task at hand without overwhelming users with unnecessary tools and information; (iii) systems must provide a layout which simulates the traditional layout of the tools employed in the domain, such as mechanical machines or paper-based tools.

3. Domain-Expert Users' EUD Activities

In our work, we primarily address the needs of communities of experts in scientific and technological disciplines. These communities are characterized by different technical methods, languages, goals, tasks, ways of thinking, and documentation styles. The members of a community communicate with each other through documents, expressed in some notations, which represent (materialize) abstract or concrete concepts, prescriptions, and results of activities. Often dialects arise in a community, because the notation is applied in different practical situations and environments. For example, technical mechanical drawings are organized according to rules which are different in Europe and in the USA (ISO standard). 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.

Recognizing users as domain experts means recognizing the importance of their notations and dialects as reasoning and communication tools. This also suggests the development of tools customized to a single community. Supporting co-evolution requires in turn that the tools developed for a community should be tailored by its members to the newly emerging requirements (Mørch and Mehandjiev, 2000). Tailoring can be performed only after the system has been released and therefore when it is used in the working context. In fact, the contrast often emerging between the user working activity, which is situated, collaborative and changing, and the formal theories and models that underlie and constitute the software system can be overcome by allowing users themselves to adapt the system they are using.

The diversity of the users calls for the ability to represent the meaning of a concept with different materializations, e.g., text or image or sound and to associate to the same materialization a different meaning according, for example, to the context of interaction. For example, in the medical domain the same X-ray is interpreted in different ways by a radiologist and a pneumologist. These two d-experts are however collaborating to reach a common goal. Therefore, they use the same set of data (of a patient), which, however, is represented differently according to their specific skills. Often experts work in a team to perform a common task and the team might be composed of members of different sub-communities, each sub-community with different expertise. Members of a sub-community need an appropriate computer environment, suited to them to manage their own view of the activity to be performed.

In (Costabile et al., 2003b), some situations that show the real need for environments that allow d-experts to perform various types of EUD activities were described. They emerged from the work of the authors primarily with biologists and earth scientists. 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; (2) local development by biologists who know some programming languages, for managing

data, analyzing results, or testing scientific ideas. The second type of development can be considered EUD. Moreover, many biologists feel the need to modify the application they use to fit their needs better. Here is a list of real programming situations that 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 relative to each subsection; *formatting*, i.e., renumber one's sequence positions from –3000 to +500 instead of 0–3500; *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.

In the domain of earth science, some scientists and technicians analyze 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 they have to achieve different tasks, documented by different sub-notations and tools. Therefore, their notations can be considered two dialects of the Earth Scientist & Technologist general notation.

From these experiences, two classes of d-expert activities have been proposed (Costabile et al., 2003b):

- *Class 1* includes activities that allow users, by setting some parameters, to choose among alternative behaviors (or presentations or interaction mechanisms) already available in the application; in the literature such activities are usually called parameterization, customization or personalization.
- *Class 2* includes all activities that imply some programming in any programming paradigm, thus creating or modifying a software artifact. Since we want to be as close as possible to the user, we will usually consider novel programming paradigms, such as programming by demonstration, programming with examples, visual programming, and macro generation.

In Table 9.1 examples of activities of both classes are provided.

4. SOFTWARE SHAPING WORKSHOPS

In scientific and technological communities, such as mechanical engineers, geologists, and physicians, experts often work in a team to perform a common task. The team might be composed of members of different sub-communities, each sub-community with a different expertise. Such domain experts, when working with a software application, feel

Table 9.1. Examples of activities of classes and descriptions

Class	Activity name	Activity description
Class 1	Parameterization	This is intended as a specification of unanticipated constraints in data analysis. In this situation d-experts are required to associate specific computation parameters to specific parts of the data, or to use different models of computations available in the program.
	Annotation	This is the activity in which d-experts write comments next to the data and the result files in order to highlight their meaning.
Class 2	Modeling from data	The system supporting the d-expert derives some (formal) models by observing data, e.g. a kind of regular expression is inferred from selected parts of aligned sequences (Blackwell, 2000), or patterns of interactions are derived (Arondi et al., 2002).
	Programming by demonstration	D-experts show examples of property occurrences in the data and the system infers a (visual) function from them.
	Use of 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).
	Indirect interaction with application objects	As opposed to direct manipulation, traditional interaction style tools, e.g., command languages, can be provided to support user activities.
	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.
	Extended annotation	A new functionality is associated with the annotated data. This functionality can be defined by any technique previously described.

the need to perform various activities that may even lead to the creation or modification of software artifacts, in order to obtain better support for their specific tasks. These are considered EUD activities. The need for EUD is a consequence of the user diversity and evolution discussed in Section 2.

Our approach to the design of a software system devoted to a specific community of domain-expert users is to organize the system into various environments, each one for a specific sub-community. Such environments are organized in analogy with the artisan workshops, where the artisans find only the tools necessary to carry out their activities. In a similar way, d-experts using a virtual workshop find available only the tools required to develop their activities by properly shaping the software they use. These tools must be designed and must behave in such a way that they can be used by the d-expert in the current situation. For this reason, the software environments are called SSWs (Costabile et al., 2002). SSWs allow users to develop software artifacts without the burden of using a traditional programming language, using high level visual languages, tailored to their needs. Moreover, users have the feeling of simply manipulating the objects of interest in a way similar to what they do in the real world. Indeed, they are creating an electronic document through which they can perform some computation, without writing any textual program code.

An important activity in the professionals' work is the annotation of documents. In the SSW methodology, electronic annotation is a primitive operator, on which the

communication among different d-experts and the production of new knowledge are based. A d-expert has the possibility of performing annotations of a piece of text, of a portion of an image or of the workshop in use to extend, make his/her current insights explicit regarding the considered problem, or even the features of the workshop. D-experts use annotation as a peer-to-peer communication tool when they exchange annotated documents to achieve a common task. By annotating the workshop they use, d-experts also use annotation as a tool to communicate with the design team in charge of the maintenance of the system.

D-experts play two main roles: (1) performing their working tasks, possibly informing the maintenance team of their usability problems; (2) participating in the development of the workshops. In the first role, at the time of use, d-experts can tailor the workshop to their current needs and context. For example, the annotation tools permit the definition of new widgets: as a reaction to the annotation activity performed by the d-expert, the workshop may transform the annotated document area into a new widget, to which a computational meaning is associated. This widget is added to the common knowledge base and is made accessible to other d-experts, each one accessing the data through his/her own workshop, enriched by the new widget that is adapted to the specific context. In the second role, at the design time, d-expert representatives participate directly in the development of the workshops for their daily work (*application workshops*). D-experts, even if they are non-professional software developers, are required to create or modify application workshops, i.e., software artifacts. To this end, different workshops (*system workshop*s) are made available to them, which permit the customization of each application workshop to the d-expert community needs and requirements.

This approach leads to a workshop network that tries to bridge the communicational gap between designers and domain-expert users, since all cooperate in developing computer systems customized to the needs of the user communities without requiring them to become skilled programmers. Thus the workshop network permits domain-expert users to work cooperatively in different places and at different time to reach a common goal; in this sense it becomes a collaboratory, as defined by Wulf: “a center without walls, in which researchers [in our case professionals] can perform their research [work] without regard to physical location, interacting with colleagues, accessing instrumentation, sharing data and computational resources, and accessing information in digital libraries” (Wulf, 1989).

Two levels can be distinguished in the workshop network:

1. the top level, that we call the *design level*, involves a sub-network of system workshops, including the one used by the software engineers to lead the team in developing the other workshops and the system workshops which are used by the team of HCI and domain experts to generate and/or adapt other system workshops or application workshops;
2. the bottom level, that we call the *use level*, includes a network of application workshops, which are used by end-users to perform their tasks.

Each system workshop in the design level is exploited to incrementally translate concepts and tools expressed in computer-oriented languages into tools expressed in notations that resemble the traditional user notations and are therefore understandable and manageable by users. The network organization of the SSWs depends on the working organization of the user community to which the SSWs are dedicated.

To develop an SSW network, software engineers and d-experts have first to specify the pictorial and semantic aspects of the Interaction Visual Languages (IVLs) through which users interact with workshops. In our approach, we capitalize on the theory of visual sentences developed by the Pictorial Computing Laboratory (PCL) and on the model of WIMP interaction it entails (Bottoni et al., 1999). From this theory, we derive the formal tools to obtain the definition of IVLs. In the WIMP interaction, the messages exchanged between the user and the system are the entire images represented on the screen display, which include texts, pictures, icons, etc. and the user can manifest his/her intention by operating on the input devices of the system such as a keyboard or a mouse. Users understand the meaning of such messages because they recognize some subsets of pixels on the screen as functional or perceptual units, called characteristic structures (css) (Bottoni et al., 1999).

From the machine point of view, a cs is the manifestation of a computational process, that is the result of the computer interpretation of a portion of the program P specifying the interactive system. The computer interpretation creates an entity, that we call virtual entity (ve) and keeps it active. A ve is defined by specifying its behavior, for example through statecharts, from which P can be implemented. It is important, however, to specify the set CS of css, which can appear on the screen, as well as their relations to the states of P from which they are generated. A ve is therefore specified as ve=<P, CS, <INT, MAT>>, where INT (interpretation) is a function, mapping the current cs $\in$ CS of the ve to the state u of the program P, generating it and MAT (materialization), a function mapping u to cs. A simple example of ve is the "floppy disk" icon to save a file in the iconic toolbar of MS Word^{TM}. This ve has different materializations to indicate different states of the computational process: for example, once it is clicked by the user the disk shape is highlighted and the associated computational process saves the current version of the document in a disk file. Once the document is saved, the disk shape goes back to its usual materialization (not highlighted). However, ves extend the concept of widgets (as in the case of the previously mentioned disk icon) and virtual devices (Preece, 1994), which are more independent from the interface style and include interface components possibly defined by users at run time. The creation of virtual entities by users is an EUD activity and distinguishes our approach from traditional ones, such as Visual Basic scripted buttons in MS Word^{TM}. In Section 5, we will discuss the creation of a ve by a user in a medical domain.

The SSW approach is aimed at overcoming the communicational gap between designers and users by a "gentle slope" approach to the design complexity (Myers 2003; 1992). In fact, the team of designers performs their activity by: (a) developing several specialized system workshops tailored to the needs of each type of designer in the team (HCI specialists, software engineers, d-experts); and (b) using system workshops to

develop application workshops through incremental prototypes (Carrara et al., 2002; Costabile et al., 2002). In summary, the design and implementation of application workshops is incremental and based on the contextual, progressive gain of insight into the user problems, which emerge from the activity of checking, revising, and updating the application workshops performed by each member of the design team.

The diversity of the users calls for the ability to represent the meaning of a concept with different materializations, in accordance with local cultures and the layouts used, sounds, colors, times and to associate a different meaning to the same materialization according, for example, to the context of interaction. The SSW methodology aims at developing application workshops which are tailored to the culture, skill, and articulatory abilities of specific user communities. To reach this goal, it becomes important to decouple the pictorial representation of data from their computational representation (Bottoni et al., 1999). In this way, the system is able to represent data according to the user needs, by taking into account user diversity. Several prototypes have been developed in this line, in medical and mechanical engineering (Mussio et al., 1992). XML technologies, which are based on the same concept of separating the materialization of a document from its content, are being extensively exploited.

To clarify the concepts on the SSW network, we refer to a prototype under study, designed to support different communities of physicians, namely radiologists and pneumologists, in the analysis of chest X-rays and in the generation of the diagnosis. Radiologists and pneumologists represent two sub-communities of the physicians community: they share patient-related data archives, some models for their interpretation, some notations for their presentation, but they have to perform different tasks, documented through different sub-notations and tools. Therefore, their notations can be considered two (visual) dialects of the physicians' general notation.

The SSW network for this prototype is presented in Figure 9.1. As we said, we distinguish two levels. At the top level, the design level includes the workshops used

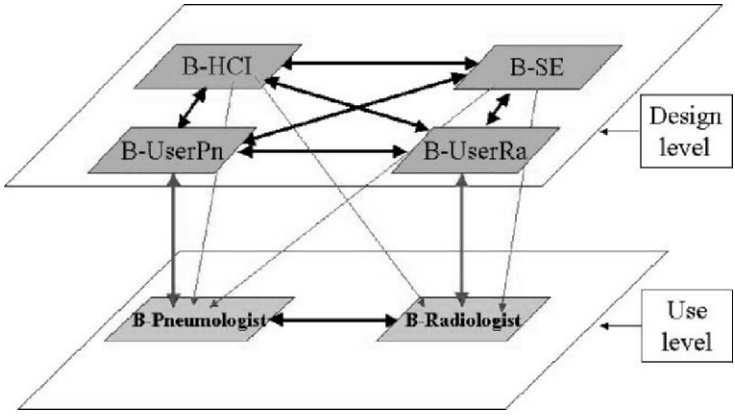

Figure 9.1. The network of Software Shaping Workshops involved in the co-evolutive use of B-Pneumologist and B-Radiologist.

by the members of the design team to develop the application workshops. The design level includes system workshops devoted to software engineers (*B-SE*), HCI experts (*B-HCI*), and d-experts (*B-UserPn, B-UserRa*), in our case, specialists in pneumology and radiology. The designers in the team collaborate in designing and updating, as required by co-evolution, the application workshops *B-Radiologist* and *B-Pneumologist*. In the design and updating phases, each member of the design team operates on the application workshop under development using his/her own system workshop tailored to his/her own culture, skills, and articulatory abilities. The application workshops are developed through a participatory design project which is carried out in an asynchronous and distributed way. At the use level, the pneumologist and radiologist, who are working in different wards or different hospitals and are involved in the study of the pulmonary diseases, can reach an agreed diagnosis using application workshops tailored to their culture, skills, and articulatory abilities, again in an asynchronous and distributed way.

In Section 5, we illustrate how EUD activities can be performed by working with *B-Radiologist* and *B-Pneumologist*. However, EUD activities can also be performed at design level: using *B-UserPn* and *B-UserRa* (see Figure 9.1), representatives of end-users may generate or adapt the application workshops *B-Radiologist* and *B-Pneumologist*. The development of *B-UserPn* and *B-UserRa* is in progress, so we focus here only on the EUD activity performed at the use level by interacting with two prototypes of *B-Radiologist* and *B-Pneumologist*. Such prototypes have been developed to speak with our domain experts, receive feedback from them about the functionalities the software system offers and understand their needs. In (Costabile et al., 2003a; Fogli et al., 2003), prototypes in the field of mechanical engineering illustrate how d-experts may perform EUD activity at the design time by interacting with software environments developed by following the SSW methodology.

5. SSWs for a Medical Domain

To concretize our view on SSWs, we introduce a scenario, drawn from an initial analysis of physicians collaborating to achieve a diagnosis (Costabile et al., 2002). In the scenario, a pneumologist and a radiologist incrementally gain insight into the case by successive interpretations and annotations of chest X-rays, performed in (possibly) different places and at (possibly) different times. They are supported by two interactive prototypes, *B-Radiologist* and *B-Pneumologist*, which share a knowledge repository. They achieve the diagnosis by updating the knowledge repository after each session of interpretation of the results reached so far and of annotation of their new findings. In particular, through the annotation activity, new software artifacts are created (e.g., a widget with a certain functionality): each new software artifact created in this way implements a virtual entity whose cs corresponds to the shape traced by the user on the X-ray and whose program P depends on the content of the annotation.

B-Radiologist and *B-Pneumologist* are application workshops that support the two physicians in recording and making the observational data available for reasoning and communication, as well as the paths of the activities physicians are performing and the progressively obtained results. To this end, they share the knowledge repository

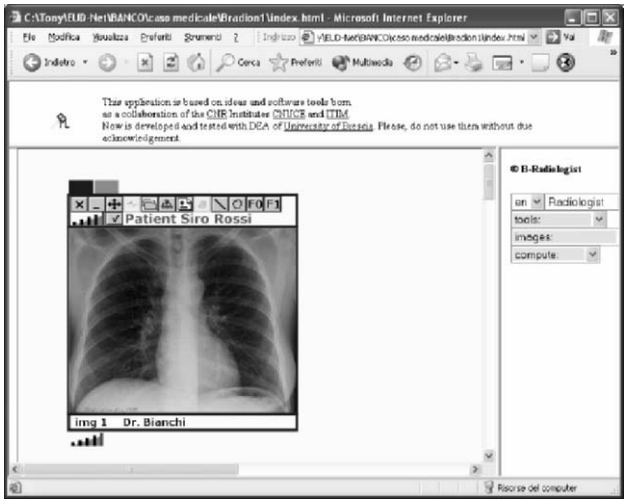

Figure 9.2. Web page with B-Radiologist workshop. The radiologist is analyzing a chest X-ray.

and also some tools for data annotation, archiving, and retrieving. However, they have to support physicians with different experience and cultural background, performing different tasks in the achievement of the diagnosis. Hence, each one is also equipped with tools specialized to the specific tasks to be performed by its own users and makes data and tools available by materializing them according to the specific culture, experience and situation of its current user.

Figure 9.2 displays a web page, as it appears to a radiologist—Dr. Bianchi, interacting with *B-Radiologist.* Due to space limitations, it is the only figure showing the complete web page, the remaining figures show only panes of our interest.

The screen is divided into two parts: the top presents the tools which interact with Internet Explorer™, the browser managing the process. The underlying part has a header at the top, presenting general information about the creators of the system. Below it, there is an *equipment area* on the right with a title identifying *B-Radiologist* as the workshop currently active and a *working area* on the left. In the equipment area, the radiologist has repositories of entities available to be worked (images and annotations) and equipment to work on the entities. Data and tools can be extracted and used or deployed in the working area and stored in the repositories. Tools are represented in the working area as icons and data as raster or vector images, materializing the css of interest. Each image represents an entity to be worked on and is associated to a handle, a toolbox and other identifiers. These four entities form a *bench*. The handle is a ve whose cs is a rectangle which identifies the bench. It is positioned on top of the toolbox and permits the bench selection and dislocation. The toolbox contains the tools required for the execution of the current task. The identifiers identify the physician performing the task, the patient to which the data refers and the image (set of data) referring to that patient.

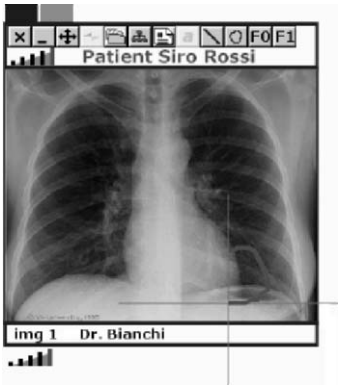

Figure 9.3. Using B-Radiologist, the radiologist circles a zone of pleural effusion.

In Figure 9.2, the radiologist is working on two benches, one associated to raster X-ray which underlies a bench associated to a transparent vector image, which is a support for annotation. Hence, two handles appear on top of the toolbox, while system generated identifiers identify Dr. Bianchi as the radiologist annotating the X-ray, Mr. Rossi as the patient and img1 as the considered image. Figure 9.2 resumes the state of the activity of interpretation of an X-ray after the radiologist (a) has obtained the data of his interest (an X-ray of the patient, whose surname is Rossi) and (b) has superimposed the annotation bench on the bench containing the X-ray.

In Figure 9.3, the radiologist (a) has recognized an area of interest denoting a pleural effusion; (b) has selected from the toolbox the tool for free-hand drawing of close curves, the tenth button from the left (whose `cs` is a close curve); and (c) *B-Radiologist* has reacted, presenting him with a cursor, whose `cs` is the cross. The radiologist is now circling the area of interest using a mouse to steer the cross, so identifying a `cs`. After closing the curve, the radiologist selects the eighth button ('a') in the top menu, firing the annotation activity; then he can type his classification of the `cs` 'Pleural effusion'. Figure 9.4 shows the radiologist storing these results by the selection of the 'add Note' button. As a reaction to this last user action, *B-Radiologist* (a) closes the annotation window; (b) adds to the framed area an icon of a pencil as an anchor to the annotation, and (c) transforms the framed area into a *widget*, by associating it to a pop-up menu. The menu title and items depend on the radiologist's classification of the `cs`s in the framed area. In other words, *B-Radiologist* creates an active widget whose characteristics depend on the contextual activity and which is added to the set of tools known to the system and then becomes available to the users. In particular, the pop-up menu associated with the widget allows the radiologist to choose between two activities related with pleural effusion areas: the density evaluation and the NMR analyses retrieval. After having obtained the results of the selected computations, the radiologist writes a new annotation suggesting a possible diagnosis to be shared with the pneumologist (potential pneumonia).

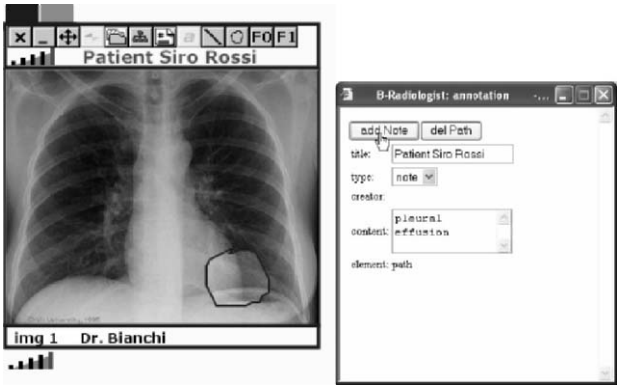

Figure 9.4. Using B-Radiologist, the radiologist annotates a zone of pleural effusion.

At the end of the annotation activity, *B-Radiologist* stores the annotation and other possible results from its activity in the knowledge repository shared with *B-Pneumologist*, permanently updating to the patient file, thus evolving *B-Radiologist, B-Pneumologist*, and the knowledge repository. In the current version, the radiologist sends an email message to the pneumologist whenever s/he wants to inform the other physician that the knowledge repository has been updated.

The workshops make two different types of tools available to their users: system predefined tools, which are always available and the tools created and associated to the data by the users, such as the annotation button. Their meaning depends on the medical context in which annotation is used. For example, in *B-Pneumologist*, a CS classified as 'pleural effusion' is not associated to the same menu as in *B-Radiologist*, but is associated to a multi-link to the records of available data on the patient, i.e., radiological interpretation, associated TACs, and hematic parameters. In *B-Pneumologist* the pencil associated to the area of interest outlined by the radiologist is associated to the tools for visualizing the data related to the patient and supporting their exploration in order to reach a final diagnosis. The linking to the new tools—the new computational meaning of the annotation—occurs at start-up time, i.e., when a physician accesses *B-Pneumologist* to initiate the interactive session. Therefore, when the pneumologist Dr. Neri selects the pencil, *B-Pneumologist* displays the text of the annotation performed by the radiologist and the multi-link (Figure 9.5). In Figure 9.5 the pneumologist selects 'Radiological interpretation' to query details on Dr. Bianchi's observations. He obtains the media and estimated error of the density of the pleural effusion. He can also add his diagnosis to the document recording the opinions increasingly annotated by Dr. Bianchi (Figure 9.6).

The SSW life cycle follows a star approach (Hix and Hartson, 1993), starting with the analysis of the users of the application workshops. The design process proceeds by developing incremental workshop prototypes at various levels in the hierarchy, going bottom-up as well as top-down. In the case study, user analysis started by examining how the radiologists classify, interpret, and annotate chest X-rays and how the pneumologists

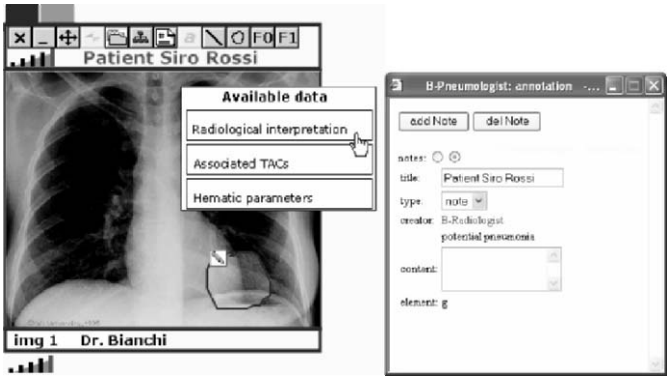

Figure 9.5. Working in B-Pneumologist the pneumologist accesses the radiological interpretation.

use the interpreted images, provide their diagnoses and record them using an annotation tool. On the basis of this analysis, the team of experts involved in the design felt the need to develop the two separate but consistent application workshops, each one dedicated to a specific sub-community. Moreover, the team of experts observed that not all situations can be foreseen in advance and that sometimes *B-Radiologist* and *B-Pneumologist* must both be consistently adapted to different new tasks and situations. This adaptation requires the knowledge of both dialects and activities, of the tasks to be executed and of the working organization and the awareness of the use of diagnostic documents outside the organization. Only senior physicians have such a global skill and knowledge and can assume this responsibility. Therefore, the team decided that a senior pneumologist and a senior radiologist should act as managers of the whole activity and be responsible for recognizing the tasks to be performed, identifying the dialect notations of interest, and consequently defining the system of consistent application workshops. The senior physicians achieve these goals using two system workshops, *B-UserRa* and *B-UserPn*,

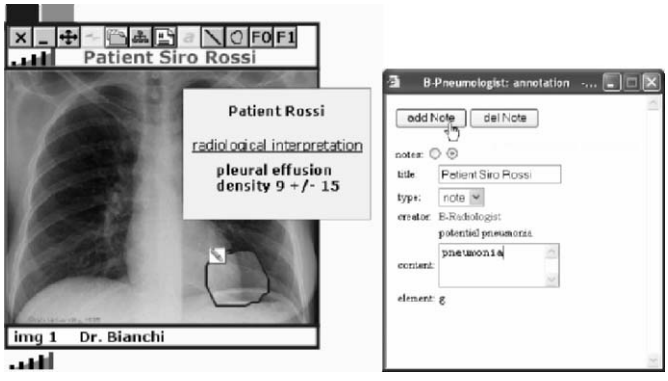

Figure 9.6. The pneumologist obtains the radiological interpretation and gives his diagnosis.

where they find usable tools for implementing and adapting both *B-Radiologist* and *B-Pneumologist* (see Figure 9.1). They can also collaborate with HCI experts and software engineers as required by the progressive results of the experiences.

6. Related Work

As designers, our challenge is to develop interactive software systems which (a) support their users in exploiting their "practical competence and professional artistry in achieving a task" (Schön, 1983) and (b) enable the practitioner to develop and extend the knowledge available to the profession (Schön, 1983). To achieve this goal, we adopt a 'semiotic computer' point of view (Andersen, 2001; 1992), recognizing the existence of two interpretations of each CS and the importance of notations developed by d-expert communities such as reasoning, communication, and documentation tools.

Another important issue in our design approach is the co-evolution of users and systems. Carroll and Rosson (1992) speak about co-evolution of users and tasks, while co-evolution of artifacts supporting HCI design in the different steps of the product lifecycle is discussed by (Brown et al., 1998). Co-evolution of users and systems, as proposed in this paper, stresses the importance of co-evolving the systems, as soon as users evolve the performance of their tasks. Co-evolution of users and systems is rooted in the usability engineering, in that it supports designers in collecting feedback on systems from the field of use, to improve the system usability (Nielsen, 1993). Tools designed to support co-evolution are suitable for observational evaluation in user-centered design approaches (Preece, 1994). Moreover, these evaluation tools integrated within the SSW networks allow system adaptation (Arondi et al., 2002), in the more general frame of co-evolution of users, organization, systems, and environment, as observed by Bourguin et al. (2001). This extends the view of Mackay, who postulates that the use of information technology is a co-adaptive phenomenon (Mackay, 1990). Co-evolution implies tailoring. SSWs are designed to permit tailoring, i.e. "further development of an application during use to adapt it to complex work situations" (Kahler et al., 2000) by end-users.

In our approach, d-experts play a role similar to the handymen in (MacLean et al., 1990). The handyman bridges between workers (people using a computer application) and computer professionals; s/he is able to work alongside users and communicate their needs to programmers. Similarly, d-experts bridge between workers and computer professionals, but are end-users themselves and not necessarily computer professionals. They must be provided with environments to be able to participate in SSWs development that are adapted to their culture, skills and articulatory abilities. In (Costabile et al., 2003a; Fogli et al., 2003) we describe an environment devoted to mechanical engineers who were the d-experts involved in the development of the application workshop devoted to assembly-line operators.

In (Mackay, 1991) and (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 customize the system, intending as

customization the possibility of modifying software to make persistent changes. She finds that many users do not customize 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 these users actually perform activities of end user programming, thus generating new software artifacts; these users are even able to master the formal languages embedded in these applications when they have a real motivation for doing so.

SSWs are also in the area of research on *Gentle Slope Systems*, "which are systems where for each incremental increase in the level of customizability, the user only needs to learn an incremental amount" (Myers, 2003). In fact, the SSW methodology favors the construction of systems which are more acceptable to the users, since they are based on a knowledge (often tacit), languages, and notations belonging to the interested user community. Moreover, SSWs allow users to perform EUD activities, overcoming the problems currently affecting other types of EUD, such as the development of macros in spreadsheets or of scripts in active web pages, which usually require the learning of conventional programming (Myers, 2003).

Domain knowledge plays a key role in the approach to software system construction described by Fischer (1998), Fischer and Ostwald (2002), and Fischer et al. (2001). In these works, the authors propose designing systems as *seeds*, with a subsequent *evolutionary growth*, followed by a *reseeding* phase. SER (Seeding, Evolutionary growth, Reseeding) is thus a process model for the development and evolution of the so-called DODEs (Domain-Oriented Design Environments), which are "software systems that support design activities within particular domains and that are built specifically to evolve" (Fischer, 1998). Three intertwined levels of design activities and system development are envisaged: at the lower level, there is a multifaceted domain-independent architecture constituting the framework for building evolvable systems; at the middle level, the multifaceted architecture is instantiated for a particular domain in order to create a DODE; at the top level, there are individual artifacts in the domain, developed by exploiting the information contained in the DODE. The SER model describes the evolution of such environments at the three levels.

We have a domain-independent architecture as well, which can be instantiated according to the considered domain (Fogli et al., 2003). This architecture is implemented by exploiting open source code, such as XML-suite tools and ECMAscript language, so that a system SSW and the application SSWs generated from it have the same web-based structure. However, the construction of SSWs is always based on a formal specification of the Interaction Visual Languages through which the user interacts in order to guarantee a variety of properties [such as usability, determinism, viability, non-ambiguity (Preece, 1994)]. The architecture reflects the formal model proposed to specify the static and dynamics component of the systems. In the SSW framework there is a clear distinction between the design and the use level: the system workshops at the design level can be used by d-experts to create and/or update application workshops. Both system and application workshops can first represent seeds, which, according to

the user interaction, can be evolved into new system and application workshops respectively, still remaining separate. This separation, which helps not to disorient the users during their task activities, is not so well established in the works with which we are comparing ours.

There is a separation between the design and use level in many commercial tools for authoring systems, such as, for example, Micromedia Flash or Toolbook. In these systems, the author mode and the user mode are present, but the author mode usually requires the use of a programming language (typically a scripting one). Therefore, these systems turn out to be less accessible and usable by experts in domains different from computer science. Moreover, both system and application workshops present the users with a familiar environment in which only the tools necessary to carry out the working task are available. On the other hand, also commercial tools allow the definition of libraries of personalized tools, but they may only be added to the tools already available in the developmental system.

7. Conclusions

Nowadays, new computer technologies force many users, who are not experts in computer science but are experts in their own domain of activity, to ask for software environments in which they can do some programming activity related to their tasks and adapt the environments to their emerging new needs. Therefore, in such a scenario, EUD becomes a challenging issue for future software systems. To study novel solutions to cope with this issue, we propose a unified view of the variety of phenomena affecting the HCI process, such as the communicational gap which often exists between designers and systems, the user diversity, the co-evolution of systems and users, the grain imposed by software tools, the implicit information, and tacit knowledge that influence users' behavior while interacting with software systems.

In the chapter we have analyzed these phenomena, by showing the hurdles they impose in user activities and the new interaction and communication possibilities they offer and have framed them in a systemic HCI model. Such a model underlies our approach to system design and development—the SSW methodology. Within the SSW methodology, EUD means that (1) d-experts may create other SSWs suitable to the considered domain by using simple facilities, such as a drag-and-drop; and (2) d-experts may create new tools within the workshop they are using, for example as a result of an annotation activity. The latter case has been analyzed in a medical domain: physicians use tailored environments (application workshops), which they can enrich by themselves with new tools through annotation activity. The results of the annotation are shared by the application workshops, so allowing physicians to create tools to be used also by their colleagues, possibly according to their own needs, background, expertise, and preferences. In both cases, users are required neither to write codes, nor to know any programming languages or paradigms. Users simply create programs by interacting with the system through visual languages resembling the activities they usually perform in their daily work. For the sake of brevity, the case study discussed in

this paper shows only an example of the second type of EUD activity. More details about the first one are by Costabile et al., (2003a) and Fogli et al. (2003). The architecture we have implemented to develop SSWs is based on the W3C framework and the XML technology, thus permitting the construction of very "light" applications (Fogli et al., 2003).

Acknowledgments

The authors wish to thank the reviewers for their useful comments and Giuseppe Fresta for the stimulating discussions during the development of this work and for his contribution to the implementation of the prototypes presented in the paper. They also wish to thank Dr. Lynn Rudd for her help in correcting the English manuscript.

The support of EUD-Net Thematic Network (IST-2001-37470) is acknowledged.

References

Andersen, P.B. (1992). Computer semiotics. *Scandinavian Journal of Information Systems* **4**, 3–30.

Andersen, P.B. (2001). What semiotics can and cannot do for HCI. *Knowledge Based Systems* **14**, 419–424.

Arondi, S., Baroni, P., Fogli, D. and Mussio, P. (2002). Supporting co-evolution of users and systems by the recognition of Interaction Patterns. *Proceedings of the International Conference on Advanced Visual Interfaces (AVI 2002)*, Trento, Italy, New York: ACM Press, pp. 177–189.

Barnard, P., May, J., Duke, D. and Duce, D. (2000). Systems, Interactions, and Macrotheory. *ACM Trans. on Human-Computer Interaction* **7**(2), 222–262.

Blackwell, A. (2001). See what you need: Helping end users to build abstractions. *Journal of Visual Languages and Computing* **12**(5), 475–499.

Borchers, J. (2001). *A Pattern Approach to Interaction Design*, Chichester: John Wiley & Sons.

Bottoni, P., Costabile, M.F. and Mussio, P. (1999). Specification and dialogue control of visual interaction through visual rewriting systems. *ACM Trans. on Programming Languages and Systems (TOPLAS)* **21**(6), 1077–1136.

Bourguin, G., Derycke, A. and Tarby, J.C. (2001). Beyond the interface: Co-evolution inside interactive systems—A proposal founded on activity theory. *Proceedings of IHM-HCI 2001,* Lille, France, Berlin Heidelberg: Springer-Verlag, pp. 297–310.

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

Brown, J., Graham, T.C.N. and Wright, T. (1998). The Vista environment for the coevolutionary design of user interfaces. *Proceedings of CHI 98,* Los Angeles, New York: ACM Press, 376–383.

Carrara, P., Fogli, D., Fresta, G. and Mussio, P. (2002). Toward overcoming culture, skill and situation hurdles in human-computer interaction. *International Journal Universal Access in the Information Society* **1**(4), 288–304.

Carroll, J.M. and Rosson, M.B. (1992). Deliberated evolution: Stalking the view matcher in design space. *Human-Computer Interaction* **6** (3 and 4), 281–318.

Costabile, M.F., Fogli, D., Fresta, G., Mussio, P. and Piccinno, A. (2002). Computer environments for improving end-user accessibility. *Proceedings of 7th ERCIM Workshop "User Interfaces For All"*, Paris, France, LNCS 2615, Berlin Heidelberg: Springer-Verlag, pp. 187–198.

Costabile, M. F., Fogli, D., Fresta, G, Mussio, P. and Piccinno, A. (2003a). Building environments for end-user development and tailoring. *Proceedings 2003 IEEE Symposia on Human Centric*

Computing Languages and Environments (HCC' 03), Aukland, New Zeland, Danvers: IEEE Computer Society, pp. 31–38.

Costabile, M.F., Fogli, D., Letondal, C., Mussio, P. and Piccinno, A. (2003b). Domain-expert users and their needs of software development, *Proceedings of UAHCI Conference*, Crete, London: Lawrence Erlbaum Associates, pp. 232–236.

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

Dix, A., Finlay, J., Abowd, G. and Beale, R. (1998). *Human-Computer Interaction*, London: Prentice Hall.

EUD-Net Thematic Network, Network of Excellence on End-User Development, http://giove.cnuce.cnr.it/eud-net.htm.

Fischer, G., Grudin, J., McCall, R., Ostwald, J., Redmiles, D., Reeves, B. and Shipman, F. (2001). Seeding, evolutionary growth and reseeding: The incremental development of collaborative design environments. In: *Coordination Theory and Collaboration Technology*, Mahwah, NJ: Lawrence Erlbaum Associates, 447–472.

Fischer, G. and Ostwald, J. (2002). Seeding, evolutionary growth, and reseeding: Enriching participatory design with informed participation. *Proceedings of PDC'02*, Malmö, Sweden, New York: ACM Press, pp. 135–143.

Fischer, G. (1998). Seeding, evolutionary growth, and reseeding: Constructing, capturing, and evolving knowledge in domain-oriented design environments. *Automated Software Engineering* **5**(4), 447–468.

Fogli, D., Piccinno A. and Salvi, D. (2003). What users see is what users need. *Proceedings of DMS 03*, Miami, USA, Skokie, USA: Knowledge Systems Institute, pp. 335–340.

Hix, D. and Hartson, H. R. (1993). *Developing User Interfaces: Ensuring Usability through Product & Process*. Chichester: John Wiley & Sons.

Kahler, H., Mørch, A., Stiemerling, O. and Wulf, V. (2000). Introduction to the special issue on tailorable systems and cooperative work. *Computer Supported Cooperative Work* **9**, 1–4, Kluwer Academic Publishers.

ISO Standard: ISO 5456 Technical Drawing Projection Methods.

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

Mackay, W.E. (1990). Users and Customizable Software: A Co-Adaptive Phenomenon, Ph. D. Thesis, MIT.

Mackay, W.E. (1991). Triggers and barriers to customizing software. *Proceedings of ACM CHI'90*, New Orleans, USA. New York: ACM Press, pp. 153–160.

MacLean, A., Kathleen, C., Lövstrand, L. and Moran, T. (1990). User-tailorable systems: pressing the issues with buttons, *Proceedings of ACM CHI'90*, New Orleans, USA. New York: ACM Press, pp. 175–182.

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

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**, 2000, 75–100.

Mussio, P. (2004). E-Documents as tools for the humanized management of community knowledge. In: H. Linger et al. (eds.), *Constructing the Infrastructure for the Knowledge Economy: Methods and Tools*; *Theory and Practice*. Dordrecht: Kluwer Academic, pp. 27–43.

Mussio P, Finadri M, Gentini P. and Colombo F. (1992). A bootstrap technique to visual interface design and development. *The Visual Computer* **8**(2), 75–93.

Mussio, P., Pietrogrande, M. and Protti, M. (1991). Simulation of hepatological models: A study in visual interactive exploration of scientific problems. *Journal of Visual Languages and Computing* **2**, 75–95.

Myers, B.A., Hudson, S. E. and Randy, P. (2003). Past, present, and future of user interface software tools. In Carroll (ed.), *Human-Computer Interaction in the New Millennium*, New York: Addison-Wesley, pp. 213–233.

Myers, B.A., Smith, D.C. and Horn, B. (1992). Report of the 'End-User Programming' Working Group. In: *Languages for Developing User Interfaces*. Boston, MA: Jones and Bartlett, pp. 343–366.

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

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

Polanyi, M. (1967). *The Tacit Dimension*. London: Rouledge & Kegan Paul.

Preece, J. (1994). *Human-Computer Interaction*. Harlow: Addison-Wesley.

Schön, D. (1983). *The Reflective Practinioner—How Professionals Think in Action*. Jackson: Basic Books.

Schuler, D. and Namioka, A. (1993). *Preface—Participatory Design, Principles and Practice*, Lawrence Erlbaum Ass. N.J: Inc. Hillsday, vii.

Wulf, W.A. (1989). The National Collaboratory: A White Paper. Appendix A in Toward a National Collaboratory, *National Science Foundation invitational workshop* held at Rockfeller University, Washington D.C., p. 1.