A Glass Box Design: Making the Impact of Usability on Software Development Visible

Natalia Juristo[1], Ana Moreno[1], Maria-Isabel Sanchez-Segura[2], M. Cecília C. Baranauskas[3]

[1] School of Computing - Universidad Politécnica de Madrid, Spain; [2]Department of Computing - Universidad Carlos III de Madrid, Spain; [3]Institute of Computing – UNICAMP State University of Campinas, Brazil
{natalia, ammoreno}@ fi.upm.es, misanche@inf.uc3m.es, cecilia@ic.unicamp.br

Abstract. User-centered design is not just about building nice-looking and usable interfaces, and software development is not just about implementing functionality that supports user tasks. This paper aims to build a tighter fit between human-computer interaction and software engineering practices and research by addressing what software and usability engineering practitioners can learn from each other regarding the impact of usability on software development. More specifically we aim to support usability people in helping developers to elicit requirements that can incorporate usability functions into software development. The paper shows what type of impact usability has on software models and suggests how this impact can be dealt with at the requirements elicitation and specification stages of the development cycle.

Keywords: usability features, software development,

1 Introduction and Motivation

User-centered design is not just about building nice-looking and usable user interfaces, and software development is not just about implementing functionality that supports user tasks. Belonging to different traditions and times, the disciplines of software engineering (SE) and human computer interaction (HCI) have evolved separately and developed their own methods to meet the needs of software application customers and users. Over the last twenty years, especially, several practices have been developed in both communities. This illustrates the human-centered perspective's influence on the software development lifecycle. Stakeholder involvement in the requirements elicitation process and evolutionary process models [39], which enable iterative construction of the application software to best meet the user requirements, and the widespread use of prototyping by the HCI community are examples of the approximation of both disciplines [31].

Even so, the closeness of the two disciplines has not facilitated their practitioners' work. One of the most significant problems is that there is little knowledge and communication about each others' jobs, even though they are supposed to work together and collaborate on developing software applications. Both communities use different vocabulary and sometimes the same word to represent different artifacts. The word “design”, for example, can be understood and used differently in both communities: to express the software modeling phase (in SE), and the final look and feel of the product (in HCI).

The boundaries between SE and HCI and attempts at bridging gaps between the two disciplines by fusing the efforts of the two communities come up against plenty of challenges. Seffah [37] discusses the major obstacles and related myths as viewed by the two communities. He argues that some widespread expressions, such as “friendliness of the user interface”, and “user interface” *per se*, are some of the underlying obstacles to more usable interactive systems. In his view, such terms give the impression that the user-centered approach and usability methods are only for “decorating a thin component sitting on top of the software or the “real” system”. The behavior of some practitioners in both communities illustrates this dichotomy that isolates the user interface from the remaining system. Some software engineers consider themselves the “real” designers of the software system, leaving the task of making the UI more “user friendly” to the usability people. On the other hand, some usability professionals and interaction designers believe that it is their job just to design and test the interface with users, leaving the task of implementing the system functionality that supports the user tasks to the software engineers.

Two separate processes – one for the overall system development process and another for the user-centered process – for building an interactive system are certainly not manageable. It would not appear to be possible to control and synchronize software development and user-centered design separately. Also, efficiency would fall and costs would increase because of the overlapping functions between the two processes. Milewski [31] argues that, despite improvements in SE-HCI interactions, problems of communication and efficiency still remain and deserve further investigation. Seffah [36] proposes that software and usability engineers should learn from each other to facilitate and encourage the convergence of practices in both communities. Therefore we need to investigate and to promote the mutual understanding of the activities and responsibilities of the two communities in the construction of usable systems.

We argue that by making the impact usability has on software development visible to usability people, they will be better prepared to help developers and analysts to elicit the requirements necessary to develop the software. To do this, the HCI people must have an “insider” view of the software to be able to see how to best mediate between users and developers and ensure that usability issues will be properly dealt with in the software design during the whole development cycle. This paper aims to shed light on how usability recommendations relate to software models and suggests an approach to deal with this as of the SE requirements stage.

The paper is organized as follows. The next section discusses the concept of usability as a quality requirement for software and, as such, the suggested interdependencies with other software-related concepts. It also discusses the roles the two communities have to play to integrate their practices and build usable interactive systems. Section 3 gives an example to illustrate the implications of considering a usability feature in the software model of a real application. Section 4 proposes a way to guide developers and usability engineers through the process of eliciting and specifying requirements to deal with usability features at design time. Section 5 sets out our conclusions.

2 Usability as a Quality Attribute for Software

Software quality has been traditionally associated with the satisfaction of specified requirements that are primarily functional. This perception of quality was broadened as of ISO/IEC 9126 [20], which defines quality from a user perspective as functionality, reliability, usability, efficiency, maintainability and portability. Thus, the concept of usability that in the HCI field tradition has been associated with the "usefulness" of software and represented by its learnability, efficiency of use, memorability, few and noncatastrophic errors and subjective satisfaction [32], [38], gains a formal status as a quality attribute for software.

Quality in use has a broader sense, although some ergonomic standards (e.g. [21] and [22]) take the expression to be a synonym of usability. Bevan [6] explains the ISO/IEC 14598-1[23] concept of quality in use as the combined effect of the system's quality characteristics, and the way they are experienced by the end user. In particular, it depends on the circumstances in which a system is used, including factors such as the tasks, equipment (hardware, software and materials), and the physical and social environment. As discussed in Bevan [5], quality in use depends not only on software usability, but also on other software quality attributes: the appropriateness of the provided functionality for a given task, the performance that users can achieve using the system, and system reliability.

For the past two decades software usability has been perceived from the SE perspective as related to the presentation of information to the user [37],[16]. This underlying belief has led software engineers to treat usability primarily by separating the presentation portion from the system functionality, as recommended by generally accepted design strategies (for example, MVC or PAC [10]). This separation makes it easier to modify the user interface in the hope of improving its usability, without affecting the rest of the application. Consequently, usability, unlike other software quality attributes, has been considered late in the development process.

Nevertheless, some authors have recently shown that usability has implications beyond the user interface *per se*. Perry and Wolf have claimed that usability issues place static and dynamic constraints on the software components [33]. Bass et al. [2] and Bass and John [3] used a bottom-up approach based on fieldwork observation to describe a set of scenarios representing software usability issues that have an effect on the software architecture. The STATUS project [25],[26] decomposed usability into features and studied which features have an impact on software architecture. The implications of usability features for software design materialize as the creation of special items (components, responsibilities, interactions, classes, methods, etc.) that affect both the presentation and application layers of the software system architecture. Therefore, addressing usability features at the end of the construction process leads to major rework. To avoid this, it has been suggested that software usability should be addressed proactively at architectural time instead of retroactively after testing [4],[8],[16],[27].

While the SE community developed its own set of methods and tools to manage the software development lifecycle and covering usability concerns, the HCI community has developed a variety of user-centered approaches to software design over the last decades. Although many developers recognize the importance of user-centered design (UCD) for achieving usability, the methods and techniques are not the everyday practice for software developers for many reasons, some of which have already been mentioned in this paper. Seffah and Metzker [37] also point out that UCD techniques are usually regarded as being uncoupled from the mainstream software development lifecycles, and propose educating software and usability professionals to work together as a way of filling the gap between software and usability engineering. The authors argue that a sound understanding of HCI and SE culture and practices will help practitioners to communicate and upgrade methodologies. They also suggest the need to consider a possible cause-and-effect relationship between the internal software quality attributes and the UI usability factors. For example, for a system that must provide continuous feedback, developers should consider this point when designing time-consuming system functionalities [36].

Faulkner [14] also analyzes the need for integration between HCI and SE practices. Faulkner explains this integration in terms of their role in the context of the computer science curriculum. She claims that usability practitioners lack essential SE skills. This is why they find it so difficult to influence developers' activities and attitudes. At the same time, software developers receive only rudimentary training in usability and user-centered approaches to design, despite the fact that between 50% and 80% of all source code is concerned with the user interface [7],[11]. Along similar lines, other authors [18], [15] acknowledge the importance, for all software practitioners, not only of being acquainted with a variety of usability methods, but also of being able to determine which method is best suited for a particular situation in a software project. We argue that this is important the other way round too: usability practitioners should also be able to visualize and understand what impact usability features are likely to have on software models. This way they will be able to understand the implications they have and be able to communicate better with developers and mediate in the requirements elicitation process.

In this paper we propose a "glass box" approach to software design showing that usability recommendations have an impact and therefore should have features represented in software models. These are recommendations involving building certain functionalities into the software to improve user-system interaction. For example, features like cancel an ongoing task [9], undo a task [38],[42],[30] receive feedback on what is going on in the system [19],[38],[41] or adapt the software functionalities to the user profile [35], [42]. In general terms we advocate bringing usability into the software development process earlier, at requirements time, as is being done with other quality attributes [1]: "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 tradeoffs early in the lifecycle than later in the lifecycle when modifications are often difficult, impractical, or even impossible".

3 The Impact of Usability Recommendations on Software Design

Both HCI [24] and SE [17] deal with usability as a non-functional requirement. Usability requirements specify user effectiveness, efficiency or satisfaction levels that the system should achieve. These specifications are then used by the usability people at the evaluation stage: “Task A should be performed by a novice user in less than X minutes”, or “End user satisfaction with the application should be higher than Z on a 1-to-5 scale”. Dealing with usability as a non-functional requirement is useful for evaluation purposes, but it is not appropriate for developing usable software.

Usability recommendations have an impact on software design [28]. Suppose that we want to build the cancel functionality for specific commands into an application. The recommendations mention “provide a way to instantly cancel a time-consuming operation, with no side-effects” [41] or “if the time to finish the processing of the command will be more than 10 seconds, along to the feedback information, provide a cancel option to interrupt the processing and go back to the previous state”[32]. To satisfy such requirements the software system must at least: gather information (data modifications, resource usage, etc.) that allow the system to recover the status prior to a command execution; stop command execution; estimate the time to cancel and inform the user of progress in cancellation; restore the system to the status before the cancelled command, etc. This means that, apart from the changes that have to be made to the UI presentation to add the cancel button, specific software components must be built into the software design to deal with these responsibilities. Building the cancel usability feature into a finished software system is not a minor change; it takes a substantial amount of software design rework. Therefore, it should be dealt with well in advance like any other software system functionality. The same applies to other usability recommendations that represent specific functionalities to be incorporated into a software system. We have termed these recommendations functional usability features (FUF), as most of these heuristics/rules/principles make recommendations on functionalities that the software should incorporate.

3.1 Functional Usability Features (FUFs)

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 [32], principles of usability [4],[38], usability guidelines [19], etc. Although all these recommendations share the same goal of improving software system usability, they are stated at different levels. For example, there are very abstract guidelines, like “prevent errors” [32] or “support internal locus of control” [38],[19], and others that provide more definite usability solutions, such as “make the user actions easily reversible” [19] or “provide clearly marked exits” [32].

To identify a preliminary list of functional usability features we took the usability features with strong design implications (according to the STATUS project [25],[26] and Bass et al. [2],[3]). Table 1 shows the list of FUFs. The usability features in Table 1 are not intended to be an exhaustive list; they are a starting point for identifying usability features with an impact on software system functionality. They are a good example of usability features that have to be considered at the software requirements stage.

If the usability-related functionalities described in Table 1 are properly described in the software requirements specification, developers are more likely to build them into the system.

Table 1. A Preliminary List of Functional Usability Features

Usability Features	Usability benefits
Feedback	Nielsen [32], Constantine & Lockwood [12], Shneiderman [38], Hix & Hartson [19]
Undo / Cancel	Nielsen [32], Hix & Hartson [19], Shneiderman [38], Constantine & Lockwood [12]
User input errors prevention/correction	Shneiderman [38], Hix & Hartson [19], Constantine& Lockwood [12]
Wizard	Constantine& Lockwood [12]
User profile	Hix & Hartson [19]
Help	Nielsen [32], Constantine& Lockwood [12]
Commands aggregation	Nielsen [32], Constantine& Lockwood [12], Hix [19]
Shortcuts (key and tasks)	Nielsen [32], Constantine& Lockwood [12], Shneiderman [38], Hix & Hartson [19]
Reuse information	Constantine & Lockwood [12]

Properly specifying functional usability features is not that simple. For example, Nielsen's feedback heuristic "The system should always keep users informed about what is going on through appropriate feedback within reasonable time" [32] does not provide the amount of information that a software developer needs to satisfactorily specify the feedback functionality, build the software design model and implement it adequately.

To illustrate what information is missing let us look at the complexity and diversity of the feedback functionality. There are four possible 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, a developer needs many details to build a system that provides 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 (for example, 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 incorporated in the software requirements specification document for developers to properly build the feedback feature into a software system. This information needs to be discussed with and elicited from the different stakeholders, and requirements elicitation, analysis and specification is a process where both developers and usability engineers have a role to play.

3.2 Making the Relationship between FUFs and Software Design Visible

This section discusses an example illustrating the incorporation of one FUF for a system developed to manage on-line table bookings for a restaurant chain. The FUF we will look at deals with one particular kind of feedback, System Status Feedback (to report the system status to users) [41],[13].

Figure 1 illustrates part of the UML [29] class model for this system, including the classes involved in the restaurant table booking process only. The original classes designed without considering system status feedback are shown on a white background in the diagram and the classes derived from the later inclusion of this feature on a grey background. Let us first analyze this diagram without taking into account the system status feedback. The Interface class represents the part of the bookings terminal with which the user interacts to book a table at the restaurant over the Internet. The Restaurants-Manager and Reservations-Manager are two classes specialized in dealing with restaurants and bookings, respectively. They have been added to make the system more maintainable by separating the management processes so that they can be modified more efficiently in the future. The Reservations class accommodates the information related to each booking that is made at the restaurant, whereas the restaurant is represented by the Restaurant class and each of the tables at this restaurant that are not taken are identified by the Table class.

The System Status Feedback Usability Feature involves the system notifying the user of any change of state that is likely to be of interest. HCI experts recommend that the notification should be more or less striking for the user depending on how important the change is. The status changes that the client considered important referred to failures during the performance of certain system operations owing either to runtime faults, insufficient resources or problems with devices or external resources (specifically Internet). This means that the system should have the following specific responsibilities to provide this usability feature:

R1: The software should be able to listen to active commands, because they can provide information about the system status. If this information is useful to the user, the system should be able to pass this information on as, when and where needed.

R2: As the system may fail, the software should be able to ascertain the state of active commands, because users should be informed if the command is not working due to outside circumstances. The system should be equipped with a mechanism by means of which it can check whether a given command is being executed and, if the command fails, inform users that the command is not operational.

R3: The software should be able to listen to or query external resources, like networks or databases, about their status and inform the user if a resource is not working properly.

R4: The software should be able to check the status of the internal system resources and alert users in advance to save their work, i.e. if the system is running short of a given resource and is likely to have to shut down.

The grey boxes in Figure 1 show the changes that had to be made to the design model for the software to be able to deal with these responsibilities. They include three new classes, five new methods, and four new associations. Their description follows:

Three new classes:

Internal-Resource-Checker, responsible for changes and for determining failures due to internal system resources like memory, etc. It deals with responsibilities *R1*, *R2* and *R4*.

External-Resource-Checker, responsible for changes and determining failures due to external system resources like networks, databases, etc. It deals with responsibility *R3*.

Status-Feedbacker, responsible for providing the client with information stating the grounds for both changes and system failures in the best possible format and using the most appropriate heuristic in each case. It deals with all four responsibilities.
Five new methods:

ExternalResourcesAvailable determines whether the external resources are available.

CheckSystemResources determines whether the system has the resources it needs to execute the booking operation for the restaurant.

AreYouAlive finds out whether the active command is still running correctly.

IAmAlive, enables the Reservations-Manager to tell the system that it is still running, i.e. that it is alive.

Feedback tells the user of any changes or failures identified in the system while the table is being booked at the restaurant.
Four new associations between:

Reservations-Manager&StatusFeedbacker enables the ReservationsManager class to tell the StatusFeedbacker about changes or system failures that are relevant for the user and about which the user is to be informed.

Internal-Resource-Checker&Status-Feedbacker and *External-Resource-Checker&Status-Feedbacker* can tell the Status-Feedbacker class that a change or failure has occurred.

Reservations-Manager&Internal-Resources-Checker enables the Internal-Resources-Checker to check that the reservation process is still active and working properly.

If an external resource, in this case the network, failed, the *External-Resources-Checker* would take charge of detecting this state of affairs (as no *ExternalResourceAvailable* message would be received). It would then alert the user through the *Status-Feedbacker*. If the shaded classes were omitted from this diagram, the system would not be able to detect the network failure. Note that a skilful software developer could have implemented this operation with a time-out in anticipation of a possible network failure. In this case, however, the potential usability of the application would depend on the developer's skill, with no guarantee of it working properly. Explicitly incorporating the components for this usability feature in the software design models provides assurance that these components will be codified by the programmer and, therefore, the usability feature will be included in the final system.

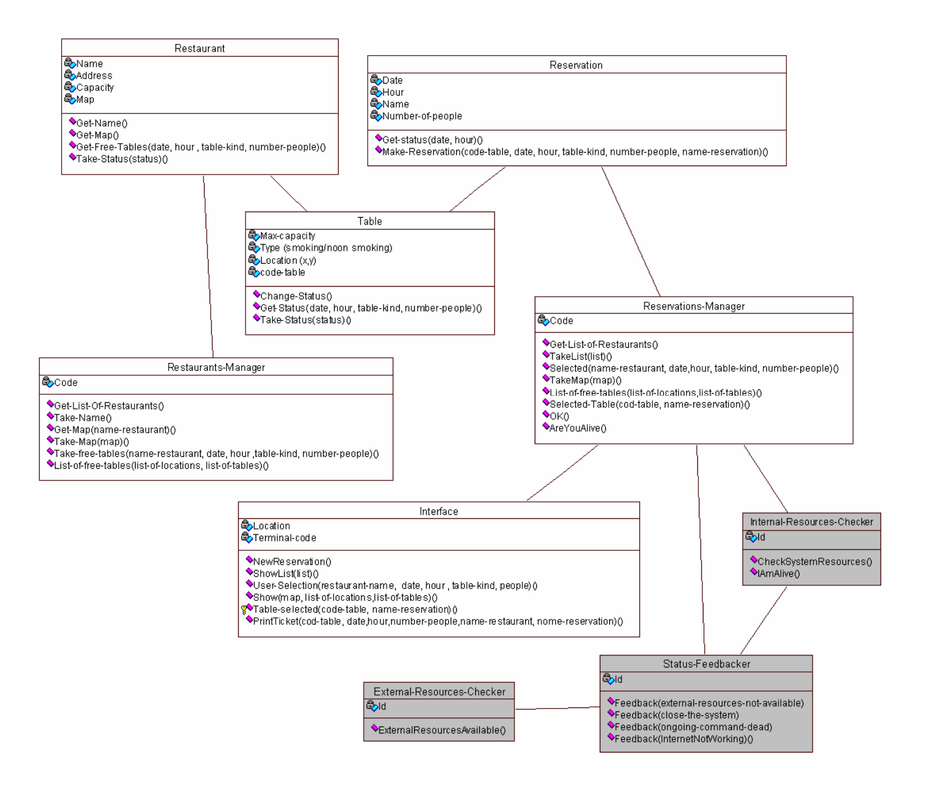
Restaurant
Name
Address
Capacity
Map
Get-Name()
Get-Map()
Get-Free-Tables(date, hour , table-kind, number-people)()
Take-Status(status)()
Reservation
Date
Hour
Name
Number-of-people
Get-status(date, hour)()
Make-Reservation(code-table, date, hour, table-kind, number-people, name-reservation)()
Table
Max-capacity
Type (smoking/noon smoking)
Location (x,y)
code-table
Change-Status()
Get-Status(date, hour, table-kind, number-people)()
Take-Status(status)()
Reservations-Manager
Code
Get-List-of-Restaurants()
TakeList(list)()
Selected(name-restaurant, date,hour, table-kind, number-people)()
TakeMap(map)()
List-of-free-tables(list-of-locations,list-of-tables)()
Selected-Table(cod-table, name-reservation)()
OK()
AreYouAlive()
Restaurants-Manager
Code
Get-List-Of-Restaurants()
Take-Name()
Get-Map(name-restaurant)()
Take-Map(map)()
Take-free-tables(name-restaurant, date, hour ,table-kind, number-people)()
List-of-free-tables(list-of-locations, list-of-tables)()
Interface
Location
Terminal-code
NewReservation()
ShowList(list)()
User-Selection(restaurant-name, date, hour , table-kind, people)()
Show(map, list-of-locations,list-of-tables)()
Table-selected(code-table, name-reservation)()
PrintTicket(cod-table, date,hour,number-people,name-restaurant, nome-reservation)()
Internal-Resources-Checker
Id
CheckSystemResources()
IAmAlive()
Status-Feedbacker
Id
Feedback(external-resources-not-available)
Feedback(close-the-system)
Feedback(ongoing-command-dead)
Feedback(InternetNotWorking)()
External-Resources-Checker
Id
ExternalResourcesAvailable()

Figure 1. Class Diagram for restaurant management with System Status Feedback feature

The above example illustrates what impact the inclusion of the System Status Feedback feature has on software design (and therefore on later software products and code). Other designers could have come up with different, albeit equally valid, designs to satisfy the above-mentioned responsibilities. For example, a less reusable class model might not have included new classes for this type of feedback but have added any new methods needed to existing classes. In any case, the incorporation of this usability feature to the system can be said to have a significant impact on software design.

Note that this example only shows the effect of this usability feature for a specific functionality (table booking). This effect is multiplied if this feature is added to the other system functionalities and even more so if we consider all the other FUFs.

4 Guiding the Elicitation and Specification of Functional Usability Features

As we have shown in the last section, usability features, like any other quality attribute, have an impact on software design. Consequently, they should be considered as of the early stages of software development.

Usability elicitation patterns (USEP) were developed for each usability mechanism identified and are available[1] as a result of the STATUS project and later research. The USEP can be viewed not only as a support for software developers to extract all the necessary information to completely specify a functional usability feature, but also as an instrument for communication between and a common ground for both software developers and usability people. This can help developers to think systematically about the effect of usability mechanisms across the system, assisting them to identify what requirements will be affected. At the same time, it may help usability people to understand its impact on the "insides" of the software application and trade off users' needs against developers concerns. Table 2 illustrates a fragment of an elicitation pattern for the System Status Feedback.

Table 2. A Fragment of the System Status Feedback Elicitation Pattern

IDENTIFICATION	
Name: Name: System Status Feedback	
Family: Feedback	
Alias: Status Display [7], Modelling Feedback Area [2]	
PROBLEM	
Which information needs to be elicited and specified so that the application provides 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: - During task execution - Because there are not enough system resources - Because external resources are not working properly. Examples of status feedback can be found in status bars on windows applications; train, bus or airline schedule systems; VCR displays; etc	
SOLUTION	
Usability Mechanism Elicitation Guide	
HCI Rationale	Issues to be discussed with stakeholders
1. HCI experts argue that the user wants to be notified when a change of status occurs [7] 2.… 3…	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 want the system to provide notification of **system statuses**? If so, which ones? 1.2 Does the user want 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?

[1] at http://is.ls.fi.upm.es/research/usability/usability-elicitation -patterns

	2.1 … 2.2 … 3.1 …
Usability Mechanism Specification Guide	
The following information will need to be instantiated in the requirements document. - The system statuses that should 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.	

The *identification* part of the USEP gives an understanding of the generics of the usability mechanism to be addressed, and the *problem* and *context* to which it applies. The *solution* part contains two elements: the *usability mechanism elicitation guide* and the *usability mechanism specification guide*. The usability mechanism elicitation guide provides knowledge for eliciting information about the usability mechanism. It lists the issues that stakeholders should discuss to properly define how the usability mechanism needs to be considered along with the corresponding HCI rationale (i.e. the HCI recommendation used to derive the respective issues). Usability engineers could help developers to understand the HCI rationales in the guide and to mediate the discussion of the related issues with stakeholders. They may also extend and adapt the information when necessary for specific situations.

The elicited usability information can be specified following the usability mechanism specification guide. This guide is a prompt for the developer to modify each requirement affected by the incorporation of each mechanism. Figure 3 shows a fragment of a requirement for the restaurant booking application, modified to include all the usability mechanisms that affect it. The parts added as a result of using the respective USEPs are highlighted using bold face and italics.

This system is an attempt at an innovative solution for reserving restaurant tables using a reservation support terminal located at street level and connected to Internet.

When a customer accesses a reservation support terminal, the terminal asks the user the name of the restaurant, the date and the time at which he or she would like to book a table. Through a central reservations office, which has information on the status of all the tables in the restaurant chain, the terminal checks whether there is a vacant table at the chosen restaurant at the elected time. If there is, the central reservations office, with the aid of the chosen restaurant, first sends a plan of the restaurant and then the vacant tables with their respective positions on the plan. This way the reservation support terminal can reconstruct the restaurant plan indicating which tables are vacant.

The user selects a table and enters the number of people for which the table is to be booked. The reservation support terminal communicates this information to the central reservations office, which then checks with the restaurant that everything is still in order. If everything is OK, the terminal asks the user to enter the name under which the table is to be booked. The user enters the name, and the terminal communicates this information to the central reservations office. The central reservations office books the table and issues a ticket specifying the date, time, table number and the name under which the table has been booked. ***This confirms the reservation (status).***

If there are no vacant tables at the time specified by the user, the reservation support terminal informs the customer, giving the user the option of asking the system to list the restaurants that have vacancies on the day and at the time of his/her choice.

System operationality should be such as to take into account possible system and network (Internet) crashes. The user should be given a warning if the system is not working because of an item that is vital for function execution being blocked (internal resources). Additionally, the user will be warned obtrusively if the system is unable to access the network in the course of table booking due to a network crash, as the user should not be allowed to continue with the reservation if a connectivity failure (external resources) prevents the customer's request from being satisfied.

Figure 2. Fragment of a requirement modified with FUF

Modifying a requirement to include certain usability mechanisms involves adding the details describing how to apply these mechanisms to this functionality. The remaining development phases are undertaken as always, and the new usability functionality is integrated into the development process.

5 Conclusions

Although belonging to different traditions and times, the disciplines of software engineering (SE) and human computer interaction (HCI) share the same goal of meeting the needs of application software customers and users. Even so, the boundaries between SE and HCI and attempts at bridging gaps between the two disciplines by fusing the efforts of both communities come up against plenty of challenges. The literature suggests that the main obstacles are problems of communication and efficiency, and that software and usability engineers have little knowledge of each others' jobs.

This paper aimed to facilitate and encourage the convergence of practices in both communities by first showing that usability does have an impact on software models and second proposing a pattern-based way of dealing with usability features with particular functional implications at requirements time. Usability features are more difficult to specify than it would seem; a lot of details need to be explicitly discussed with stakeholders and need the expertise of both usability engineers and software developers. Although the functional usability features addressed by USEPs are not sufficient to make software usable, they do represent the features with the biggest impact on functionality. The potential benefits of USEPs have been investigated in real projects developed as part of an SE Master program since 2003. Different analyses of final system usability performed have shown how useful the elicitation patterns are for building more usable software.

In summary, the proposed approach encourages a mutual understanding of the activities and responsibilities between the two communities with a view to the construction of usable systems. By making the implications of functional usability features in software models visible, better prepared usability engineers will be able to mediate between users and developers at requirements elicitation and specification time and to ensure that usability issues will be properly dealt with during the whole development cycle.

Acknowledgments. We thank CNPq (476381/2004-5) and the Cátedra Ibero-Americana Unicamp-Santander Banespa for supporting one of the authors.

References

1. Barbacci, M., Ellison, R., Lattanze, A., Stafford, J.A., Weinstock, C.B., Wood, W.G.: Quality Attribute Workshop, 3rd ed. CMU/SEI-2003-TR-016, Software Engineering Institute (2003)
2. Bass, L., John, B., Kates J.: Achieving Usability through Software Architecture. Technical Report. CMU/SEI-2001-TR-005, Software Engineering Institute, CMU (2001)
3. Bass, L., John. B.: Linking Usability to software architecture patterns through general scenarios. The Journal of Systems and Software. vol 66 (3) (2003) 187-197
4. Bass, L., John, B., Juristo, N., Sanchez, M.I.: Usability Supporting Architectural Patterns**.** Tutorial International Conference on Software Engineering (2004)
5. Bevan, N.: Quality and usability: A new framework. In: van Veenendaal, E, and McMullan, J (eds) Achieving software product quality. The Netherlands (1997) 25-34
6. Bevan, N. Quality in Use for All. In: User interfaces for all. Stephanidis, C (ed), Lawrence Erlbaum (1999)
7. Bias, R.G., Mayhew D.J.: Cost-Justifying Usability. Elsevier (2005)
8. Bosch, J., Juristo N., Designing Software Architectures for Usability. Tutorial International Conference on Software Engineering (2003)
9. Brighton.: Usability Pattern Collection. (2003)www.cmis.brighton.ac.uk/research/patterns/
10. Buschmann, F., Meuneir, R., Rohnert, H., Sommerland, P., Stal. M.: Pattern-Oriented Software Architecture, A System of Patterns. J. Wiley and Sons (1996)
11. Chirstel, M. G., Kang K.C.: Issues in Requirements Elicitation. Technical Report CMU/SEI-92-TR-012, Software Engineering Institute, CMU (1992)
12. Constantine, L., Lockwood. L.: Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design. Addison-Wesley (1999)
13. Coram, T., Lee, L. Experiences: A Pattern Language for User Interface Design (1996) http://www.maplefish.com/todd/papers/experiences/Experiences.html
14. Faulkner, X., Culwin, F.: Enter the Usability Engineer: Integrating HCI and Software Engineering. ITICSE ACM (2000) 61-64
15. Ferré, X., Juristo, N., Moreno, A.: Which, When and How Usability Techniques and Activities Should be Integrated. In Seffah, A, Gulliksen, J, Desmarais, M.C. (eds.) Human-Centered Software Engineering - Integrating Usability in the Software Development Lifecycle, Human-Computer Interaction Series, vol. 8 Kluwer (2005)
16. Folmer, E., van Group, J., Bosch J.: Architecting for usability: a survey. Journal of Systems and Software, vol. 70 (1-2) (2004) 61-78
17. Guide to the Software Engineering Body of Knowledge. (2004) version. http://www.swebok.org

18. Holzinger, A.: Usability Engineering Methods for Software Developers. Communications of the ACM, vol. 48 (1) January (2005) 71-74
19. Hix, D., Hartson. H.R.: Developing User Interfaces: Ensuring Usability through Product and Process. J. Wiley and Sons (1993)
20. ISO/IEC FCD 9126-1 Software product quality - Part 1: Quality model (1998)
21. ISO 9241-11 Ergonomic requirements for office work with visual display terminals (1998)
22. ISO 13407 User-centred design process for interactive systems (1999)
23. ISO/IEC 14598-1 Information Technology - Evaluation of Software Products Part 1 General guide (1998)
24. Jokela, T.: Guiding Designers to the World of Usability: Determining Usability Requirements through Teamwork. In Seffah, A., Gulliksen, J., Desmarais, M. (eds) Human-Centered Software Engineering. Kluwer (2005)
25. Juristo, N., Moreno, A., Sánchez M.: Architectural Sensitive Usability Patterns. ICSE Workshop Bridging the Gaps between Usability and Software Development (2003)
26. Juristo, N., Moreno, A., Sánchez M.: Techniques and Patterns for Architecture-Level Usability Improvements. Deliv. 3.4 STATUS project, May (2003) Http://www.ls.fi.upm.es/status
27. Juristo, N., Moreno, A., Sánchez M.: Clarifying the Relationship between Software Architecture and Usability. 16th International Conference on Software Engineering and Knowledge Engineering (2004)
28. Juristo, N., Moreno, A., Sánchez M.: Analysing the impact of usability on software design. Journal of System and Software. Accepted for publication January 2007 (2007)
29. Unified Modeling Language (UML). Version 2.0. OMG Object Management Group (2007) http://www.omg.org/technology/documents/formal/uml.htm. Visited January 2007
30. Laasko, S. A.: User Interface Designing Patterns. (2003) http://www.cs.helsinki.fi/u/salaakso/patterns/index_tree.html Visited October 2004
31. Milewski, A.E.: Software Engineers and HCI Practitioners Learning to Work Together: A Preliminary Look at Expectations. Proceedings of the 17th Conference on Software Engineering Education and Training CSEET´04 IEEE (2004)
32. Nielsen, J.: Usability Engineering. John Wiley & Sons (1993)
33. Perry, D., Wolf, A.: Foundations for the Study of Software Architecture. ACM Software Engineering Notes, vol 17 (4), October (1992) 40-52
34. QUIS^{TM} Questionnaire For User Interaction Satisfaction. http://lap.umd.edu/QUIS/
35. Rubinstein, R, Hersh, H.: The Human Factor. Digital Press, Bedford, MA. (1984)
36. Seffah, A., Djouab, R., Antunes, H.: Comparing and Reconciling Usability-Centered and Use Case-Driven Requirements Engineering Processes. IEEE (2001) 132-139
37. Seffah, A., Metzker, E.: The Obstacles and Myths of Usability and Software Engineering. Communications of the ACM. December (2004) vol 47 (12) 71-76
38. Shneiderman, B.: Designing the User Interface: Strategies for Effective Human-Computer Interaction. Addison-Wesley (1998)
39. Sommerville, I.: Software Engineering. 7th edition, Pearson Education (2004)
40. Tidwell, J.: Common Ground: A Pattern Language for Human-Computer Interface Design (1999)http://www.mit.edu/%7Ejtidwell/interaction_patterns.html
41. Tidwell, J. Designing Interfaces. Patterns for Effective Interaction Design. O´Reilliy, USA (2005)
42. Welie M. Amsterdam Collection of Patterns in User Interface Design (2003) http://www.welie.com/