Usability Basics for Software Developers

Xavier Ferré and Natalia Juristo, *Universidad Politécnica de Madrid*
Helmut Windl, *Siemens AG, Germany*
Larry Constantine, *Constantine & Lockwood*

In recent years, software system usability has made some interesting advances, with more and more organizations starting to take usability seriously.[1] Unfortunately, the average developer has not adopted these new concepts, so the usability level of software products has not improved.

This tutorial examines the relationship between usability and the user interface and discusses how the usability process follows a design-evaluate-redesign cycle. It also discusses some management issues an organization must face when applying usability techniques.

Contrary to what some might think, usability is not just the appearance of the user interface (UI). Usability relates to how the system interacts with the user, and it includes five basic attributes: learnability, efficiency, user retention over time, error rate, and satisfaction. Here, we present the general usability process for building a system with the desired level of usability. This process, which most usability practitioners apply with slight variation, is structured around a design-evaluate-redesign cycle. Practitioners initiate the process by analyzing the targeted users and the tasks those users will perform.

Clarifying usability concepts

According to ISO 9241, Part 11, usability is "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."[2] This definition ties a system's usability to specific conditions, needs, and users—it requires establishing certain levels of usability based on the five basic attributes.

Usability engineering defines the target usability level in advance and ensures that the software developed reaches that level. The term was coined to reflect the engineering approach some usability specialists take.[3] It is "a process through which usability characteristics are specified, quantitatively and early in the development process, and measured throughout the process."[4] Usability is an issue we can approach from multiple viewpoints, which is why many different disciplines, such as psychology, computer science, and sociology, are trying to tackle it. Unfortunately, this results in a lack of standard terminology. In fact, the term usability engineering is not universally accepted—other terms used include usage-centered design, contextual design, participatory design, and goal-directed design. All these philosophies adhere to some extent to the core issue of usability engineering: evaluating usability with real users from the first stages of development.

Usability attributes

We can't define usability as a specific as-

0740-7459/00/$10.00 © 2001 IEEE

pect of a system. It differs depending on the intended use of the system under development. For example, a museum kiosk must run a software system that requires minimum training, as the majority of users will use it just once in their lifetime. Some aspects of usability—such as efficiency (the number of tasks per hour)—are irrelevant for this kind of system, but ease of learning is critical. However, a bank cashier's system would require training and would need to be highly efficient to help reduce customer queuing time.

Because usability is too abstract a term to study directly, it is usually divided into the attributes we mentioned at the beginning of the article:[5]

- *Learnability*: How easy it is to learn the main system functionality and gain proficiency to complete the job. We usually assess this by measuring the time a user spends working with the system before that user can complete certain tasks in the time it would take an expert to complete the same tasks. This attribute is very important for novice users.
- *Efficiency*: The number of tasks per unit of time that the user can perform using the system. We look for the maximum speed of user task performance. The higher system usability is, the faster the user can perform the task and complete the job.
- *User retention over time*: It is critical for intermittent users to be able to use the system without having to climb the learning curve again. This attribute reflects how well the user remembers how the system works after a period of nonusage.
- *Error rate*: This attribute contributes negatively to usability. It does not refer to system errors. On the contrary, it addresses the number of errors the user makes while performing a task. Good usability implies a low error rate. Errors reduce efficiency and user satisfaction, and they can be seen as a failure to communicate to the user the right way of doing things.
- *Satisfaction*: This shows a user's subjective impression of the system.

One problem concerning usability is that these attributes sometimes conflict. For example, learnability and efficiency usually influence each other negatively. A system must be carefully designed if it requires both high learnability and high efficiency—for example, using accelerators (a combination of keys to perform a frequent task) usually solves this conflict. The point is that a system's usability is not merely the sum of these attributes' values; it is defined as reaching a certain level for each attribute.

We can further divide these attributes to precisely address the aspects of usability in which we are most interested. For example, *performance in normal use* and *advanced feature usage* are both subattributes of efficiency, and *first impression* is a subattribute of satisfaction. Therefore, when analyzing a particular system's usability, we decompose the most important usability attributes down to the right detail level.

Usability is not only concerned with software interaction. It is also concerned with help features, user documentation, and installation instructions.

Usability and the user interface

We distinguish between the visible part of the UI (buttons, pull-down menus, checkboxes, background color, and so forth) and the interaction part of the system to understand the depth and scope of a system's usability. (By interaction, we mean the coordination of the information exchange between the user and the system.) It's important to carefully consider the interaction not just when designing the visible part of the UI, but also when designing the rest of the system.

For example, if a system must provide continuous feedback to the user, the developers need to consider this when designing the time-consuming system operations. They should design the system so it can frequently send information to the UI to keep the user informed about the operation's current status. The system could display this information as a percentage-completed bar, as in some software installation programs.

Unfortunately, it is not unusual to find development teams that think they can design the system and then have the "usability team" make it usable by designing a nice set of controls, adding the right color combination, and using the right font. This approach is clearly wrong. Developers must consider user interaction from the beginning of the development process. Their understanding of the interaction will affect the final product's usability.

It's important to carefully consider the interaction not just when designing the visible part of the user interface, but also when designing the rest of the system.

Usability testing alone is not enough to output a highly usable product, because it uncovers but does not fix design problems.

Usability in software development

The main reason for applying usability techniques when developing a software system is to increase user efficiency and satisfaction and, consequently, productivity. Usability techniques, therefore, can help any software system reach its goal by helping the users perform their tasks. Furthermore, good usability is gaining importance in a world in which users are less computer literate and can't afford to spend a long time learning how a system works. Usability is critical for user system acceptance: If users don't think the system will help them perform their tasks, they are less likely to accept it. It's possible they won't use the system at all or will use it inefficiently after deployment. If we don't properly support the user task, we are not meeting user needs and are missing the main objective of building a software system.

For a software development organization operating in a competitive market, failure to address usability can lead to a loss of market share should a competitor release a product with higher usability. Also, a software product with better usability will result in reduced support costs (in terms of hotlines, customer support service, and so forth).

Even if a system is being used, it does not necessarily mean it has a high level of usability. There are other aspects of a software product that condition its usage, such as price, possibility of choice, or previous training. In addition, because users are still more intelligent than computers, it is usually the human who adapts to the computer in human–computer interaction. However, we shouldn't force the user to adapt to software with poor usability, because this adaptation can negatively influence efficiency, effectiveness, and satisfaction. Usability is a key aspect of a software product's success.

The usability process

As we mentioned, a system's usability depends on the interaction design. Therefore, we must deal with system usability throughout the entire development process. Usability testing alone is not enough to output a highly usable product, because usability testing uncovers but does not fix design problems. Furthermore, usability testing has been viewed as similar to other types of software quality assurance testing, so developers often apply the techniques late in the development cycle—when major usability problems are very costly, if not impossible, to fix. Therefore, it is crucial to evaluate all results during the product development process, which ultimately leads to an iterative development process. A pure waterfall approach to software development makes introducing usability techniques fairly impossible.

All software applications are tools that help users accomplish certain tasks. However, before we can build usable software tools—or, rather, design a UI—we need information about the people who will use the tool:

- Who are the system users?
- What will they need to accomplish?
- What will they need from the system to accomplish this?
- How should the system supply what they need?

The usability process helps user interaction designers answer these questions during the analysis phase and supports the design in the design phase (see Figure 1).

There are many usability methods—all essentially based on the same usability process—so we have abstracted a generic usability process from the different approaches to usability mentioned earlier. We hope this makes it easier for the reader to understand the different usability techniques we will be describing.

Usability analysis phase

First, we have to get to know the users and their needs, expectations, interests, behaviors, and responsibilities, all of which characterize their relationship with the system.

User analysis. There are numerous approaches for gathering information about users, depending on each individual system under development and the effort or time constraints for this phase. The main methods are *site visits*, *focus groups*, *surveys*, and *derived data*.

The primary source for user information is site visits. Developers observe the users in their working environment, using the system to be replaced or performing their tasks manually if there is no existing tool. In addition, developers interview the users to understand their motivation and the strategy behind their actions. A well-known method

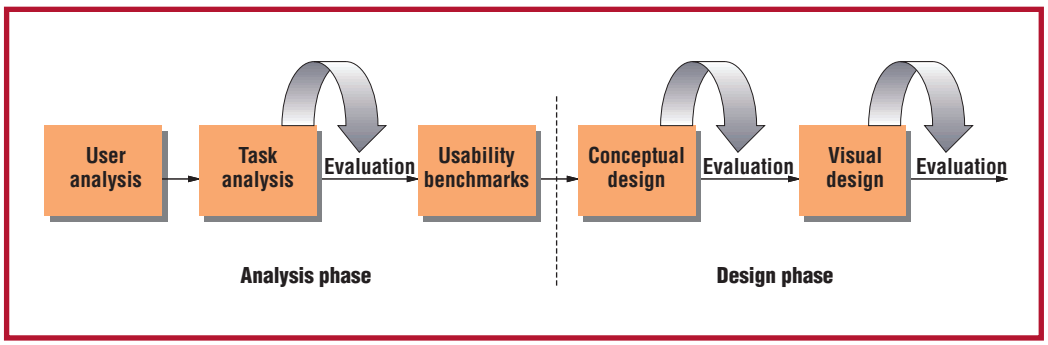


Figure 1. The usability process.

for doing user analysis jointly with task analysis is *contextual inquiry*.[6] This method provides a structured way for gathering and organizing information.

A focus group is an organized discussion with a selected group of users. The goal is to gather information about their views and experiences concerning a topic. It is well suited for getting several viewpoints about the same topic—for example, if there is a particular software product to discuss—and gaining insight into people's understanding of everyday system use.

In a survey, the quality of the information depends on the quality of the questions. Surveys are a one-way source, because it is often difficult or even impossible to check back with the user. Don A. Dillman's book *Mail and Internet Surveys* provides a structured method for planning, designing, and conducting surveys.[7]

Derived data includes hotline reports, customer complaint letters, and so forth. It can be a good source of usability implications but is often difficult to interpret. The most important limitation is that such sources are one-sided. They report only problems and say nothing about the features that users liked or that enabled efficient use.

The most important thing about user analysis is to record, structure, and organize the findings.

Task analysis. Task analysis describes a set of techniques people use to get things done.[8] The concept of a task is analogous to the concept of a use case in object-oriented software development; a task is an activity meaningful to the user. User analysis is taken as input for task analysis, and both are sometimes performed jointly.

We analyze tasks because we can use the located tasks to drive and test UI design throughout the product development cycle. Focusing on a small set of tasks helps rationalize the development effort. Therefore, we suggest prioritizing the set of tasks by importance and frequency to get a small task set. This approach guarantees that you'll build the most important functionalities into the system and that the product will not suffer from "featuritis." These tasks should be the starting point for developing the system. One approach to analysis is to build a task model within the Usage-Centered Design method, a model-driven approach for designing highly usable software applications, where tasks, described as essential use cases, are the basis for a well-structured process and drive UI design.[9]

Task analysis ends when we evaluate the discovered task set, which is best done collaboratively with users. When the user population is already performing a set of tasks, we perform task analysis during user analysis to apprehend the tasks the user performs routinely and how the user perceives these tasks. After the optional first analysis, we identify the tasks our system will support, based on a study of the goals the user wants to attain. Then, we break the tasks into subtasks and into particular actions that the user will perform and take the identified tasks as the basis for building the usability specifications. We then instantiate them to real-world examples and present them to test participants in a usability test.

Usability benchmarks. We set usability benchmarks as quantitative usability goals, which are defined before system design begins.[10] They are based on the five basic usability attributes or their subattributes.

We need these benchmarks because, if we want to assess the value of the usability attributes for the system under development, we need to have a set of operationally defined usability benchmarks.

We establish usability benchmarks by defining a set of benchmarks for each usability attribute we want to evaluate—that is, for each usability attribute we consider important for our system. We must define the benchmarks in a way that makes them calcu-

Table 1

A Sample Usability Specification Table[4]

Usability attribute	Measuring instrument	Value to be measured	Current level	Worst acceptable level	Planned target level	Best possible level	Observed results
Performance in normal use	"Answer request" task	Length of time taken to successfully perform the task (minutes and seconds)	2 min, 53 sec	2 min, 53 sec	1 min, 30 sec	50 sec	
First impression	Questionnaire	Average score (range –2 to 2)	—	0	1	2	

lable in a usability test or through a user satisfaction questionnaire. Table 1 shows the format of a usability specification table. (The "Observed results" column is filled with the data gathered during the usability tests.)

We take task analysis as an input for this activity, because most usability benchmarks are linked to a task specified in task analysis.

Usability design

Once we have analyzed the tasks our system will support, we can make a first attempt at the UI's conceptual design, which we will evaluate and possibly improve in the next iteration.

Conceptual design. During the conceptual design phase, we define the basic user–system interaction and the objects in the UI and the contexts in which interaction takes place. The findings of the user and task analysis are the basis for the conceptual design. The deliverables from this phase are typically paper prototypes, such as pencil drawings or screen mockups, and a specification, which describes the UI's behavior.

Conceptual design is the most crucial phase in the process, because it defines the foundation for the entire system. Unfortunately, design is a very creative process, and it can't be automated with a method. There is a set of design principles and rules that we must creatively adapt for a certain design problem. (A good reading for any designer—not just software designers—is *The Design of Everyday Things*,[11] which presents general design principles by evaluating the design of everyday objects.)

The main principles of UI design cover feedback, reuse, simplicity, structure, tolerance, and visibility in UIs. Knowing usability design principles is the basis for good design. Compare this to an adult drawing class. Not everyone will be Picasso by the end of the course, but the students will be able to paint reasonable pictures if they use the principles they learned. Another way to improve design ability is to examine UIs. Analyzing the UIs of every software application you can access is very helpful and can sometimes be a source of inspiration for finding innovative, alternative solutions.

The conceptual design phase also ends with evaluating the results. It is a good idea to test the paper prototypes against the defined task set to check that all the prioritized tasks can be enacted. The last test in this phase is run together with users as a usability test or usability inspection of the paper prototype.

Visual design. Having completed the conceptual design, the final step in our process is visual design, where we define the UI's appearance. This covers all details, including the layout of screens and dialog boxes, use of colors and widgets, and design of graphics and icons. There are also rules and principles for visual design, addressing use of color, text, screen layout, widget use, icon design, and so forth. It pays to have a professional screen designer, especially in this phase. Recommended readings about visual and conceptual design are *About Face*[12] and *Software for Use*,[9] which both include numerous design tips. *Designing Visual Interfaces* focuses on screen design and graphics design in the context of UIs, as well as the underlying principles of visual design.[13]

The deliverables of this phase are prototypes that must be tested, an exact specification of the UI appearance, and behavior plus the specification for new widgets that must be developed.

Prototyping

Prototypes are not exclusive to UI design, but they are valuable for performing usability testing in early development phases. We need to build prototypes because abstract technical specifications are not a good way of

communicating when we want to involve users in the design process—users understand tangible system prototypes much better.[5]

Some prototyping techniques help perform usability testing and require little implementation effort. We create prototypes to test them on the user through usability evaluation techniques. The prototyping techniques with which software developers usually are not familiar include

- *Paper mock-ups*: At the beginning of the design process, the designer creates paper prototypes—usually pencil drawings or printouts of screen designs—for the user. The designer will act as the computer, showing the user the next element when a transition between graphical elements occurs.[8]
- *"Wizard of Oz" technique*:[8] A human expert acts as the system and answers the user's requests, without the user's knowledge. The user interacts normally with the screen, but instead of using software, a developer sits at another computer (network-connected to the user's computer) answering the queries. The user gets the impression of working with a real software system, and this method is cheaper than implementing a real software prototype.
- *Scenarios, storyboards, and snapshots*: A scenario describes a fictional story of a user interacting with the system in a particular situation; snapshots are visual images that capture the interaction occurring in a scenario; and storyboards[8] are sequences of snapshots that focus on the main actions in a possible situation. They make the design team think about the appropriateness of the design for a real context of use, and they help make the process user-centric.

Usability evaluation

Usability evaluation is a central activity in the usability process. It can determine the current version's usability level and whether the design works.

Usability testing. The term *usability testing* describes the activity of performing usability tests in a laboratory with a group of users and recording the results for further analysis. We can't predict a software system's usability without testing it with real users.

First, we must decide which groups of users we want to use to test the system and how many from each group we will try to recruit as test participants. Then, we must design the test tasks we'll ask the participants to perform. We usually take them from the results of the task analysis activity and apply them to hypothetical real-life situations. Some characteristics of the test require consideration, such as

- whether the participant can ask the evaluator for help;
- should two participants jointly perform each test task to observe the remarks they exchange in the process;
- what information participants will receive about the system prior to the test; and
- whether to include a period of free system access after completing the predefined tasks to get the user's overall impression of the system.

After we prepare the test and recruit test participants, we run the tests, optionally recording them with video cameras or audio recorders, and log the users' actions in the system for further analysis (also optional). Once we have performed all the tests, we analyze the data and gather results to apply them in the next iterative cycle.

Thinking aloud. Formative evaluation seeks to learn which detailed aspects of the interaction are good and how to improve the interaction design.[5] This opposes summative evaluation, which is performed at the end of the development process, after the system has been built. The results of summative evaluation do not help shape the product.

Thinking aloud helps perform formative evaluation in usability tests. We ask the test participant to think aloud while using the system in a usability test,[8] to verbalize his or her actions so we can collect the remarks. For example, a participant might say, "First, I open the file, and I click once on the file icon. Nothing happens. I don't know why this is not working like the Web. I press the Enter key, and it opens. Now I want to change the color of the label, so I search in the Tools menu, but I can't find any option for what I want to do." User remarks obtained in usability tests can provide signifi-

We can't predict a software system's usability without testing it with real users.

Further Reading

The following are books about human–computer interaction (HCI) and usability that are more likely to interest software practitioners with little or no knowledge about the field (the information for each book is in the References section of this article).

B. Shneiderman, *Designing the User Interface: Strategies for Effective Human–Computer Interaction*—This book summarizes all aspects related to interactive systems from a serious scientific viewpoint, although some readers might prefer a more engineering-focused approach. It includes a valuable set of guidelines for designing the user interface. There have been some interesting additions in the third edition about issues such as hypermedia and the Web and Computer Supported Cooperative Work.

D. Hix and H.R. Haertsen, *Developing User Interfaces: Ensuring Usability Through Product and Process*—Despite its title, this book is not just about the user interface; it focuses on the process of user interaction design. Written in a very practical style, it provides a hands-on approach to designing the interactive part of a software system. Software practitioners might want to skip the chapters devoted to the User Action Notation—a technique for representing interaction designs—which is too formal for non-HCI experts.

L.L. Constantine and L.A.D. Lockwood, *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*—The most recent of the books reviewed here, it presents a process for designing usable software systems based on one of the current trends in software engineering: use cases. The book is written in a practical style that is likely to appeal to software practitioners.

J. Nielsen, *Usability Engineering*—This book provides a good introduction to the issue of usability engineering. It is easy to read and includes stories of real situations. It deals with a wide variety of issues related to usability engineering—but none are addressed in depth.

cant insight into the best way of designing the system interaction. By detailing their mental process, test participants can uncover hidden usability problems.

Formative evaluation is the usual form of evaluation in a usability process, combining qualitative data gathered from user comments with quantitative data to check against previously defined usability benchmarks.

Heuristic evaluation. A usability expert can perform a heuristic evaluation of the system to make some development iterations shorter and to perform more iterations in the development process. The expert will make a critique founded on both his or her interaction design experience and on generally accepted usability guidelines, like the ones by Ben Shneiderman[14] and Jakob Nielsen.[5]

Experts provide a different kind of feedback than final users through usability testing. Expert suggestions for modification are usually more applicable, and they are more precise about the underlying usability problems, such as a lack of consistency or poor navigation. On the other hand, usability testing must be performed with real users to identify specific usability problems. Heuristic evaluation can complement but not replace usability testing.

Collaborative usability inspection. A collaborative usability inspection is a systematic examination of a finished system, design or prototype from the end user's viewpoint.[9] A team of developers, end users, application or domain experts, and usability specialists collaboratively perform the review. Collaborative usability inspections (CUIs) use features and techniques from heuristic evaluation, pluralistic usability walkthroughs, and expert evaluations and are less expensive and faster than usability testing. Behind this technique is a set of strict rules to avoid the problems that typically arise if end users discuss their work together with designers or developers. CUIs uncover more—albeit different—usability defects (up to 100 defects per hour) than usability testing.

Apart from efficiency, one advantage is that people with multiple perspectives and expertise examine the test object. Another advantage is that the participating developers build skills and know-how about how to make software more usable.

Management and organizational issues

When introducing usability, an organization must first commit management to the ideas behind the usability process and convince them of its benefits.[4,5] The newest concepts they need to accept include creating conceptual design in the first stages of development and evaluating usability throughout the development process. *Cost-Justifying Usability* presents cost-benefit arguments in favor of performing usability practices, which can be used when trying to get management commitment.[15] Another option to convince management is to take a recently developed system or one that is currently being developed and to perform videotaped usability tests with a few users who are novel to the system. Showing the results to management and the development team can produce a change of attitude to-

ward usability testing, as the results will probably show that the system is not as good in usability terms as expected.

Integrating UI designers into the development team isn't always easy, especially if they are assigned to several projects at the same time. One approach to applying usability techniques in some projects is to promote one member of each development team to usability champion,[5] similar to process improvement champions. Usability champions learn the basic usability skills and are coordinated by a user interaction designer. The user interaction designer then acts as a consultant in several projects but can interact with the usability champion in each group.[4]

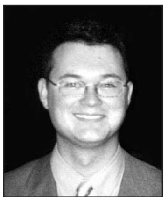
Don't try to do a full-scale usability process from the beginning. You can start by setting a small set of usability specifications with a simple task analysis of the most prominent tasks, some conceptual design with paper prototypes and simple usability tests to be carried out with a small set of users. You can also act as usability expert performing heuristic evaluation on the system using the guidelines we mentioned (by Shneiderman[14] and Nielsen[5]). Starting with modest objectives will contribute more firmly to the final success of your endeavor.

Despite increasing usability awareness in software development organizations, applying usability techniques in software development is not easy. Software engineers and usability engineers have a different conception of software development, and conflicts can arise between them due to differences in terminology and procedures. To create acceptable usability concepts, the software engineering community must integrate usability techniques into a software engineering process that is recognizable from both fields. Use cases offer a good starting point, as they are the software engineering construct closer to a usable software development approach.

References

1. L. Trenner and J. Bawa, *The Politics of Usability*, Springer-Verlag, London, 1998.
2. *Ergonomic Requirements for Office Work with Visual Display Terminals*, ISO 9241-11, ISO, Geneva, 1998.
3. M. Good et al., "User-Derived Impact Analysis as a Tool for Usability Engineering," *Proc. CHI Conf. Human Factors in Computing Systems*, ACM Press, New York, 1986, pp. 241–246.
4. D. Hix and H.R. Hartson, *Developing User Interfaces: Ensuring Usability Through Product and Process*, John Wiley & Sons, New York, 1993.
5. J. Nielsen, *Usability Engineering*, AP Professional, Boston, Mass., 1993.
6. H. Beyer and K. Holtzblatt, *Contextual Design: A Customer-Centered Approach to Systems Design*, Morgan Kaufmann, San Francisco, 1997.
7. D.A. Dillman, *Mail and Internet Surveys: The Tailored Design Method*, John Wiley & Sons, New York, 1999.
8. J. Preece et al., *Human-Computer Interaction*, Addison-Wesley Longman, Reading, Mass., 1994.
9. L.L. Constantine and L.A.D. Lockwood, *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Longman, Reading, Mass., 1999.
10. J. Whiteside, J. Bennett, and K. Holtzblatt, "Usability Engineering: Our Experience and Evolution," *Handbook of Human-Computer Interaction*, Elsevier North-Holland, Amsterdam, 1988.
11. D.A. Norman, *The Design of Everyday Things*, Doubleday, New York, 1990.
12. A. Cooper, *About Face: The Essentials of User Interface Design*, IDG Books Worldwide, Foster City, Calif., 1995.
13. K. Mullet and D. Sano, *Designing Visual Interfaces: Communication Oriented Techniques*, Prentice Hall, Upper Saddle River, N.J., 1994.
14. B. Shneiderman, *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Addison-Wesley Longman, Reading, Mass., 1998.
15. R.G. Bias and D.J. Mayhew, *Cost-Justifying Usability*, Academic Press, Boston, Mass., 1994.

About the Authors



Xavier Ferré is an assistant professor of software engineering at the Universidad Politécnica de Madrid, Spain. His primary research interest is the integration of usability techniques into software engineering development practices. He has been a visiting PhD student at CERN (European Laboratory for Particle Physics) and at the HCIL (Human–Computer Interaction Laboratory) at the University of Maryland. He received an MS in computer science from the Universidad Politécnica de Madrid. He is a member of the ACM and its SIGCHI group. Contact him at xavier@fi.upm.es.

Natalia Juristo is a full professor in the Computer Science Department at the Universidad Politécnica de Madrid, where she directs master's-level courses in knowledge engineering and software engineering. She is also an editorial board member of *IEEE Software* and the *International Journal on Software Engineering and Knowledge Engineering*. She has a BS and PhD in computer science from the Technical University of Madrid. She is a senior member of the IEEE Computer Society and a member of the ACM, the American Association for the Advancement of Science, and the New York Academy of Sciences. Contact her at the Facultad de Informática UPM, Campus de Montegancedo, s/n, Boadilla del Monte, 28660 Madrid, Spain; natalia@fi.upm.es.

Helmut Windl leads the User Interface Design Group for Simatic Automation Software at Siemens' Automation & Drives Division, where he has helped define and implement a structured usability process within the software development process. He is an experienced user-interface and visual designer for large-scale software applications and a project leader for usability-focused products. He is also a trainer and presenter in Siemens AG and with Constantine & Lockwood. He received a diploma in electrical engineering from the University of Applied Sciences Regensburg. Contact him at Siemens AG, A&D AS S8, PO Box 4848, D-90327 Nuremberg, Germany; helmut.windl@nbgm.siemens.de.

Larry Constantine is director of research and development at Constantine & Lockwood, a training and consulting firm. He is also an adjunct professor in the School of Computing Sciences at the University of Technology, Sydney, where he teaches software engineering and managing organizational change, and he is on the faculty of the Cutter Consortium. He has authored or coauthored 10 books, including *Software for Use: A Practical Guide to the Methods and Models of Usage-Centered Design* (Addison Wesley Longman, 1999). Contact him at Constantine & Lockwood, 58 Kathleen Circle, Rowley, MA 01969; larry@foruse.com; www.foruse.com.