Handbook of Software Engineering and Knowledge Engineering
Vol. 1, 2001, pp. 179-192, ISBN: 981-02-4973-X
© World Scientific Publishing Company

USABILITY IN THE SOFTWARE LIFE CYCLE

MARIA FRANCESCA COSTABILE

Dipartimento di Informatica, Università di Bari
Via Orabona 4, 70125 Bari, Italy
costabile@di.uniba.it

The objective of this chapter is to clarify the role of usability processes and methods within the software life cycle. After discussing the concept of usability as quality factor and providing various definitions, including those in ISO standards, it presents the user-centered design methodology for designing usable systems and provides indications on how to include usability in the software life cycle.

Keywords: usability, software life cycle, user-centered design

1. Introduction

"People are required to conform to technology. It is time to reverse this trend, time to make technology conform to people" Donald A. Norman says in the article "Designing the Future", published in Scientific American in September 1995. This is particularly true for the computer technology, which nowadays is providing everybody with the possibility of using a computer, and directly interacting with software systems and exploring information resources. For too long we have designed software systems that require a lot of effort by users. Many computer users experience frustration and anxiety, and this phenomenon is even growing, due to the huge amount of data that are becoming available on networked information systems, such as the WWW. As a consequence, current software systems must provide enhanced user interfaces that support this intensive interaction with users.

Referring to any software product, from the users' point of view the user interface is the most important component, since it is what the users see and work with when using a product. We need to create successful user interfaces, whose quality is primarily evaluated from the point of view of the users. A good design of the user interface results when designers understand people as well as technology. The designers must understand who will be the users of their products, their personal characteristics, their physical capabilities, their goals and the tasks they need to accomplish, the circumstances under which they work.

Unfortunately, very little attention has been devoted to the user interface by software engineers, with the unavoidable result that most software systems are very hard to use. The user interface has been considered a detail to be added at the end of the development of the system. This is not possible anymore with interactive systems, for which it is estimated that about 50% of the code is devoted to the user interface [1].

Since the late 1970s, within the software engineering discipline quality factors have been introduced as part of software quality measurements frameworks that have been developed to provide a measurement approach for software engineering. In [2], McCall says that quality factors "represent attributes or characteristics of the software that a user, or customer of the software product, would relate to its overall quality". Pionering work on quality factors was performed by Boehm [3], and by McCall and al. [4]. Interestingly, already at that time one of these factors was *usability*, defined as the "effort required to learn, operate, prepare input, and interpret output of a program" [2,4].

The fact that it was already included as a quality factor did not really mean that software engineers have put so far much attention on it. Traditionally, they are trained to judge their work by criteria that may have little to do with the needs and constraints of the end users. They are more sensitive to other factors, such as efficiency of code or flexibility of the programs, which are certainly important engineering goals, but have little or nothing to do with creating a computer system accessible to and supportive for a particular type of user. The need of improving the user interface is now emerging, and more recent books on software engineering (e.g. [5]) devote more attention to the user interface design. Software engineers are starting to appreciate the importance of high-quality users interfaces, but more has to be done. In particular, we need to insert the notions developed in the Human-Computer Interaction (HCI) discipline into the software professional training. HCI is now a consolidated discipline and there is a fast accumulating body of knowledge regarding user interface design that must be taken into account to guide software developers into their design decisions, thus avoiding to repeat bad designs. More importantly, we must integrate user interface design methods and techniques into the standard software development methodologies and to adopt for the interface testing and quality control procedures analogous to those already accepted for testing other aspects of software design [6,7].

This chapter aims at clarifying the role of usability processes and methods within the software life cycle. The organization of the chapter is the following. Section 2 discusses the concept of usability and provides various definitions, including those in ISO standards. Section 3 presents the main features of the user-centered design methodology whose main objective is to design usable software systems. Section 4 provides indications on how the software life cycle can be augmented to include usability, while Section 5 discusses different usability evaluation methods and their applicability at the various steps of the software life cycle. Finally, the conclusions are provided.

2. Definition of usability

Regardless the scarce attention devoted so far to usability by software engineers, it is now widely acknowledged that usability is a crucial factor of the overall quality of interactive applications. Several definitions of usability have been proposed. We report here those we consider the most representative. The first one is proposed by J. Nielsen and provides a detailed description of the usability attributes [8]. The second definition is provided in the standard ISO/IEC 9126 and reflects the software engineering perspective over the quality of software products [9]. As it will be shown in the discussion, from a certain point of view the latter definition is similar to the one given in the ISO 9241 standard [10], that is the standard of the ergonomic and HCI communities.

Nielsen proposes a model in which usability is presented as one of the aspects that characterizes a global feature of a system that is *acceptability* by the end users, reflecting whether the system is good enough to satisfy the needs and the requirements of the users. In Nielsen' model, usability is not a one-dimensional property of a system, rather it has multiple components. It can be decomposed into five attributes: *learnability*, i.e., the ease of learning the functionality and the behavior of the system; *efficiency*, i.e., the level of attainable productivity, once the user has learned the system; *memorability*, i.e., the ease of remembering the system functionality, so that the casual user can return to the system after a period of non-use, without needing to learn again how to use it; *low error rate*, i.e., the capability of the system to support users in making less errors during the use of the system, and in case they make errors, to let them easily recover; *user's satisfaction*, i.e., the measure in which the users like the system. The latter attribute must not be underestimated, since finding a system pleasant to use increases user's productivity**.**

These attributes can be objectively and empirically verified through different evaluation methods. Defining the abstract concept of usability in terms of more precise and measurable components is an important step towards the definition of usability engineering as a discipline, where usability is not just argued, but is systematically approached, evaluated, and improved [8].

We already said in the introduction that usability has been so far neglected by software engineers. Nevertheless, it was even mentioned in the original definition of the standard for software product evaluation ISO/IEC 9126. In a more recent formulation, the standard ISO/IEC 9126-1 (*Information-Technology Software Product Quality*) emphasizes the importance of *designing for quality*, focusing on intrinsic system features which can help create products which are effective, efficient and satisfying for the intended users [9]. The overall quality of a software product is given by its internal and external capability to support the achievement of the goal of users and their organizations, thus improving productivity and human health. The standard describes a model for software product quality, which includes internal quality, external quality and quality in use. Usability is defined as one of the six characteristics of software quality. Specifically, it is the "capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions". It is further subdivided into five subcharacteristics: *understandability*, i.e., the intrinsic capability of the software product of showing to the users its appropriateness to the tasks to be accomplished and to the context of use; *learnability*, i.e., the intrinsic capability of the software product to help users to easily learn its functionality; *operability*, i.e., the intrinsic capability of the software product to make possible for the users the execution and the control of its functionality; *attractiveness*, i.e., the intrinsic capability of the software product to be pleasant for users; *compliance*, i.e., the capability of the software product to adhere to standards, conventions, style guides about usability.

The standard then introduces the concept of *quality in use*, as a feature of the interaction between user and software product, which is measurable only in the context of a real and observable task, also taking into consideration different relevant internal attributes, such as usability. Quality in use is defined in terms of factors that represent the user's view of the software quality, i.e., *effectiveness*, *productivity*, *safety* and *satisfaction*. These factors are very much related to those defining usability in another standard, the ISO 9241 (*Ergonomic Requirements for Office Work with Visual Display Terminals*), and the concept of quality in use, as described in ISO/IEC 9126-1, is closer to that of usability given in Part 11 of ISO 9241 (*Guidance on Usability*), where it is defined as “the extent to which a product can be used by specified users to achieve specified goals with *effectiveness*, *efficiency* and *satisfaction* in a specified context of use”. *Effectiveness* is defined as the accuracy and the completeness with which specified users achieve specified goals in particular environments. *Efficiency* refers to the resources expended in relation to the accuracy and completeness of goals achieved (it is similar to the productivity factor that characterizes quality in use in the ISO/IEC 9621-1). *Satisfaction* is defined as the comfort and the acceptability of the system for its users and other people affected by its use. Usability is therefore intended as a high level goal of system design. We may conclude that both concepts of quality in use and usability, as defined in ISO 9241, incorporate the most significant aspects generally associated to usability by the HCI community.

The overall standard ISO 9241 contains guidance on user interfaces design, and provides requirements and recommendations which can be used during the design and evaluation of user interfaces [10]. The standard consists of 17 parts. The first nine parts are concerned with hardware issues (requirements for visual display, colors and non-keyboard input devices), the others are devoted to software issues such as the design of different styles of the dialog between the user and the computer (dialog principles, menu dialogs, presentation of information, user guidance, command dialogs, direct manipulation dialogs, and form-filling dialogs).

Usability is strictly dependent on the particular circumstances in which a product is used, i.e., the nature of the users, the tasks they perform, and the physical and social environments in which they operate. Special attention must be devoted to usability of software products used in stress conditions by the user, e.g. safety critical systems like those for flight control.

The HCI discipline is devoting a lot emphasis on defining methods for ensuring usability of interactive systems. Much attention on usability is currently paid by industry, which is recognizing the importance of adopting usability methods during product development, for verifying the usability of new products before they are put on the market [11]. Some studies have in fact demonstrated how the use of such methods enables cost saving, with a high cost-benefit ratio [12,13].

3. System-centered vs user-centered design

As discussed in [14], one of the reasons why many high-tech products, including computer-based systems as well as electronic equipment and every day appliances, are so hard to use is that during the development of a product, the emphasis and focus have been on the system, not on the people who will be the ultimate end-user. May be developers counted on the fact that humans are flexible and adaptable, they can better adapt to the machine rather than vice versa. Human needs have been neglected in the past also because engineers were developing products for end users who were very much like themselves. With the large spreading of computers everywhere, the target audience has changed dramatically and keeps changing every day. One of the main requirements of the information technology society is to design for universal access, i.e., computer systems must be accessible by any kind of users. What has been done in the past does not work for today's users and technology. Designers must allow the human users to focus on the task at hand and not on the means for doing that task. We need methods and techniques to help designers change the way they view and design products, methods that work from the users' needs and abilities.

One of the approaches in this directions is user-centered design, which has already proven as a key factor for leading towards the development of successful interfaces [14-16]. User-centered design implies that final users are involved from the very beginning of the planning stage, and identifying user requirements becomes a crucial phase. Early involvement of users has the potential for preventing serious mistakes when designing innovative systems. Indeed, it compels designers to think in terms of utility and usability of the system they are going to develop. Benefits of the user-centered approach are mainly related to completeness of system functionality, repair effort saving, as well as user satisfaction. Involving users from early stages allows basing the system core on what is effectively needed. Poor or inadequate requirement specifications can determine interaction difficulties, including lack of facilities and usability problems. Even if late evaluations are useful to assess the usability of final systems, it is unrealistic to expect that these results bring about a complete redesign.

The basic principles of user-centered design are: 1) analyze users and task; 2) design and implement the system iteratively through prototypes of increasing complexity; 3) evaluate design choices and prototypes with users. User-centered approach requires understanding reality: who will use the system, where, how, and to do what. Then, the system is designed iterating a design-implementation-evaluation cycle. In this way it is possible to avoid serious mistakes and to save re-implementation time, since the first design is based on empirical knowledge of user behavior, needs, and expectations. Collecting user information is not an easy task, even discussing with users, since users often neglect aspects that they erroneously consider not important.

Many different techniques can be applied for collecting user information, among them direct observation, interviews and questionnaires [13,15,16]. Direct observation means observing the users while they carry out their tasks at their workplace. It is the most reliable and precise method for collecting data about users, especially valuable for identifying user classes and related tasks. Moreover, it allows identifying critical factors, like social pressure, that can have a strong effect on user behavior when the system will be used in the field. Unfortunately, direct observation is very expensive because it requires experimenters to observe each user individually. For this reason, it is useful when a reduced number of observations is enough to generalize behavioral predictions or when hypotheses have to be tested rather than generated. Interviews collect self-reported experience, opinion, and behavioral motivations. They are essential to finding out procedural knowledge as well as problems with currently used tools. Interviews cost a bit less than direct observations, because they can be shorter and easier to code. However, they still require skilled experimenters to be effective. By contrast, self-administered questionnaires can be handed out and collected by untrained personnel allowing to gather from various users a huge quantity of data at low cost. They allow statistical analyses and stronger generalizations than interviews.

Questionnaires provide an overview on the current situation as well as specific answers. Which combination of these methods is worth applying depends both on requirements and budget. By elaborating the outcome of the knowledge phase, designers define a first version of the system. At this stage, design techniques (e.g., task-centered [17] or scenario-based [16]) provide satisfying solutions. The goal is to explore different design alternatives before settling on a single proposal to be further developed. Possibly, in this way designers will propose different solutions and different interaction strategies. Techniques like paper mock-ups and prototyping can be applied. Paper mock-ups are the cheapest: pieces of the system interface are drawn on paper and the interaction with a user is simulated by an experimenter. Despite its trivial appearance, this technique allows collecting reliable data which can be used for parallel reviewing. Prototyping allows testing some functionalities in depth (vertical prototyping) or the whole interface (horizontal prototyping). Then, one or more solutions can be evaluated with or without users. This step, called formative evaluation, aims at checking some choices and getting hints for revising the design.

At the end of the design cycle, summative evaluations are run. They test of the final system with actual users performing real tasks in their working environment. Therefore, a summative evaluation should be considered as the very last confirmation of the correctness of the hypotheses stated during the design process.

4. Modifying the software life cycle to include usability

Software production process is a way to call the process followed to build, deliver, and evolve a software product [18]. By acknowledging the fact that software, like any other industrial product, has a life cycle that spans from the initial concept formation of a software system up to its retirement, this process is generally called *software life cycle*. Several models of the software life cycle have been proposed, with the aim of controlling the life cycle and therefore producing high quality software products reliably, predictably and efficiently. Our intent is not to discuss the various models, but to raise some issues that affect the usability of interactive systems and are relevant within all activities of the software life cycle. To this aim, we take into account the standard *waterfall model*, in which each activity naturally leads into the next: the requirements are collected at the beginning, and then this information is processed and converted into a general design that drives next phases of detailed design of modules, which are then coded, integrated and tested. Then the product is completed, tested and maintained for the rest of its life. The activities of the classical waterfall model are the white blocks in Fig. 1.

In this cycle, which is system-centered, usability is not addressed. Moreover, there are some further drawbacks. For instance, the system is tested only at the end of the cycle, when unfortunately is too late for going through radical design modifications to cope with possible discrepancies with the requirements. Another problem is that these requirements are collected with customers, who often are different from the people who will use the system. We call customers the people who negotiate with designers the features of the intended system, while users or end users are those people that will actually use the designed systems. As an example, customers may be company managers that request and buy the new system, while users will be the company employees who will work with the new system. Furthermore, requirements are usually restricted to functional requirements, i.e., services the system must provide in the work domain, and do not take into account features of the system more directly related to the manner in which these services must be provided, such as easy to learn, easy to use, safety, etc.

A direct consequence of the restricted nature of the requirement specifications is that, usually, system testing not only is performed late in the development cycle, but is also limited to some of its functional aspects, thus neglecting system usability. Hopefully, the health and safety regulations now available and the ISO standards mentioned in the previous section will then make necessary to certify the final system according to usability as well.

In order to create usable interactive systems, it is therefore necessary to augment the standard life cycle to explicitly address usability issues. The shadowed boxes in Fig. 1 indicate some activities to be performed in order to shift from the system-centered design typical of the classical waterfall model to user-centered design that may lead to designing usable systems. The shadowed box denoted by 1 indicates that it is mandatory to integrate the requirement phase with a careful analysis of the users, meaning the people who will actually use the system, the tasks they perform and the environments in which they work. The best

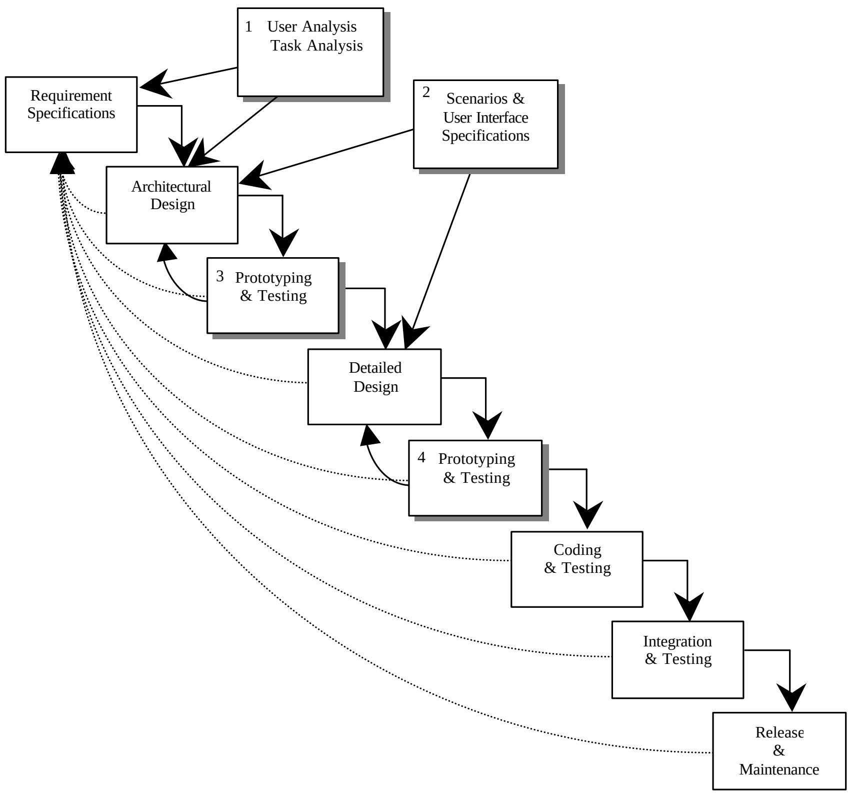

Fig. 1. The revised waterfall life cycle. White boxes refer to the classical model, while gray boxes are added to include usability.

way to collect this information is by visiting users at their workplace, observe the way the carry out their tasks, and talk to them through different types of interviews [13]. The user analysis will also affect the general design of the system. Moreover, the classical phases of architectural design and detailed design should explicitly include the design of the user interface, that is not anymore deferred to the end of the system development. Indeed, the user interface is the most relevant part of the system from the users' point of view, and its design should be discussed by the designers directly with the users since the very beginning of the life cycle. As indicated in the shadowed box 2, use scenarios [16], that is a sequence of steps describing an interaction between a user and the system, may help figuring out the design of a usable interface.

As discussed in the previous section, the user-centered design methodology stresses the iteration of the design-implementation-evaluation cycle. We therefore inserted shadowed boxes 3 and 4 in the waterfall model in order to make explicit that the both architectural and detailed design of software modules must be carried out through the development of prototypes that are evaluated with users to check if they meet the intended requirements. If not, a new prototype is developed and then evaluated again, and this iterative process ends when the specified requirements are satisfied; only at this point we may proceed to the development of a more advanced prototype, that includes new functionalities. Moreover, the iterative approach actually implies that from each phase of the software life cycle it is possible to go back to any of the previous phases, in particular to go back and update the requirement specifications, as indicated by the dotted arrows in Fig. 1. Paper mock-ups may be used as early prototypes, since they are very easy to develop and can then save time still

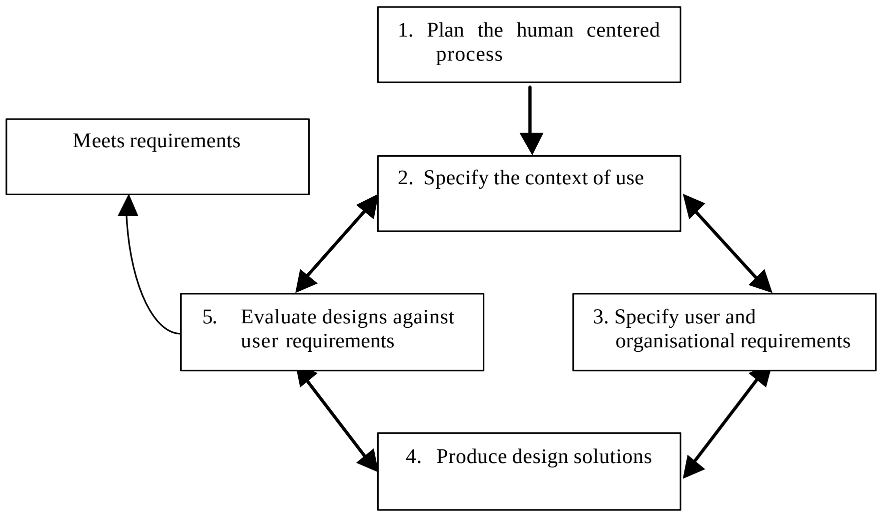
1. Plan the human centered process
Meets requirements
2. Specify the context of use
5. Evaluate designs against user requirements
3. Specify user and organisational requirements
4. Produce design solutions

Fig. 2. The ISO 13407 Human-centered design process for interactive systems.

giving the possibility of checking the important ideas with users. Prototypes developed with languages such as Visual Basic™ are the next phase after paper prototypes, and are used by many developers.

The key principles of the user-centered design methodology, namely the focus on users, the task they perform and the context in which they work, and the iterative development through prototypes of increasing complexity that are evaluated with the users, have been captured in the standard ISO 13407 (Human-centered design process for interactive systems), that is shown in Fig. 2 [19]. The design solutions mentioned in block 4 of Fig. 2 are implemented through prototypes that are evaluated, and if they do not meet the specified requirements, the process is iterated and goes again through a revision of the specifications and a proposal of a new prototype. The iterative process is stopped when requirements are met.

From our discussion above, it follows that evaluation represents the central phase in the development cycle. For this reason, within the HCI community, Hartson and Hix have developed the star life cycle model shown in Fig. 3 [20]. While the waterfall model suggests a top-down (analytic) approach, the star model recognizes that this approach needs to be complemented by a bottom-up (synthetic) approach, and can start from any point in the star (as shown by the entry arrows), and followed by any other stage (as shown by the double arrows). In this way, the requirements, the design and the product gradually evolve, becoming step by step well defined.

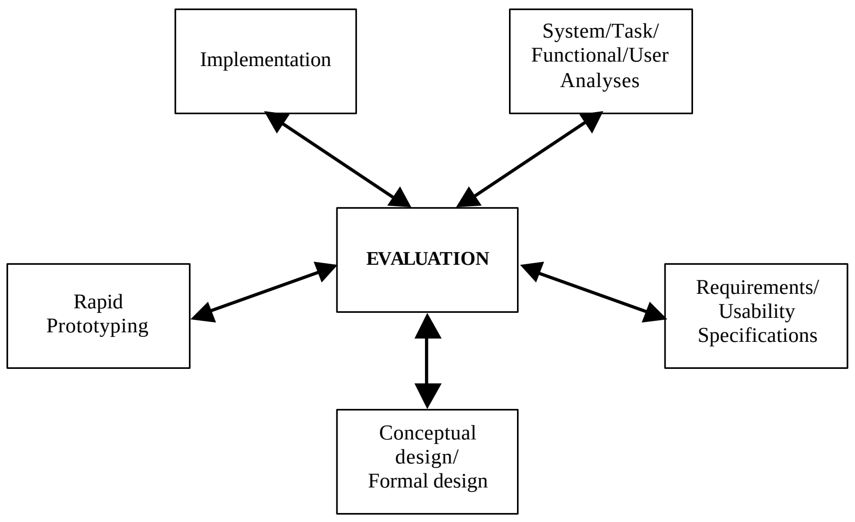
Implementation
System/Task/ Functional/User Analyses
EVALUATION
Rapid Prototyping
Requirements/ Usability Specifications
Conceptual design/ Formal design

Fig. 3. The star life cycle model [20].

5. Applying usability evaluation methods in the software life cycle

In order to design usable systems, we have seen that in the software life cycle usability evaluation plays the fundamental role. The HCI research has provided several principles and guidelines that can drive the designers in taking their decisions. However, applying available design guidelines is a good start, but there are no “cookbooks”, and therefore no substitute for system evaluation.

Different methods can be used for evaluating systems at the different phases of their development: the most commonly adopted are user-based methods and inspection methods. User-based methods mainly consist of user testing, in which usability properties are assessed by observing how the system, or a prototype of the system, is actually used by some representative of real users performing real tasks [15,16,21]. Usability inspection methods involve expert evaluators only, who inspect the user interface in order to find out possible usability problems, provide judgements based on their knowledge, and make recommendations for fixing the problems and improving the usability of the application [22].

User-based evaluation provides a trusty evaluation, because it assesses usability through samples of real users. However, it has a number of drawbacks, such as the difficulty to properly select a correct sample of the user community, and to train it to manage not only the main application features but also the most sophisticated and advanced facilities of an interactive system.

With respect to user-based evaluation, usability inspection methods are more subjective, having heavy dependence upon the inspector skills. Among the inspection methods, we may include heuristic evaluation, cognitive walkthrough, formal usability inspection, guidelines reviews [22]. Heuristic evaluation is the most informal method; it involves a usability expert who analyses the dialogue elements of the user interface to check if they conform to usability principles, usually referred as heuristics, hence the name of this method. In a cognitive walkthrough, the expert uses some detailed procedures to simulate users’ problem solving processes during the user-computer dialogue, in order to see if the functionalities provided by the system are efficient for users and lead the to correct actions. Formal usability inspection is a review of users’ potential task performance with a product. It was designed to help engineers to review a product and find a large number of usability defects. It is very similar to the code inspection methods with which software developers are familiar. It is carried out by the engineer designing the product and a team of peers, looking for defects. Finally, in a guidelines review, the expert inspects the interface to check if it is conform to a list of usability guidelines. The method can be considered as a cross between heuristic evaluation and standard inspection, the latter is another kind of inspection to check the compliance of the interface to some interface standards. A detailed description of these and other inspection methods is in [22].

The main advantage of inspection methods is however the cost saving: they do not involve users nor require any special equipment or lab facilities [8, 22]. In addition, experts can detect a wide range of problems and possible faults of a complex system in a limited amount of time. For these reasons, inspection methods have achieved widespread use in the last years, especially in the industrial environments [23], since industry is very much interested in effective methods, that can provide good results being still cost-effective.

Inspection methods aim at finding usability problems in an existing user interface, and make recommendations for fixing these problems. Hence, they can be applied at various steps of the software development, and are certainly used for evaluating the design of the system in a prototype form, even a paper prototype, so that possible defects can be fixed as soon as possible. When a system implementation is available, user-based evaluation is often recommended. It includes experimental methods, observational methods, and survey techniques. Among experimental methods, controlled experiments are very valuable, they provide empirical evidence to support specific hypotheses. They allow a comparative evaluation, very useful when alternative prototypes or versions of the same system are available. An experiment consists of the following steps: formulation of the hypotheses to be tested, definition of the experimental conditions that differ only in the values of some controlled variables, execution of the experiment, analysis of collected data. In order to verify the usability of a single prototype, we can observe users working with it. A valid technique is the thinking aloud, in which users are asked to think aloud when they use the system or prototype. In this way, evaluators can detect users misconceptions and the system elements that cause them. Both experimental and observational methods are used for collecting data about system and user performance; they do not provide data about users' satisfaction, that is a subjective measure that can be obtained by survey techniques, such as interviews and questionnaires that we mentioned in Section 3 since they are also used for requirement gathering. For further details, the reader is referred to [15, 16].

By considering the industry's interest for cheap but effective methods, heuristic evaluation plays an important role. It prescribes having a small set of experts analyzing the system, and evaluating its interface against a list of recognized usability principles, the heuristics. Some researches have shown that heuristic evaluation is a very efficient usability engineering technique [24], with a high benefit cost-ratio [23], and therefore it falls within the so-called discount usability methods.

In principle, only one evaluator can conduct heuristic evaluation. However, in an analysis of six studies, it has been assessed that single evaluators are able to find only the 35% of the total number of the existent usability problems [8,22]. Different evaluators tend to find different problems. Therefore, the more experts are involved in the evaluation, the more problems it is possible to find. The mathematical model defined in [12] shows that reasonable results can be obtained by having only five evaluators.

Heuristics evaluation, and in general the inspection methods proposed so far are not systematic, and lack methodological frameworks. In some recent papers, the User Action Framework (UAF) has been proposed, as a unifying and organizing framework, supporting usability inspection, design guidelines, classification and reporting of usability problems [25]. It provides a knowledge base, in which different usability problems are organized according to the Interaction Cycle model - an extension of the Norman's theory of actions [26]. Following such a model, problems are organized taking into account how user interaction is affected by the application design, at various points where users must accomplish cognitive or physical actions. The knowledge base is a notable contribution of the UAF methodology, which tries to provide a solution to the need for more focused usability inspection methods, and more readable and comparable inspection reports.

Classifying problems, and organizing them in a knowledge base, is also a way to keep track of the large number of problems found in different usability studies, thus capitalizing on past experiences. Reusing the past evaluation experience, and making it available to less experienced people is also one of the basic goals of a new inspection technique described in [27, 28], which introduces the use of evaluation patterns for guiding the inspector activity. The patterns precisely describe which objects to look for and which actions to perform in order to analyze such objects. Such evaluation patterns are called Abstract Tasks for the following reasons: *Task*, because each one describes a task the evaluator must perform during inspection in order to detect usability problems; *Abstract*, because it describes the operations for accomplishing the tasks, without any reference to a specific application, providing instead a description that is independent from any application. Their formulation is the reflection of the experiences of some skilled evaluators. They allow even less experienced evaluators to come out with good results. The major effectiveness and efficiency of the inspection based on the use of Abstract Tasks, with respect to the traditional heuristic evaluation, has been verified through a comparative study described in [29].

6. Conclusions

In the chapter we have stressed the importance of a good user interface for today's interactive systems. The quality of the interface must be evaluated especially from the users' point of view, by taking into account the users that will actually use the system. Usability, a quality that has been neglected by the software engineering community for too long time, becomes a crucial factor for judging the quality of the user interface. The user-centered design methodology for designing usable systems has been discussed, and indications on how to include usability in the software life cycle have been provided.

References

1. B. Myers and M.B. Rosson, Survey on User Interface Programming, *Proc. CHI'92, ACM Press*, 195-202.
2. J.A. McCall, Quality factors. In *Encyclopedia of Software Engineering* (John J. Marciniak, Ed), pp. 958-969, John Wiley & Sons, New York, 1994.
3. B. Boehm, *Characteristics of Software Quality*, North Holland Publishing Co., New York, 1978.
4. J. McCall, P. Richards, and G. Walters, *Factors in Software Quality,* 3 vol. NTIS AD-A049-015, 015, 055, Nov. 1977.
5. R.S. Pressman, *Software Engineering: A Practitioner' s Approach*. Fourth Edition, McGraw-Hill Companies, New York, 1997.
6. D.J. Mayhew, *Principles and Guidelines in Software User Interface Design,* Prentice Hall, Englewood Cliffs, 1992.
7. D.J. Mayhew, *The Usability Engineering Lifecycle: a Practitioner's Handbook for User Interface Design,* Morgan Kaufmann Publishers, California, 1999.
8. J. Nielsen. *Usability Engineering,* Academic Press, Cambridge, MA, 1993.
9. ISO/IEC (International Organization for Standardization and International Electrotechnical Commision), ISO/IEC 9126-1: Information Technology – Software Product Quality, 1998.
10. ISO (International Organization for Standardization), ISO 9241: Ergonomics Requirements for Office Work with Visual Display Terminal (VDT) - Parts 1-17, 1997.
11. K.H. Madsen, Special Issue on "The Diversity of Usability", *Communication of ACM*, **Vol. 42, No. 5**, 1999.
12. J. Nielsen and T.K. Landauer, A Mathematical Model of The Finding of Usability Problems, *Proc. ACM INTERCHI'93 – International Conference on Human Factors in Computing Systems, ACM*

Press, Amsterdam, NL, 296-213, 1993.

13. J. Hackos and J.C. Redish, *User and Task Analysis for Interface Design*, John Wiley & Sons, 1998.
14. J. Rubin, *Handbook of Usability Testing*, John Wiley & Sons, 1994.
15. Dix, J. Finlay, G. Abowd and R. Beale, *Human Computer Interaction*, Prentice Hall, 1998.
16. J. Preece, Y. Rogers, H. Sharp, D. Beyon, S. Holland and T. Carey, *Human-Computer Interaction*, Addison Wesley, 1994.
17. F. Paterno', Task Models for Interactive Software Systems, In this volume.
18. C. Ghezzi, M. Jazayeri and D. Mandrioli, *Fundamentals of Software Engineering*, Prentice Hall International, 1991.
19. ISO (International Organization for Standardization), ISO 13407: Human-Centered Design Process for Interactive Systems, 1998.
20. H.R. Hartson and D. Hix, *Developing User Interfaces*, John Wiley, New York, 1993.
21. J. Whiteside, J. Bennet and K. Holtzblatt, Usability Enginnering Our Experience and Evolution, In *Handbook of Human-Computer Interaction*, (ed. M. Helander), Elsevier Science, 791-817, 1988.
22. J. Nielsen and R.L. Mack, *Usability Inspection Methods*, John Wiley & Sons, New York, 1994.
23. J. Nielsen, Guerrilla HCI: Using Discount Usability Engineering to Penetrate Intimidation Barrier, In *Cost-Justifying, Usability* (R.G. Bias, D.J. Mayhew, Eds) Academic Press, Also available at the URL *http://www.useit.com/papers/guerilla_hci.html*, 1994.
24. R. Jeffries and H.W. Desurvire, Usability Testing vs. Heuristic Evaluation: Was There a Context? *ACM SIGCHI Bulletin*, **Vol. 24 No. 4**, 39-41, 1992.
25. H.R. Hartson, T.S. Andre, R.C. Willges and L.Van Rens, The User Action Framework: A Theory-based Foundation For Inspection and Classification of Usability Problems, *Proc. HCII '99, Munich, Germany*, August 1999, vol. I, 1058-1062.
26. D.A. Norman, *The Psychology of Everyday Things*, Basic Books, 1988.
27. M. Matera, SUE: A Systematic Methodology for Evaluating Hypermedia Usability, Ph.D. Thesis, Dipartimento di Elettronica e Informazione, Politecnico di Milano, 2000.
28. M.F. Costabile and M. Matera, Proposing Guidelines for Usability Inspection, *Proc. TFWWG'2000, Biarritz, France*, October 7-8, 2000, Spinger Verlag, 283-292.
29. De Angeli, M. Matera, M.F. Costabile, F. Garzotto, and P. Paolini, Evaluating the SUE Inspection Technique, *Proc. AVI'2000 – International Conference on Advanced Visual Interfaces, Palermo, Italy, May 24-27, 2000, ACM Press*, 143-150.