Guidelines for Eliciting Usability Functionalities

Natalia Juristo, Ana Maria Moreno, and Maria-Isabel Sanchez-Segura

Abstract—Like any other quality attribute, usability imposes specific constraints on software components. Features that raise the software system's usability have to be considered from the earliest development stages. But, discovering and documenting usability features is likely to be beyond the usability knowledge of most requirements engineers, developers, and users. We propose an approach based on developing specific guidelines that capitalize upon key elements recurrently intervening in the usability features elicitation and specification process. The use of these guidelines provides requirements analysts with a knowledge repository. They can use this repository to ask the right questions and capture precise usability requirements information.

Index Terms—Usability requirements, usability features elicitation, requirements elicitation.

1 Introduction

Usability is a quality attribute found in most classifications [26], [27], [8]. It is the extent to which specific users can use a product to their satisfaction in order to effectively and efficiently achieve specific goals in a specific context of use [30]. Usability is a critical aspect in interactive software systems [14], [48], offering important cost savings and revenue increases [18], [43], [13].

For the past two decades, software usability has been perceived in software development as related to the presentation of information to the user [47], [21]. Software engineers have treated usability primarily by separating the presentation portion from the system functionality as recommended by generally accepted design strategies (e.g., MVC or PAC [11]). This separation makes it easier to modify the user interface and improve usability without affecting the rest of the application. Consequently, there is a belief that usability can be considered late in the development process (generally after testing) as it should not take too much rework to improve this quality attribute.

Recently, usability's implications for the application core have been explicitly highlighted from a software engineering (SE) perspective. Perry and Wolf have claimed that usability issues place static and dynamic constraints on the software components [44]. Bass et al. [2], [3] have used a bottom-up approach based on fieldwork observation to describe a set of scenarios representing usability issues that have an effect on the software architecture. We[1] have decomposed usability into features and examined which ones have an impact on software architecture [1]. The implications of the usability features for design materialize as the creation of special items (components, responsibilities, interactions, classes, methods, etc.) that affect both the presentation and application layer of the software system architecture. Therefore, addressing usability features at the end of the construction process will involve major rework. To avoid this, it has been suggested that software usability should be dealt with proactively at the architectural design stage instead of retroactively after testing [4], [10], [21], [35]. We think this should be taken a step further and usability should be brought forward in the development process and considered at the requirements stage. Addressing usability at the requirements stage has the same benefits as considering other quality attributes early on in the development process [5]: "The earlier key quality attribute requirements are identified and prioritized, the more likely it is that the essential quality attributes will be built into the system. It is more cost-effective to reason about quality attribute trade-offs early in the lifecycle than later in the lifecycle when modifications are often difficult, impractical, or even impossible."

Building usability into a software system has a cost and calls for negotiation with users and other stakeholders about which usability features should be included, the consequences of their inclusion, how to provide them, etc. As applied to other quality attributes, it is more cost-effective to reason about usability trade-offs early in the lifecycle.

This paper focuses on particular usability features with high functional implications and discusses how to deal with them at the requirements stage. To do this, Section 2 addresses what we have called functional usability features. Section 3 presents some completeness problems caused by the incorporation of functional usability features as requirements and discusses how the traditional approaches for dealing with incompleteness are hard to apply in this case. Section 4 presents the approach we followed to avoid such problems. Section 5 discusses the pattern-oriented solution we propose. Finally, Section 6 shows some results related to pattern use.

1. EU funded STATUS Project IST-2001-32298 available at http://is.ls.fi.upm.es/status/.

• N. Juristo and A.M. Moreno are with the School of Computing, Universidad Politécnica de Madrid, Campus de Montegancedo s/n, 28660 Boadilla del Monte, Madrid, Spain.
E-mail: {natalia, ammoreno}@fi.upm.es.
• M.-I. Sanchez-Segura is with the Computer Science Department, Universidad Carlos III de Madrid, Avda. De la Universidad, 30, 28911 Leganés, Madrid, Spain. E-mail: misanche@inf.ucsm.es.

*Manuscript received 21 Aug. 2006; revised 29 Jan. 2007; accepted 30 Aug. 2007; published online 10 Sept. 2007.
Recommended for acceptance by A. Anton.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-0199-0806.
Digital Object Identifier no. 10.1109/TSE.2007.70741.*

TABLE 1
A Preliminary List of Functional Usability Features

Usability features with design implications [2][3][33][34]	Description	Usability benefits
Feedback	Keep the user informed of what is going on inside the system	Nielsen [41], Constantine [14], Shneiderman [48], Hix [23]
Undo Cancel	Allow the user to cancel completed or ongoing operations	Nielsen [41], Hix [23], Shneiderman [48], Constantine [14]
User input errors prevention/correction	To improve data input for users and software correction as soon as possible	Shneiderman [48], Hix [23], Constantine [14]
Wizard	To help users to do tasks involving more than one step	Constantine [14]
User profile	To allow the user to tailor the application to user features	Hix [23]
Help	To provide useful help for users on how to do tasks	Nielsen [41], Constantine [14]
Command aggregation	To help the user to create commands to execute more than one task at a time	Nielsen [41], Constantine [14], Hix [23]
Shortcuts (key and tasks)	To allow users to activate a task with one quick gesture.	Nielsen [41], Constantine [14], Shneiderman [48], Hix [23]
Reuse information	To allow the user to easily move data from one part of a system to another	Constantine [14]

2 Functional Usability Features

Both the Human Computer Interaction (HCI) [29], [32] and SE [50] disciplines deal with usability as a nonfunctional requirement. Usability requirements specify user effectiveness, efficiency, or satisfaction levels that the system should achieve. These specifications are then used as a yardstick at the evaluation stage: "A novice user should learn to use the system in less than 10 hours" or "rnd user satisfaction with the application should be higher than Z on a 1-to-5 scale." Dealing with usability in the shape of nonfunctional requirements does not provide developers with enough information about what kind of artifacts to use to satisfy such requirements. Recent studies have targeted the relationship between usability and functional requirements. Cysneiros et al. suggest identifying functional requirements that improve particular usability attributes [16]. We propose a complementary approach in which usability features with major implications for software functionality are incorporated as functional requirements. We have termed these features functional usability features.

The usability literature has provided an extensive set of guidelines to help developers to build usable software. Each author has named these guidelines differently: design heuristics [41], principles of usability [14], [48], usability guidelines [23], etc. Although all of these recommendations share the same goal of improving software system usability, they are very different from each other. For example, there are very abstract guidelines like "prevent errors" [41] or "support internal locus of control" [48], [23], and others that provide more definite usability solutions like "make the user actions easily reversible" [23] or "provide clearly marked exits" [41].

To identify a preliminary list of functional usability features, we took the usability features with relevant usability benefits (according to the usability literature) and with strong design implications (according to the STATUS project [33], [34] and Bass et al. [2], [3]). Table 1 shows the result of this process. The list of usability features in Table 1 is not intended to be exhaustive; these features are a starting point for identifying usability features with an impact on software system functionality. They are a good example of usability features that should be considered at the requirements stage.

The features described in Table 1 represent particular functionalities that can be built into a software system to increase usability. Since functional requirements describe the functions that the software is to execute [50], we consider that the usability features in Table 1 should be treated as functional requirements (even though they are usability-related requirements). Such functional usability requirements need to be explicitly specified, just like any other functionality. If these usability functionalities are properly described in the requirements specification, they are more likely to be built into the system. They will improve the system's usability and contribute to the usability levels established in the nonfunctional requirements.

We will see in the next section that properly specifying functional usability features is not void of difficulties.

3 Difficulties of Incorporating Usability into Functional Requirements

Usability functionalities could be specified by just stating the respective usability features. For example, "the system should provide users with the ability to cancel actions" or "the system should provide feedback to the user." This is actually the level of advice that most HCI heuristics provide. The HCI community assumes that this level of detail is sufficient for developers to properly build a usability feature into the system. For example, one of the most commonly recurring HCI guidelines is Nielsen's feedback heuristic: "The system should always keep users informed about what is going on through appropriate feedback within reasonable time" [41]. However, this description provides nowhere near enough information to satisfactorily specify the feedback functionality, let alone design and implement it correctly.

To illustrate what information is missing, let us look at the complexity and diversity of the feedback feature. As we will see later, the HCI literature ([51], [54], [31], [53], [15], [6]) identifies four types of Feedback: *Interaction Feedback* to inform users that the system has heard their request, *Progress Feedback* for tasks that take some time to finish, *System Status Display* to inform users about any change in the system status, and *Warnings* to inform users about irreversible actions. Additionally, each feedback type has its own peculiarities. For example, many details have to be taken into account for a system to provide a satisfactory System Status Feedback: what states to report, what information to display for each state, how prominent the information should be in each case (e.g., should the application keep control of the system while reporting or should the system let the user work on other tasks while status reporting), etc.

Therefore, a lot more information than just a description of the usability feature must be specified to properly build the whole feedback feature into a software system. Developers need to discuss this information with and elicit it from the different stakeholders.

Note that the problem of increasing functional requirements completeness is generally solved by adding more information to the requirements [38], [6]. Even so, requirements completeness is never an easy problem to solve [12], [20], [39] and this is even harder in the case of functional usability requirements. In most cases, neither users nor developers are good sources of the information needed to completely specify a usability feature. Users know that they want feedback; what they do not know is what kind of feedback can be provided, which is best for each situation, and, still less, what issues need to be detailed to properly describe each feedback type. Neither do software engineers have the necessary HCI knowledge to completely specify such functional usability requirements since they are not usually trained in HCI skills [36].

The HCI literature suggests that HCI experts should join software development teams to deal with this missing expertise [28], [40]. However, this solution has several drawbacks. The first is that communication difficulties arise between the software developer team and HCI experts as HCI and SE are separate disciplines [47]. They use different vocabulary, notations, software development strategies, techniques, etc. Misunderstandings on these points can turn out to be a huge obstacle to software development. Another impediment is the cost. Large organizations can afford to pay for HCI experts, but many small-to-medium software companies cannot.

4 Generating Guidelines for Gathering Information about Functional Usability Features

Our approach consists of packaging guidelines that empower developers to capture functional usability requirements without depending on a usability expert. These guidelines help developers to understand the implications of and know how to elicit and specify usability features for a software system.

The information provided by the HCI literature is not directly applicable for this purpose. We have analyzed this information from a software development point of view and have elaborated on elicitation and specification guidelines. In the following, we describe this work in detail.

First, we extracted and categorized the information about functional usability features provided by the different HCI authors. We found the most detailed information on usability features in [53], [54], [51], [15], [31]. This information has served as a basis for identifying which issues should be discussed with stakeholders during the elicitation process. However, there is not enough HCI information to derive the essentials to be elicited and specified for all the functional usability features in Table 1. This is why features like Shortcuts and Reuse, for example, have been left out.

The usability features that we have worked on are listed below, along with their HCI sources of information. We intend to add the missing features to the list when more HCI information becomes available.

- Feedback (Tidwell [51], van Welie [54], Laasko [31], Brighton [53], Coram and Lee [15], Benson et al. [6]),
- Undo/Cancel (Brighton [53], Tidwell [51], [52], van Welie [54], Laasko [31]),
- User input errors prevention/correction (van Welie [54], Tidwell [52]),
- Wizard (van Welie [54], Tidwell [52]),
- User profile (van Welie [54], Tidwell [51]),
- Help (Tidwell [51], [52]), and
- Command aggregation (Tidwell [51], [52]).

Note that this is a preliminary working list of usability features. We plan to add other features, such as Default Settings [51] or History Logging [51], [52], for which the HCI literature provides information but whose relation to software design has not been yet researched.

Each HCI author identifies different varieties of these usability features. We have denoted these subtypes as usability mechanisms and have given them a name that is indicative of their functionality (see Table 2). Then, we defined the elicitation and specification guides for the usability mechanisms. We focused on the information provided by HCI authors. For example, Tidwell points out, for System Status [51], "Well-designed displays of information to be shown should be chosen. They need to be unobtrusive if the information is not critically important, but obtrusive if something critical happens. Choose well-designed displays of the information to be shown. Put them

TABLE 2
Usability Mechanisms for which Usability Elicitation and Specification Guides Have Been Developed

Usability Feature	**Usability Mechanism**	**HCI Authors' Label**	**Goal**
Feedback	System Status	Modeless Feedback Area [15] Status Display [51]	To inform users about the internal status of the system
	Interaction	Interaction Feedback [53] Modeless Feedback Area [15] Let Users Know What is Going On [6]	To inform users that the system has registered a user interaction, i.e. that the system has heard users
	Warning	Think Twice [53] Warning [54]	To inform users of any action with important consequences
	Long Action Feedback	Progress Indicator [51] [52] Show Computer is Thinking [53] Time to Do Something Else [53] Progress [54] Modeless Feedback Area [15] Let Users Know What is Going On [6]	To inform users that the system is processing an action that will take some time to complete
Undo Cancel	Global Undo	Multi-Level Undo [51] [52] Undo[54] Global Undo [31] Allow Undo [53] Go Back One Step [51]	To undo system actions at several levels
	Object-Specific Undo	Object-Specific Undo [31]	To undo several actions on an object
	Abort Operation	Go Back One Step [51] Emergency Exit [53] Cancellability [52]	To cancel the execution of an action or the whole application
	Go Back	Go Back to a Safe Place [51] Go Back One Step [51]	To go back to a particular state in a command execution sequence
User Input Error Prevention/ Correction	Structured Text Entry	Forms, Structured Text Entry [51] Structured Format [52] Structured Text Entry [53]	To help prevent the user from making data input errors
Wizard	Step-by-Step Execution	Step-by-Step [51] Wizard [54] [52]	To help users to do tasks that require different steps with user input and correct such input
User Profile	Preferences	User Preferences [51] Preferences [54]	To record each user's options for using system functions
	Personal Object Space	Personal Object Space [51]	To record each user's options for using the system interface.
	Favourites	Favourites [54] Bookmarks [51]	To record certain places of interest for the user
Help	Multilevel Help	Multilevel Help [52]	To provide different help levels for different users
Command Aggregation	Command Aggregation	Composed Command [51] Macros [52]	To express possible actions to be taken with the software through commands that can be built from smaller parts.

together in a way that emphasizes the important things, deemphasizes the trivial, doesn't hide or obscure anything, and prevents confusing one piece of information with another ... ." Similarly, Coram and Lee [15] focus on where to display this information.

We analyzed all of the recommendations on the same mechanism, combined them, and removed redundancies. However, as the HCI community is concerned not with software development but with what makes software usable, the resulting HCI-derived recommendations mainly focus on presentation issues and do not address the difficulties of building such features into a software system. The HCI recommendations cannot be used directly to capture software requirements, but they can be studied from a development point of view to generate issues to be discussed with the user to properly specify usability features. For System Status Feedback, for example, the criticality of the different tasks or situations to be reported needs to be analyzed since the appropriate information has to be displayed in different ways depending on its criticality. These details are important as they have an impact on this mechanism's design. Designing obtrusive information (for which purpose the program control has to be stopped) is quite different from designing unobtrusive information (where the runtime flow does not have to be interrupted).

Summarizing, we used HCI literature as a source of information. Then, we analyzed this information from a development perspective. Finally, we elaborated upon this information to get a set of issues to be discussed with stakeholders. The result of this work is shown in the next section.

5 A Pattern-Based Solution for Gathering Functional Usability Requirements

The outcome of the previous tasks is packaged in what we call a *usability elicitation pattern*. Patterns are already being used by other authors to reuse requirements knowledge. Patterns that capture general expertise to be reused during different requirements activities (elicitation, negotiation, documentation, etc.) are to be found in [22], [46], for example. In [55], Whitenak proposes 20 patterns to guide the analyst through the application of the best techniques and methods for the elicitation process. Patterns have already been used to represent specific functional requirements to be reused in different applications [37].

Our usability elicitation patterns capitalize upon elicitation know-how so that requirements engineers can reuse key usability issues intervening recurrently in different projects. Patterns help developers to extract the necessary information to specify a functional usability feature.

We have developed one usability elicitation pattern for each usability mechanism in Table 2. They are available at http://is.ls.fi.upm.es/research/usability/usability-elicitation-patterns. Tables 3a and 3b show an example of the System Status Feedback mechanism pattern. The next section discusses how to use this pattern. First, let us briefly describe the fields making up this pattern:

- *Identification* of the usability mechanism addressed by the pattern (that is, its *name*, the *family* of usability features to which it belongs, and possible aliases by which this usability mechanism may be known).
- The *problem* addressed by each pattern, that is, how to elicit and specify the information needed to incorporate in a software system the corresponding usability mechanism.
- The *usability context* in which this pattern will be useful.
- The *solution* to the problem addressed by the pattern. This is composed of two elements. The *usability mechanism elicitation guide* provides knowledge for eliciting information about the usability mechanism. It lists the issues to be discussed to properly define how the usability mechanism needs to be considered along with the corresponding HCI rationale. The *usability mechanism specification guide* provides an example of a specification skeleton.
- The *related patterns* refer to other usability elicitation patterns whose contexts are related to the one under study and which could also be considered in the same application. No related patterns have being identified in the example shown in Tables 3a and 3b. Readers are referred, however, to other patterns available at the Website, like Long Action Feedback or Abort Operation.

6 Using Patterns to Elicit and Specify Functional Usability Information

We use an example of a software system for theatre ticket sales for use by box office operators to illustrate pattern use. This is a highly interactive system with a specific user type. These two factors condition the usability requirements to be considered quite a lot.

The use of elicitation patterns involves instantiating them for each particular system. They should be applied after a preliminary version of the software requirements has been created. There needs to be an initial common vision of system functionality before developers and users can discuss whether and how specific usability mechanisms affect the software.

After this initial understanding of the software to be built, the developer can use the *identification* part of the pattern to appreciate the generics of the usability mechanisms to be addressed. The discussion with the stakeholders starts by examining the pattern *context* section that describes the situations for which this mechanism is useful. If the mechanism is not relevant for the application, its use will be rejected. Otherwise, the respective usability functionality will be elicited and specified using the *solution* part of the pattern.

In the case of our theatre system, our client considered that the Object Specific Undo, Command Aggregation, and User Profile-related mechanisms were of no interest. These application users do not get to be expert users because staff turnover is high. On this ground and because of the cost of incorporating the first two mechanisms, stakeholders decided that they were not to be built into the system. Again because of the high turnover, very few, if any, users are on the system long enough for the User Profile functionality to be warranted or even feasible.

The next step is to deal with the solution part of the pattern. Regarding *the usability mechanism elicitation guide*, it is important that developers read and understand the HCI rationales in the guide, that is, the HCI recommendations used to derive the respective issues to be discussed with stakeholders. This will help developers to understand why they need to deal with those issues. Not all questions in the patterns require the same level of involvement of all kinds of stakeholders. In fact, we can identify three groups of questions:

1. questions that the user/client can answer on his or her own (*which statuses are relevant in a particular application*?);

TABLE 3a
System Status Feedback Usability Elicitation Pattern

IDENTIFICATION	
Name: System Status Feedback	
Family: Feedback	
Alias: Status Display [51] Modelling Feedback Area [15]	
PROBLEM	
Which information needs to be elicited and specified for the application to provide users with status information.	
CONTEXT	
When changes that are important to the user occur or when failures that are important to the user occur, for example: during application execution; because there are not enough system resources; because external resources are not working properly. Examples of status feedback can be found on status bars in windows applications; train, bus or airline schedule systems; VCR displays; etc.	
SOLUTION	
Usability Mechanism Elicitation Guide:	
HCI Rationale	**Issue to discuss with stakeholders**
1. HCI experts argue that the user wants to be notified when a change of status occurs [51]	*Changes in the system status can be triggered by user-requested or other actions or when there is a problem with an external resource or another system resource.* *1.1 Does the user need the system to provide notification of* ***system statuses?*** *If so, which ones?* *1.2 Does the user need the system to provide notification of system* ***failures*** *(they represent any operation that the system is unable to complete, but they are not failures caused by incorrect entries by the user)? If so, which ones?* *1.3 Does the user want the system to provide notification if there are not* ***enough resources*** *to execute the ongoing commands? If so, which resources?* *1.4 Does the user want the system to provide notification if there is a problem with an* ***external resource*** *or device with which the system interacts? If so, which ones?*
2. Well-designed displays of information to be shown should be chosen. They need to be unobtrusive if the information is not critically important, but obtrusive if something critical happens. Displays should be arranged to emphasize the important things, de-emphasize the trivial, not hide or obscure anything, and prevent one piece of information from being confused with another. They should never be re-arranged, unless users do so themselves. Attention should be drawn to important information with bright colours, blinking or motion, sound or all three – but a technique appropriate to the actual importance of the situation to the user should be used [51].	*2.1. Which information will be shown to the user?* *2.2. Which of this information will have to be displayed obtrusively because it is related to a critical situation? Represented by an indicator in the main display area that prevents the user from continuing until the obtrusive information is closed.* *2.3. Which of this information will have to be highlighted because it is related to an important but non-critical situation? Using different colours and sound or motion, sizes, etc.* *2.4. Which of this information will be simply displayed in the status area? For example, providing some indicator.* Notice that for each piece of status information to be displayed according to its importance, the range will be from obtrusive indicators (e.g., a window in the main display area which prevents the user from continuing until it has been closed), through highlighting (with different colours, sounds, motion or sizes) to the least striking indicators (like a status-identifying icon placed in the system status area). Note that during the requirements elicitation process, the discussion of the exact response can be left until interface design time, but the importance of the different situations about which status information is to be provided and therefore which type of indicator (obtrusive, highlighted or standard) is to be provided does need to be discussed at this stage.

TABLE 3b
System Status Feedback Usability Elicitation Pattern (Continued)

SOLUTION (Cont.)	
Usability Mechanism Elicitation Guide (Cont.):	
HCI Rationale (Cont.)	**Issue to discuss with stakeholders (Cont.)**
3. As regards the location of the feedback indicator, HCI literature mentions that users want one place where they know they can easily find this status information [15]. On the other hand, aside from the spot on the screen where users work, users are most likely to see feedback in the centre or at the top of the screen, and are least likely to notice it at the bottom edge. The standard practice of putting information about changes in state on a status line at the bottom of a window is particularly unfortunate, especially if the style guide calls for lightweight type on a grey background [14]. The positioning of an item within the status display should be used to good effect. Remember that people born into a European or American culture tend to read left-to-right, top-to-bottom, and that something in the upper left corner will be looked at most often [51].	*3.1. Do people from different cultures use the system? If so, the system needs to present the system status information in the proper way (according to the user's culture). So, ask about the user's reading culture and customs.* *3.2. Which is the best place to locate the feedback information for each situation?*
Usability Mechanism Specification Guide:	
The following information will need to be instantiated in the requirements document. - The system statuses that shall be reported are X, XI, XII. The information to be shown in the status area is..... The highlighted information is … The obtrusive information is…. - The software system will need to provide feedback about failures I, II, III occurring in tasks A, B, C, respectively. The information related to failures I, II, etc…. must be shown in status area…. The information related to failures III, IV, etc., must be shown in highlighted format. The information related to failures V, VI, etc , must be shown in obtrusive format. - The software system provides feedback about resources D, E, F when failures IV, I and VI, respectively, occur. The information to be presented about those resources is O, P, Q. The information related to failures I, II, etc….must be shown in the status area..... The information related to failures III, IV, etc., must be shown in highlighted format. The information related to failures V, VI, etc., must be shown in obtrusive format. - The software system will need to provide feedback about the external resources G, J, K, when failures VII, VIII and IX, respectively, occur. The information to be presented about those resources is R, S, T. The information related to failures I, II, etc….must be shown in the status area..... The information related to failures III, IV, etc., must be shown in highlighted format. The information related to failures V, VI, etc., must be shown in obtrusive format.	
RELATED PATTERNS[3]:	

2. questions that the user/client can answer following the recommendations of a GUI expert (*which is the best place to locate the feedback information for each situation*?—the GUI expert is able to provide guidance for displaying this kind of information, whereas the user will choose one of the options); and
3. questions that the developer must answer but on which the user should give his opinion (*does the user*

TABLE 4
Fragment of the Issue/Functionality/Requirement Table for the System Status Feedback Pattern and the Tickets Sale Application

Source: Usability Elicitation Guide		*Source: Discussion with stakeholders*		*Source: Developer*	
ISSUE FOR DISCUSSION		**IS IT APPLICABLE TO THE SYSTEM?**	**AFFECTED FUNCTIONALITIES**	**EXISTING REQS. AFFECTED**	**NEW REQS.**
SYSTEM STATUS FEEDBACK					
ISSUES FOR HCI RATIONALE 1	*1.1 Does the user want the system to provide notification of **system statuses?** If so, which ones?*	Yes, the user wants confirmation when the system status changes after executing processes that have no output.	▪ Management of the data of all the theatres at the central office (management of days, sessions, discounts, seating, sales points, theatres, shows, performances and holidays)	3.1.1, 3.1.2, 3.1.3, 3.1.4, 3.1.5, 3.1.6 3.1.7, 3.1.8, 3.1.15	
			▪ Booking cancellation	3.1.9	
	*1.2 Does the user want the system to provide notification of system **failures** (system failures represent any operation that the system is unable to complete, but they are not failures caused by incorrect user entries)? If so, which ones?*	Yes, the user wants to be notified when an ongoing task has been blocked by something unrelated to external or internal resources. Otherwise the user would be wasting time waiting for an activity that is not in progress.	• Booking of tickets	3.1.9	
	*1.3 Does the user want the system to provide notification if there are not **enough resources** to execute the ongoing commands? If so, which resources?*	It is assumed that problems with internal resources (e.g., memory shortage or laptop low battery), if any, will be reported by the operating system and not the application			
	*1.4 Does the user want the system to provide notification if there is a problem with an **external resource** or device with which the system interacts? If so, which ones?*	The user wants to be notified if there is an error in any of the external resources with which it interacts.	▪ The user should be notified of any the CREDILINE card payment system error	3.1.10	
…	...	…	…	…	…

want the system to provide notification if there are not enough resources to execute the ongoing commands? If so, which resources?—the developer must provide information about the internal resources needed to perform the different tasks and the user will decide about which ones he or she wants to be informed.

We recommend documenting this discussion by means of an issue/functionality/requirement table such as Table 4. This table shows a fragment of this information for the ticket sales system. It shows, for each issue to be discussed with stakeholders:

1. Why it is applicable to this system and the specific functionalities that it affects.
2. The requirements related to the affected functionalities and/or possible new requirements that emerge from the usability issues discussed. In the case of the ticket system, no new requirements had to be added for the system status usability mechanism. Therefore, the last column of Table 4 is left blank.

This issue/functionality/requirement table helps developers to think systematically about the effect of usability mechanisms across the system, enabling them to identify which requirements will be affected. It is useful for discovering and documenting the usability mechanisms, even though it may take some extra effort.

The elicited usability information can be specified following the pattern specification guide. This guide is a prompt for the developer to modify each requirement affected by the incorporation of each mechanism. Fig. 1 shows a fragment of a requirement modified to include all of the usability mechanisms that affect it. The parts added as a result of using the respective usability elicitation patterns are highlighted in bold face and italics.

Modifying a requirement to include certain usability mechanisms involves adding the details describing how to apply these mechanisms to this functionality. As of this point, the remaining development phases are undertaken as always and the new usability functionality is integrated into the development process. This prevents the rework that the alternative of incorporating usability features at later development stages would entail.

Building usability mechanisms into a system results in an extra workload since new functionalities are incorporated. This process can be sped up by relaxing the documentation. The developers could skip building the issues/functionalities/requirements tables and modify the functional requirements directly or enter the results of the discussion about usability recommendations directly into some of the early software products (prototypes or mock-ups, analysis models, etc.), depending on the development process. To adopt this agile alternative, the developer will need to be quite experienced in the problem domain and pattern use so as not to overlook any details. Though the

Requirement 3.1.9. Ticket Booking

A booking can be created, but not modified. If you want to modify a booking, it will have to be cancelled (as specified in requirement 3.1.10) and another one created. Cancel implies deleting the booking.

The system will display a succession of windows for creating a booking. These windows contain the information described below and ***three options: back, next and cancel booking*** (Abort Operation, Go Back):

- First, a screen will be displayed listing ***what theatres there are for the user to choose from*** (Structured Text Entry).
- Then the system ***will display the show times at that theatre for the user to choose from*** (Structured Text Entry).
- Next the system displays a seating plan showing the seats booked and/or sold for the session in question. While this image is being loaded, ***a non-obtrusive window (which the user can minimize) and an indicator will be displayed to inform the user that an image is being loaded and how many seconds it will take to finish*** (Long Action Feedback)**.** ***Additionally, the user will be given the chance to cancel the operation, and the system will again display the selected theatre show times*** (Abort Operation)**.**
- ***The user will mark the seats he or she would like to book on this image*** (Structured Text Entry).
- ***Once the user has selected the seats he or she would like to book, another window will be displayed containing all the information about the selected seats and asking the user to confirm or cancel the operation. This window will warn the user that once he or she has confirmed the booking, it cannot be modified*** (Warning)**.**
- If the user confirms the operation the system will mark the seats as reserved and they will be allocated a unique booking code. ***If the user cancels, the system will go back to the first window listing what theatres there are*** (Abort Operation).
- ***At the end of the booking process, the system will display a window reporting whether or not the operation was a success or failure. This information will be displayed obtrusively in a dialogue box*** (System Status Feedback). ***The possible causes of the operation failing are: the seats had already been booked or there was a problem with the database connection.***

Fig. 1. Fragment of a requirement modified with usability functionality.

incorporation of the usability mechanisms can be made more agile, the elicitation effort is necessary for the results of the discussion to be able to be built into the software.

7 Evaluation of the Usability Elicitation Patterns

We have analyzed the potential benefits of the usability elicitation patterns at different levels. We have been working with SE Master students since 2003 on this project. These students have developed software systems to which they added the functional usability features listed in Table 2.

Initially, we focused on refining the content of the patterns. Ten students used the patterns on requirements they had specified. These students had all the knowledge about the problem domain to respond to the issues raised in the patterns. Our aim was to determine potential difficulties associated with the pattern structure, question statements, etc. We tuned the patterns and about 20 percent of their contents were reworked. The resulting operational version of the patterns is the one that we present here.

Then, we studied how useful the patterns were for incorporating the usability mechanisms into a software system. We expected pattern use to lead to an improvement on the original situation where developers did not have any compiled or systematic usability information. We worked with five groups of three students. Each group was randomly allocated a different software requirement specification (SRS) document in IEEE 830 format [25] corresponding to a real application:

- theatre tickets sale system,
- PC storage and assembly system,
- temping agency job offers management system,
- car dealer vehicle reservation and sale system, and
- travel agency bookings and sale system.

Each of the three students in the group was asked to add the functionality derived from the functional usability features listed in Section 4 to the original SRS independently and to build the respective software system. The procedure was as follows:

- We gave one of the students the usability elicitation patterns discussed in this paper. This student used the pattern content to elicit the usability functionality as explained in the last section.
- Another student was given reduced patterns. To give readers a feeling for the difference between the reduced and full patterns, the reduced pattern for System Status Feedback is shown in Appendix A. This short pattern is just a compilation of information from the HCI literature about the usability mechanisms. We have not elaborated upon this information from a development perspective, i.e., the reduced patterns do not include the "Issues to be discussed with stakeholders" column in Tables 3a

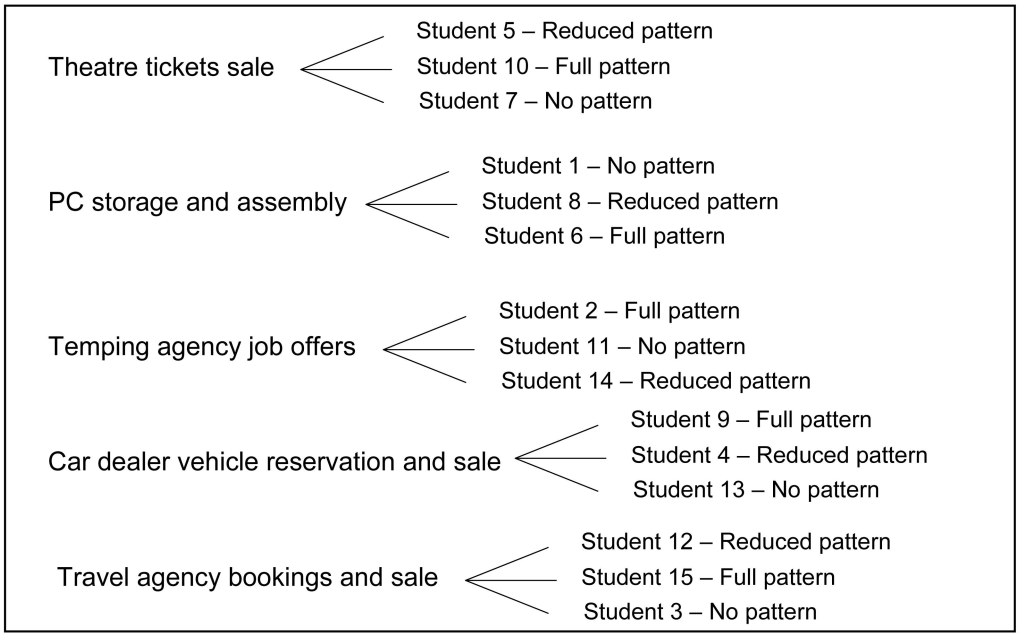
Theatre tickets sale
Student 5 – Reduced pattern
Student 10 – Full pattern
Student 7 – No pattern
PC storage and assembly
Student 1 – No pattern
Student 8 – Reduced pattern
Student 6 – Full pattern
Temping agency job offers
Student 2 – Full pattern
Student 11 – No pattern
Student 14 – Reduced pattern
Car dealer vehicle reservation and sale
Student 9 – Full pattern
Student 4 – Reduced pattern
Student 13 – No pattern
Travel agency bookings and sale
Student 12 – Reduced pattern
Student 15 – Full pattern
Student 3 – No pattern

Fig. 2. Results of the pattern allocation process.

and 3b. The idea behind using the reduced patterns was to confirm whether our processing of the HCI information resulting in the formulation of specific questions was useful for eliciting the functionality related to the mechanisms or whether developers are able to extract such details just from the HCI literature.

- Finally, the third student was given just the definitions of the usability features according to the usability heuristics found in the HCI literature and was encouraged to take information from other sources to expand this description.

Students of each group were randomly allocated the usability information they were to use (completed patterns, reduced patterns, and no patterns) to prevent student characteristics from possibly biasing the final result. Fig. 2 summarizes the allocation results.

Final system usability was analyzed differently to determine how useful the elicitation patterns were for building more usable software. We ran what the HCI literature defines as usability evaluations carried out by users and heuristic evaluations done by usability experts [14], [19], [48], [41], [42].

7.1 Users' Usability Evaluation

The usability evaluations conducted by users are based on usability tests in which the users state their opinion about the system. Before doing the tests, users need to carry out a number of standard tasks, called a scenario, to get acquainted with the software. We adapted the QUIS usability test [45] to the particularities of the users we worked with and the applications to be evaluated. Our users had no experience in usability testing and they did not have very long to perform the evaluation. Therefore, we focused on questions that would give a clearer picture of user satisfaction, rewrote the questions to make them more understandable for users in the context of the applications they were to evaluate, and used a scoring system based on a scale of 1 (lowest usability) to 5 (highest usability) rather than the standard QUIS test's 1-to-9 range (it is more straightforward and quicker to identify differences on a 5 than on a 9-point scale). Appendix B includes a usability scenario and the usability test we used. The final usability score is the mean of the responses to each question.

We worked with three representative users (typical users) for each system with whom our clients put us into touch. Each user evaluated the three versions of each application (one developed with full patterns, one with reduced patterns, and one with no patterns). This way the users could appreciate any differences between versions, although they did not know which usability information had been used to develop which version. The users evaluated these versions in a different order to prevent scenario learning possibly having a negative effect on the same version of the application. Table 5 shows this evaluation process (the order in which each user evaluated each version of the respective application appears between brackets).

All three representative users for each application were assembled in the same room. Each user executed the respective scenario for the first version of the application and completed the usability test. Users then enacted the same process for the second version of the application to be evaluated and, finally, did the same thing for the third version. As Table 5 shows, User 1 first evaluated the version of the theatre ticket system developed with the reduced patterns (i.e., executed the scenario and filled in the usability questionnaire), followed by the version developed with the full patterns, and, finally, the version developed with any pattern. The other users enacted the same process for this application and the same protocol was used for all five systems.

The mean usability values for the five applications are 4.4, 3.2, and 2.5, with standard deviations of 0.3, 0.2, and 0.4,

TABLE 5
Users Evaluation

Application	Version		
	With the Full Pattern	With the Reduced Pattern	With No Pattern
Theatre tickets sale	User 1 (2)	User 1 (1)	User 1 (3)
	User 2 (1)	User 2 (3)	User 2 (2)
	User 3 (3)	User 3 (2)	User 3 (1)
PC storage and assembly system	User 4 (1)	User 4 (2)	User 4 (3)
	User 5 (2)	User 5 (3)	User 5 (1)
	User 6 (3)	User 6 (1)	User 6 (2)
Temping agency job offers management	User 7 (3)	User 7 (2)	User 7 (1)
	User 8 (2)	User 8 (1)	User 8 (3)
	User 9 (1)	User 9 (3)	User 9 (2)
Car dealer vehicle reservation and sale	User 10 (3)	User 10 (1)	User 10 (2)
	User 11(2)	User 11 (3)	User 11 (1)
	User 12 (1)	User 12 (2)	User 12 (3)
Travel agency bookings and sale system	User 13 (3)	User 13 (2)	User 13 (1)
	User 14 (1)	User 14 (3)	User 14 (2)
	User 15 (2)	User 15 (1)	User 15 (3)

TABLE 6
Mean Percentage of Functionality Added for Each Usability Mechanism by Each Information Type

	Full usability elicitation patterns	**Reduced patterns**	**No pattern**	**Kruskal-Wallis (chi-square; df; p-value)**
Feedback	94%	47%	25%	12.658; 2; 0.002*
Undo/Cancel	90%	66%	43%	12.774; 2; 0.002*
User Profile	95%	80%	65%	12.597; 2; 0.002*
Users Input Errors Prevention/Correction	97%	85%	72%	12.727; 2; 0.002*
Wizard	100%	89%	71%	13.109; 2; 0.001*
Help	100%	81%	74%	13.109; 2; 0.001*

*** Statistically significant at 99% of confidence**

respectively. The Kruskal-Wallis test confirmed that there was a statistically significant difference among these usability means (p-value < 0.01; chi-square = 36.625). The Tamhane test (for unequal variances) showed that the usability value for the systems developed using the full patterns was statistically greater than the score achieved using the reduced patterns and both were greater than the usability value attained without any pattern (in all cases, p-value < 0.01). Therefore, we were able to confirm that the users perceived the usability of the systems developed with the full usability elicitation patterns to be higher.

With the aim of identifying the reasons that led users to assess the usability of the different types of applications differently, we had an expert in HCI run a heuristic evaluation.

7.2 Usability Expert Evaluation

A paid independent HCI expert ran the usability evaluation of the applications developed by our MSc students. The expert analyzed the applications focusing on how these systems provided the usability features listed in Table 2.

Table 6 shows the results of the heuristic evaluation. It indicates the extent to which the evaluated software incorporates the functionality related to each usability mechanism in all five applications. In the case of feedback, for example, the developers who used the respective elicitation patterns included, on average, 94 percent of the functionalities associated with this mechanism. Developers who used the reduced patterns incorporated 47 percent of the respective functionalities. Finally, developers who used no pattern included only 25 percent. The expert obtained these data working to a blind evaluation protocol, that is, the expert was not aware of the usability information used as input for each version of the applications.

Applying the Kruskal-Wallis test to the expert results for each usability feature, we found that there were statistically significant differences among the three groups of data (see the last column of Table 6 with p-value < 0.01 in all cases). Again, the Tamhane test showed that all of the usability features were built into the systems developed using the full patterns better than they were into systems developed using the reduced patterns and both provided more usability details than systems developed without patterns (with feature definitions only). This explains why users perceived differences in the usability of the systems.

Below we discuss some findings derived from this study:

- **The functionality added with the full elicitation patterns is less than 100 percent for the most complex patterns**, like Feedback and Undo. These differences are due to the fact that the complexity of these features calls for a very thorough analysis of the specifications to properly identify what parts of the system are affected. The final result then depends on how detailed and thorough the analyst is.

  Although bringing an HCI expert into systems development could possibly have led to 100 percent of all the usability details being identified, elicitation pattern use is an efficient alternative because of its cost. Also, developers should become more acquainted with the patterns as they apply them, and efficiency in use should gradually improve.
- Looking at Table 6 closely, we find that there is a **big difference in the functionality added using the usability information provided by the most and least complex patterns**.

  For example, developers using the reduced patterns are unable to identify all of the implications that the HCI recommendations have for the respective usability mechanisms. In the case of feedback, different students using the reduced patterns repeatedly considered the Status Feedback functionality related to user action confirmation (entries, value modifications, etc.) only, overlooking other types of status changes caused by system failures or failures of external resources interacting with the system. Developers who used the full patterns did explicitly consider the feedback related to these functionalities as the pattern includes precise questions concerning these possible changes. However, there are fewer differences between the two development types with regard to other usability features like Wizard or Help because these usability mechanisms are less complex and involve fewer details and cases. Even so, as mentioned before, these differences are also statistically significant.

  With regard to the usability features functionality built in without using patterns, we again found complexity-dependent differences between features. Although all such differences are statistically significant, the most relevant from a practical point of view are related to the Feedback and Undo features. Developers who did not use patterns failed to consider all the alternative usability mechanisms to be incorporated. For example, Interaction Feedback, System Status and Warning were hardly ever considered in the case of the Feedback feature. The same applies to Object Specific Undo or Go Back for the Undo feature, which were not accounted for either.
- **Some of the usability features that developers without patterns did identify were incorrectly built into the system**, sometimes causing a sizeable information overload. For example, it was quite common for developers to provide Long Feedback for all system actions, irrespective of how long they took. Also, on the few occasions where a Warning was provided, it was presented for all task types, irrespective of their irreversibility. This did not happen when patterns were used as patterns clearly specify when to provide these feedback types, depending on the task type, duration, irreversibility, etc.

  Another example of user information overload when patterns were not used is the indiscriminate use of blocking messages whatever type of information is being provided. This prevents the user from doing another task, for example, while he or she is waiting for a long task to finish. Again, pattern use rules this out as the patterns indicate under what circumstances blocking messages should or should not be used.
- The **weaknesses encountered in the systems produced by developers that did not use patterns are very much related to the problems they have in finding usability information**. The Web was the key source consulted. Developers stated that they found it difficult to locate consistent information about a specific usability feature because of the volume of highly dispersed data there is on the Web. This led developers to work with what they considered to be "sufficient information" about the usability features. From the expert evaluation, this information appears to have led to the systems being only partially usable.
- Also noteworthy is **the relationship between the usability problems that the expert pointed out about the systems and their modified SRS in which the developers specified the respective usability details**. For example, the underuse of some mechanisms or the information overloads of others are also stated in the SRS. In the first case, the modified SRS did not include the specification of the omitted mechanisms, whereas, in the second, the SRS stated that all the tasks carried out by the user should be confirmed. In other words, the usability information was incorrectly specified from the start of development and this had a negative effect on the final system's usability. This supports our belief in the importance of properly dealing with usability at the requirements stage.

Although these are interim data and further checks need to be run, the usability evaluations performed have revealed trends that need to be formally tested with a larger group of users and applications. The users' evaluation has shown that users perceive usability to be better in the versions of the application developed with the full usability elicitation patterns. On the other hand, the expert evaluation found no significant weaknesses in the usability functionality provided in the applications built using such patterns, whereas it detected sizeable gaps in applications built with reduced patterns or without any pattern at all.

These findings give us some confidence in the soundness of the usability elicitation patterns as a knowledge repository that is useful in the process of asking the right questions and capturing precise usability requirements for developing software without an HCI expert on the development team.

8 Conclusions

It is critically important to elicit requirements early enough in the development process, especially such requirements as have a big impact on software functionality [56], [17], [49]. Our work takes a step in this direction, suggesting that usability features with particular functional implications should be dealt with at the requirements stage. This is not a straightforward task as usability features are more difficult to specify than they may appear: A lot of details need to be explicitly discussed among stakeholders and, often, neither software practitioners nor users have the HCI expertise to do this.

We propose an alternative solution when HCI experts are not available or the likely communication problems between development and HCI teams are not cost justified. We have developed specific guidelines that lead software practitioners through the elicitation and specification process. This approach supports face-to-face communication among the different stakeholders during requirements elicitation to cut down ambiguous and implicit usability details as early as possible. As we have shown, these guidelines help developers to determine whether and how a usability feature applies to a particular system, leading to benefits for the usability of the final system. The use of these patterns leads to an extra workload during development because more time and effort is required to answer questions, modify requirements, etc. On the other hand, they save effort by directing developers to proven usability solutions, reducing thinking time and rework.

Evidently, the use of usability patterns and any other artifact for improving software system usability calls for a lot of user involvement throughout the development process. This is a premise in the usability literature that is also necessary in this case. If this condition cannot be satisfied, the final system is unlikely to be usable. In our opinion, therefore, a trade-off has to be made at the beginning of the development between user availability, time, and cost restrictions, on the one hand, and usability results, on the other.

Finally, note that the functional usability features addressed here are not sufficient to make software usable. As we point out in the paper, the usability literature contains a host of recommendations on how to do this and we have focused on the ones with the biggest impact on functionality.

Appendix A
Example of a Reduced Usability Pattern

An example of a reduced usability pattern is shown in Table 7.

Appendix B
Example of Usability Scenario and Usability Test

Use the Theatre Tickets System to do the following tasks:

1. List all the plays to be performed at the Lopez de Vega Theatre in the month of May 2006.
2. Enter a new theatre into the sales network. The particulars of the theatre are:
   Name: Amaya
   Address: c/Ramblas 19 Barcelona;
   Phone: 932256819
   Fax 932256811
3. Enter all of the performances at the Amaya theatre for January 2006. Performances will be at 6:00 pm from Tuesday to Thursday and at 6:00 pm and 10:00 pm from Friday to Sunday. The play will be El Quijote.
4. Book two tickets for the 10:00 pm performance of Othello at the Marquina theatre on 14/2/2006. After you have booked the tickets, you realize that you have made a mistake and you really wanted to book the seats for the 6:00 pm performance.
5. You are going to sell two booked tickets. You have the booking reference to do this. When you have located the tickets, suppose the customer asks you to check whether there are any tickets available for the next day. Check availability, cancel the booking, and book tickets for the performance at the same time on the next day.
6. Try to change the system menu formats and colors and save them for later use.

Usability Test for Theatre Ticket Sales System
User Name:
Organization:
Date:

After practicing with the respective usability scenario, answer the following questions by marking the respective number. Please feel free to make any further comments you would like to about each question. You are welcome to use the application under evaluation if you so wish to check your response to any of the questions.

1. **Is the system help useful for understanding what system options to select?**
   1 2 3 4 5
   Very useful Not at all useful
2. **Does the system provide an easy-to-use option for undoing the effect of any action once it has been taken? (For example, suppose you have booked tickets for one performance of a play and you then want to go back to the default performance.)**
   1 2 3 4 5
   Never Always

TABLE 7
Example of a Reduced Usability Pattern

IDENTIFICATION
Name: System Status Feedback
Family: Feedback
Alias: Status Display [51] Modelling Feedback Area [15]
PROBLEM
Which information needs to be elicited and specified in order to provide users with system status information.
CONTEXT
When changes that are important to the user occur or When failures that are important to the user occur, for example: - During task execution - Because there are not enough system resources - Because external resources are not working properly. Examples of status feedback can be found on status bars in windows applications; train, bus or airline schedule systems; VCR displays; etc.
SOLUTION
HCI experts argue that the user wants to be notified when a change of status occurs [51]. Therefore, choose well-designed displays of the information to be shown. They need to be unobtrusive if the information is not critically important, but obtrusive if something critical happens. Displays should be arranged to emphasize the important things, de-emphasize the trivial, not hide or obscure anything, and prevent one piece of information from being confused with another. They should never be re-arranged, unless users do so themselves. Attention should be drawn to important information with bright colours, blinking or motion, sound or all three – but a technique appropriate to the actual importance of the situation to the user should be used [51]. As regards the location of the feedback indicator, HCI literature mentions that users want one place where they know they can easily find this status information [15]]. On the other hand, aside from the spot on the screen where users work, users are most likely to see feedback in the centre or at the top of the screen, and are least likely to notice it at the bottom edge. The standard practice of putting information about changes in state on a status line at the bottom of a window is particularly unfortunate, especially if the style guide calls for lightweight type on a grey background [14]. The positioning of an item within the status display should be used to good effect. Remember that people born into a European or American culture tend to read left-to-right, top-to-bottom, and that something in the upper left corner will be looked at most often [51].

3. **Do you think the system should give you more information than it does while it is running an operation and you are waiting for the response?**

   1 2 3 4 5
   Not much more — Yes, a lot more

4. **Does the system allow you to quit any action easily and return to what you consider to be a logical state?**

   1 2 3 4 5
   Never — Always

5. **For actions that require more than one step (e.g., tickets booking), do you find the steps easy to follow instinctively?**

   1 2 3 4 5
   Never — Always

6. **Does the system allow you to save your preferences, e.g., by identifying the functions you use most?**

   1 2 3 4 5
   Never — Always

7. **Does the system allow you to save your preferences on interface details, colors, menu formats, etc.?**

   1 2 3 4 5
   Never — Always

8. **When you have to enter more than one data item into the system (e.g., to register clients), does the software help you by automatically validating data or by providing different options so that all you have to do is choose the right option?**

   1 2 3 4 5
   Never — Always

9. **Does the system notify you before taking an action with important consequences (e.g., selling the tickets or cancelling a play)?**

   1 2 3 4 5
   Never — Always

10. **Do you find it easy to move around the system instinctively?**

    1 2 3 4 5
    Not very intuitive — Very intuitive

11. **Are the system responses (e.g., to confirm actions or request information) easy to understand?**

1	2	3	4	5
Not understandable				Understandable

Acknowledgments

This research was funded by the Spanish Ministry of Education project TIN230005-00176.

References

[1] A. Andrés, J. Bosch, A. Charalampos, R. Chatley, X. Ferre, E. Folmer, N. Juristo, J. Magee, S. Menegos, and A. Moreno, *Usability Attributes Affected by Software Architecture,* Deliv. 2. STATUS project, June 2002, http://www.ls.fi.upm.es/status.

[2] L. Bass, B. John, and J. Kates, "Achieving Usability through Software Architecture," Technical Report CMU/SEI-2001-TR-005, Software Eng. Inst., Carnegie Mellon Univ., 2001.

[3] L. Bass and B. John, "Linking Usability to Software Architecture Patterns through General Scenarios," *The J. Systems and Software,* vol. 66, no. 3, pp. 187-197, 2003.

[4] L. Bass, B. John, N. Juristo, and M.I. Sanchez, "Usability Supporting Architectural Patterns," *Proc. Int'l Conf. Software Eng.,* tutorial, 2004.

[5] M. Barbacci, R. Ellison, A. Lattanze, J.A. Stafford, C.B. Weinstock, and W.G. Wood, *Quality Attribute Workshop,* third ed., CMU/SEI-2003-TR-016, Software Eng. Inst., Carnegie Mellon Univ., 2003.

[6] C. Benson, A. Elman, S. Nickell, and C. Robertson, GNOME Human, "Interface Guidelines," http://developer.gnome.org/projects/gup/hig/1.0/index.html, 2007.

[7] D. Berry, "The Importance of Ignorance in Requirements Engineering," *J. Systems and Software,* vol. 28, no. 2, pp. 179-184, 1995.

[8] B. Boehm, J.R. Brown, H. Kaspar, M. Lipow, G.J. Macleod, and M.J. Merritt, *Characteristics of Software Quality.* North Holland, 1978.

[9] R.G. Bias and D.J. Mayhew, *Cost-Justifying Usability. An Update for the Internet Age.* Elsevier, 2005.

[10] J. Bosch and N. Juristo, "Designing Software Architectures for Usability," *Proc. Int'l Conf. Software Eng.,* tutorial, 2003.

[11] F. Buschmann, R. Meuneir, H. Rohnert, P. Sommerland, and M. Stal, *Pattern-Oriented Software Architecture, A System of Patterns.* John Wiley and Sons, 1996.

[12] M.G. Chirstel and K.C. Kang, "Issues in Requirements Elicitation," Technical Report CMU/SEI-92-TR-012, Software Eng. Inst., Carnegie Mellon Univ., 1992.

[13] M. Chrusch, "Seven Great Myths of Usability," *Interactions,* pp. 13-16, Sept./Oct. 2000.

[14] L. Constantine and L. Lockwood, *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design.* Addison-Wesley, 1999

[15] T. Coram and L. Lee, "Experiences: A Pattern Language for User Interface Design," 1996, http://www.maplefish.com/todd/papers/experiences/Experiences.html.

[16] L.M. Cysneiros, V.M. Werneck, and A. Kushniruk, "Reusable Knowledge for Satisficing Usability Requirements." *Proc. 13th Int'l Conf. Requirements Eng.,* 2005.

[17] G.B. Davis, "Strategies for Information Requirements Determination," *IBM Systems J.,* vol. 21, no. 1, pp. 3-30, 1982.

[18] G.M. Donahue, "Usability and the Bottom Line," *IEEE Software,* vol. 18, no. 1, pp 22-30, Jan./Feb. 2001.

[19] J.S. Dumas and J.C. Redish, *A Practical Guide to Usability Testing.* Exert, 1999.

[20] C. Ebert and J.D. Man, "Requirements Uncertainty: Influencing Factors and Concrete Improvements," *Proc. Int'l Conf. Software Eng.,* pp. 553-560, 2005.

[21] E. Folmer, J. van Group, and J. Bosch, "Architecting for Usability: A Survey," *J. Systems and Software,* vol. 70, nos. 1-2, pp. 61-78, 2004.

[22] L. Hagge and K. Lappe, "Sharing Requirements Engineering Experience Using Patterns," *IEEE Software,* vol. 22, no. 1, pp. 24-31, Jan./Feb. 2005.

[23] D. Hix and H.R. Hartson, *Developing User Interfaces: Ensuring Usability through Product and Process.* John Wiley and Sons, 1993.

[24] N.L. Hsueh and J.Y. Kuo, "Distributed Requirements Elicitation Using Patterns Proceedings of the Modelling," *Proc. Identification and Control Conf.,* 2003.

[25] "IEEE Std 830: Recommended Practice for Software Requirements Specifications," IEEE, 1998.

[26] "IEEE Std 1061: Standard for a Software Quality Metrics Methodology," IEEE, 1998.

[27] "ISO 9126-1 Software Engineering—Product Quality—Part 1: Quality Model," ISO, 2000.

[28] "ISO Std 13407: Human-Centred Design Processes for Interactive Systems," ISO, 1999.

[29] "ISO Std. 18529: Human-Centered Lifecyle Process Descriptions," ISO, 2000.

[30] "ISO Std. 9241-11: Ergonomic Requirements for Office Work with Visual Display Terminals. Part 11: Guidance on Usability," ISO, 1998.

[31] S.A. Laasko, "User Interface Designing Patterns," 2003, http://www.cs.helsinki.fi/u/salaakso/patterns/index_tree.html.

[32] T. Jokela, "Guiding Designers to the World of Usability: Determining Usability Requirements through Teamwork," *Human-Centered Software Eng.,* A. Seffah, J. Gulliksen, and M. Desmarais, eds., Kluwer, 2005.

[33] N. Juristo, A. Moreno, and M. Sánchez, "Architectural Sensitive Usability Patterns," *Proc. Int'l Conf. Software Eng. Workshop Bridging the Gaps between Usability and Software Development,* 2003.

[34] N. Juristo, A. Moreno, and M. Sánchez, *Techniques and Patterns for Architecture-Level Usability Improvements,* Deliv. 3.4 STATUS project, May 2003, http://www.ls.fi.upm.es/status.

[35] N. Juristo, A.M. Moreno, and M.I. Sánchez-Segura, "Analysing the Impact on Usability on Software Design," *J. Systems and Software,* vol. 80, no. 9, pp. 1506-1516, Sept. 2007.

[36] R. Kazman, J. Gunaratne, and B. Jerome, "Why Can't Software Engineers and HCI Practitioners Work Together?" *Human-Computer Interaction Theory and Practice,* C. Stephanidis and L. Erlbaum, eds., Elsevier, 2003.

[37] S. Konrad and B. Cheng, "Requirements Patterns for Embedded Systems," *Proc. IEEE Int'l Conf. Requirements Eng.,* 2002.

[38] B. Kovitz, "Ambiguity and What to Do about It," *Proc. IEEE Joint Int'l Conf. Requirements Eng.,* key talk, 2002.

[39] S. Lauesen, "Communication Gaps in a Tender Process," *Requirements Eng.,* vol. 10, no. 4, pp. 247-261, Nov. 2005.

[40] D.J. Mayhew, *The Usability Engineering Lifecycle.* Morgan Kaufmann 1999.

[41] J. Nielsen, *Usability Engineering.* John Wiley & Sons, 1993.

[42] J. Nielsen, "Heuristic Evaluation," *Usability Inspection Methods,* J. Nielsen and R.L. Mack, eds., John Wiley & Sons, 1994.

[43] J. Nielsen, *Return on Investment for Usability.* Alertbox, Jan. 2003, http://www.useit.com.

[44] D. Perry and A. Wolf, "Foundations for the Study of Software Architecture," *ACM Software Eng. Notes,* vol. 17, no. 4, pp. 40-52, Oct. 1992.

[45] "QUISTM Questionnaire for User Interaction Satisfaction," http://lap.umd.edu/QUIS/, 2007.

[46] "REPARE," http://repare.desy.de/Repare/RepareController, 2006.

[47] A. Sheffah and E. Metzker, "The Obstacles and Myths of Usability and Software Engineering," *Comm. ACM,* vol. 47, no. 12, pp. 71-76, Dec. 2004.

[48] B. Shneiderman, *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Addison-Wesley, 1998.

[49] The Standish Group International Inc., "CHAOS Chronicles c.3.0," http://ww.standishgorup.com/chaos/toc.php, 2003.

[50] *Guide to the Software Engineering Body of Knowledge,* 2004, http://www.swebok.org,

[51] J. Tidwell, *The Case for HCI Design Patterns,* http://www.mit.edu/jdidwell/common_ground_onefile.htm, 1999.

[52] J. Tidwell, *Designing Interfaces. Patterns for Effective Interaction Design.* O'Reilly, 2005.

[53] *Usability Pattern Collection,* http://www.cmis.brighton.ac.uk/research/patterns/home.html, 2007.

[54] M. van Welie, *The Amsterdam Collection of Patterns in User Interface Design,* http://www.welie.com, 2007.

[55] B.G. Whitenak, "RAPPeL: A Requirements-Analysis Pattern Language for Object Oriented Development," *Pattern Languages of Program Design,* J.O. Coplien and D.C. Schmidt, eds., Addison-Wesley, 1995.

[56] K.E. Wiegers, *Software Requirements.* Microsoft Press, 1999.

Natalia Juristo received the BS and PhD degrees in computing from the Technical University of Madrid (UPM). She is a full professor of software engineering with the Computing School at UPM, Spain. She has been the director of the UPM MSc in Software Engineering program for 10 years. Her research areas include software usability, empirical software engineering, requirements engineering, and software process.

Ana Maria Moreno received the BS and PhD degrees in computing. She is an associate professor with the Computer Science School at the Universidad Politecnica de Madrid. Since 2001, she has been the director of the MSc in Software Engineering program. Her research interests are software usability, requirements engineering, and empirical software engineering.

Maria-Isabel Sanchez-Segura received the PhD degree in computer science from the Universidad Politécnica of Madrid. She has been an associate professor in the Computer Science Department at Carlos III University of Madrid since 1998. Her research interests include project management, software reuse, process improvement, and process improvement usability using collaborative environments.