

Moving usability forward to the beginning of the software development process

Natalia Juristo*, Ana Moreno*, Maria-Isabel Sanchez-Segura‡

**School of Computing - Universidad Politécnica de Madrid
Spain*

*‡Department of Computing - Universidad Carlos III de Madrid
Spain*

1. Introduction

Software usability is a quality attribute found in a number of classifications (IEEE, 1998), (ISO9126, 1991), (Boehm, 1978). Nielsen gave one of the most well-known descriptions related to software system learnability and memorability, efficiency of use, ability to avoid and manage user errors, and user satisfaction (Nielsen, 1993). In spite of the relevance of usability in software development it is still insufficient in most software systems (Seffah, 2004) (Bias, 2005).

For the past two decades, software usability has been perceived, from a software development perspective, as related to the presentation of information to the user (Seffah & Metzker, 2004) (Folmer et al., 2004). 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 (Buschmann et al.)). This separation would make it easier to modify the user interface to improve usability without affecting the rest of the application. Accordingly, 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, however, usability's implications in the application core have been highlighted. Some authors have already illustrated, albeit informally, a possible relationship between usability and architectural design (Bass, 2003) (Folmer, 2004). If this relationship is confirmed, the cost of rework to achieve an acceptable level of usability would be much higher than expected according to the hypothesis of separation. If this is the case, usability should be dealt with earlier in the development process in order to define and evaluate its impact on design as soon as possible. Notice that this approach is consistent with the tendency in SE to carefully consider quality attributes early on in the development process (Barbacci, 2003). This strategy has already been applied to other quality attributes like performance, modifiability, reliability, availability and maintainability, where a number of authors have proposed techniques to deal with these attributes, for example, at architectural design time (Klein, 1999) (Bass 1999) (Eskenazi, 2002) (Bosch, 2003).

In this context, the objective of this paper is twofold. On the one hand, we will offer some evidence to demonstrate the relationship between usability and software design. To do this, we will first analyze what different sorts of impact usability heuristics and guidelines discussed in the human computer interaction (HCI) literature are likely to have on a software system. We have identified usability features with a potential impact on the user interface, on the whole development process, and on the design models. To confirm this potential impact, we are particularly interested in design models. Therefore, the next step in this book chapter is to examine what sort of effect such usability features have on software design. With this aim in mind, we have surveyed several real systems that have the usability features in question built in. By way of an illustration, we will detail our study in terms of new classes, methods and relationships derived from adding one particular usability feature. As a result of this analysis we are able to demonstrate that usability really does effect the system's core functionality. With the goal of quantifying as far as possible the effect of usability features with an impact on design, this book chapter goes on to discuss the data gathered from applying a number of usability features to develop several real systems. We used these data to demonstrate the relationship between usability and software design and to get an informal estimation of the implications of building these features into a system.

Consequently, we can demonstrate that particular usability issues have a real impact on software design. Such issues have big functional implications and, therefore, need to be considered from as of the early development phases to avoid design rework, like any other functional requirement.

Accordingly, the second objective of this paper is to discuss how to deal with such usability issues at requirements time. In particular, we present some completeness problems caused by incorporating functional usability features as requirements, and discuss how the traditional solutions for dealing with incompleteness are hard to apply in this case. Then we present the approach we followed to avoid such problems, using a pattern-oriented approach to capture the knowledge to be managed to elicit and specify usability requirements. Finally we show some results related to pattern use.

To achieve the above objectives, this book chapter has been structured as follows. Section 2 discusses the different usability recommendations that can be found in the HCI literature. Section 3 provides some evidence about the relationship between particular usability recommendations and software design. This evidence shows that incorporating such recommendations involves modifying the core of the software design, as required to build in any other functionality. Section 4 shows the problems of dealing with usability during the requirements phase, and section 5 discusses a pattern-based representation of usability recommendations that avoid such limitations. Finally, section 6 presents some data gathered from the evaluation of such patterns.

2. Usability Recommendations in the HCI Literature

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 (Nielsen, 1993), principles of usability (Constantine, 1998) (Shneiderman, 1999), usability

guidelines (Hix, 1993), etc. Although all 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” (Nielsen, 1993) or “support internal locus of control” (Shneiderman, 1999), and others that provide more definite usability solutions like “make the user actions easily reversible” (Hix, 1993) or “provide clearly marked exits” (Nielsen, 1993). It is not our aim to provide a detailed classification of these usability features, as this is outside the scope of software engineering. What we can do, though, is structure these features depending on their potential impact on software development. Accordingly, such features can be divided into three groups:

- 1) Usability recommendations with a potential impact on the UI. Examples of such recommendations refer to presentation issues like buttons, pull-down menus, colors, fonts, etc. Building these recommendations into a system involves slight modifications to the detailed UI design.
- 2) Usability recommendations with a potential impact on the development process, which can only be taken into account by modifying the development process itself, e.g. recommendations referring to reducing the user cognitive load, involving the user in software construction, etc.
- 3) Usability recommendations with a potential impact on the design. They involve building certain functionalities into the software to improve user-system interaction. We have termed these set of usability recommendations Functional Usability Features (FUFs). Examples of FUFs are providing cancel, undo, feedback, etc. Let’s suppose that we want to build the cancel functionality for specific commands into an application. To satisfy the requirements for this functionality 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 to add the cancel button, specific components should be built into the software design to deal with these responsibilities. Table 1 shows the most representative FUFs that can be foreseen to have a crucial effect on system design. This table also includes the HCI authors that suggest each recommendation.

In this chapter, we are interested in usability recommendations with a potential impact on design and try to provide real evidence about their impact on design.

3. Analyzing the Effect of Usability on Software Design

To study the relationship between FUFs and software design we have worked on a number of real development projects carried out by UPM Master in Software Engineering students as part of their MSc dissertations from 2004 to 2005. Students originally developed the respective systems without any FUFs. These designs were then modified to include the FUFs listed in Table1.

Functional Usability Features	Goal
FEEDBACK (Tidwell, 1999) (Brigthon, 1998) (Coram, 1996) (Welie, 2003) (Tidwell, 2005) (Nielsen, 1993) (Constantine, 1999) (Shneiderman, 1998) (Hix, 1993) (Rubinstenin, 1994) (Heckel, 1991)	To inform users about what is happening in the system
UNDO (Tidwell, 2005) (Welie, 2003) (Brigthon, 1998)	To undo system actions at several levels
CANCEL (Tidwell, 2005) (Brigthon, 1998) (Nielsen, 1993)	To cancel the execution of a command or an application
USER INPUT ERRORS PREVENTION/CORRECTION (Tidwell, 2005) (Brigthon, 1998) (Shneiderman, 1998) (Hix, 1993) (Rubinstein, 1984) (Constantine, 1999)	To improve data input for users and software correction as soon as possible
WIZARD (Welie, 2003) (Tidwell, 2005) (Constantine, 1998)	To help to do tasks that require different steps involving user input
USER PROFILE (Tidwell, 1999) (Welie, 2003) (Hix, 1993) (Rubinstenin, 1994) (Heckel, 1991)	To adapt system functionality to users' profile
HELP (Tidwell, 2005)(Welie, 2003)(Nielsen, 1993)	To provide different help levels for different users
COMMAND AGGREGATION (Nielsen, 1993) (Constantine, 1999) (Hix, 1993)	To help users to create commands to execute more than one task at a time
SHORTCUTS (Nielsen, 1993) (Constantine, 1999) (Hix, 1993) (Shneiderman, 1998)	To allow users to activate a task with one quick gesture.
REUSE INFORMATION (Constantine, 1999)	To allow users to easily move data from one part of a system to another

Table 1. Preliminary list of usability features with impact on software design

The projects used were interactive systems (an on-line table booking for a restaurant chain, an outdoor advertising management system, a car sales system, an adaptable surface transport network system, a computer assembly warehouse management system; an on-line theatre network ticket sales and booking system; and an employee profile and job offer processing and matching software system). We deliberately chose interactive systems because usability is more relevant in these cases, and FUFs can be expected to have a bigger impact.

For each of the systems to which the FUFs listed in Table 1 were added, we quantified a number of criteria:

- *FUF impact on system functionality* (FUF-Functionalities). This parameter mirrors the number of functionalities (in terms of expanded use cases) affected by the FUF in question. To assess this criterion we calculated the percentage of expanded use cases affected by each FUF, which was rated as low, medium or high depending on the interval to which the percentage belongs (under 33%, from 33% to 66%, over 66%).
- *FUF-derived classes* (FUF-Classes). This criterion refers to the number of classes that appear in the design as a result of adding a FUF. This has been assessed by calculating the percentage of new classes derived from the feature, which was rated as low, medium or high depending on the interval to which the percentage belongs (under 33%, from 33% to 66%, over 66%).
- *FUF-derived methods complexity* (FUF-Methods Complexity). The criterion refers to how complex the methods that need to be created as a result of incorporating a given FUF into the system are. It is not easy to provide a measure of the complexity of a method at design time. For the purposes of our study, however, we have classified the possible class methods based on their functionality as follows:
 - Methods related to displaying information, running checks, etc., have been rated as low
 - Methods related to filters, error corrections, etc., have been rated as medium.
 - Methods related to returning to the earlier state of an operation, saving the state, etc., have been rated as high.
- *Interaction with other system components* (FUF-Interaction). This parameter represents how the classes involved in FUF design couple with the other system classes. To assess this parameter, we measured the percentage of interactions between the FUF-derived classes or between these and other system classes that can be observed in the interaction diagrams. The value of this criterion will be low, medium and high depending on what third this percentage belongs to (under 33%, from 33% to 66%, over 66%).

The need to build different FUFs into a particular project will depend on the project features. For example, *shortcuts* will have a low value for impact on system functionality (FUF-Functionality) if we are dealing with a software system that will only be executed from time to time, whereas it will have a high value if the application runs continuously and performs the same tasks again and again. Similarly, the other FUFs could be designed to affect more or fewer parts of the software system. In our study, all usability features addressed were specified as being included in the whole system and related to the

maximum number of functionalities to which they applied. For example, when *feedback* was added, it was considered that the whole breadth of this feature was needed, including progress bars, clocks, etc., for all the tasks that have need of this functionality.

Additionally, the FUF-Classes, FUF-Methods and FUF-Interactions criteria will very much depend on the type of design. The values output for our systems should not be construed as absolute data. On the contrary, they are intended to illustrate to some extent what effect adding the respective FUFs could have on design.

Readers are referred to (Juristo et al. 2007) for details of this study for one of the above applications. Table 2 summarizes the mean values of the metrics derived from incorporating the FUFs in the above systems. It is clear from this table that the *cancel* and *undo* FUFs have the biggest impact on design. Not many more classes (FUF-Classes) are added (as in the chosen design a single class is responsible for saving the last state for whatever operations are performed, although another equally valid design could have envisaged a separate class to save the state for each operation). However, the complexity of the methods (FUF-Methods-Complexity) that need to be implemented is high, as is the number of interactions between the different classes (FUF-Interactions). In the *cancel* case especially, this feature is closely related to all system functionalities (FUF-Functionalities), because the HCI literature recommends that easy exit or cancellation should be provided for each and every one of the tasks that the user uses the system to do (Tidwell, 99).

Another FUF with a big impact on all system functionality is *feedback*. Apart from the *system status feedback* discussed in the last section, the HCI literature also recommends that the user should receive feedback reporting the *progress* of the operations when the user is doing long tasks (Tidwell, 2005) (Brighon, 1998) (Coram, 1996) (Welie, 2003), when the *tasks are irreversible* (Brighon, 1998) (Welie, 2003) and, additionally, every time the user *interacts* with the system (Brighton, 2003). It is this last recommendation especially that leads to the high FUF-functionality for this feature, as it means that feedback affects all a software system's non-batch functionalities.

On the other hand, we find that the impact of adding other FUFs, like for example *user profile*, are less costly because they can be easily built into a software system and do not interact very much with the other components. A similar thing applies to *help*. In this case, though, despite its low impact on functionality (because this functionality was designed as a separate use case, yielding a 5% and therefore low FUF-functionality value), its interaction is high, as it can be called from almost any part of the system.

It is noteworthy that no big differences were found among the applications because they were similar, i.e. they were all management systems. Note that these same FUFs may have a slightly different impact on other software system types, for example, control systems (in which FUFs like *user input errors prevention/correction* or *commands aggregation* may have a bigger impact than shown in Table 2 due to the criticality of the tasks performed) or less interactive systems (in which *feedback* or *cancel* will have less impact).

In sum, the data in Table 2 confirm that some usability recommendations, in particular the ones we have named FUFs, affect the core functionality of a software system. As with any other functionality, specific design components will have to be created to build such FUFs into a software application. The approach we take is to consider such FUFs as functional

requirements and deal with them during the requirements process as any other functionality. The rest of the chapter focuses on how to address this proposal.

Summary	FUF-Functionality	FUF-Classes	FUF-Methods Complexity	FUF-Interactions
Feedback	HIGH 90%	LOW 27%	MEDIUM	MEDIUM/HIGH 66%
Undo	MEDIUM 40%	LOW 10%	HIGH	MEDIUM/HIGH 66%
Cancel	MEDIUM 95%	LOW 8%	HIGH	MEDIUM/HIGH 66%
User Input Errors Prevention/Correction	MEDIUM 36%	LOW 11%	MEDIUM	LOW 6%
Wizard	LOW 7%	LOW 10%	LOW	HIGH 70%
User Profile	LOW 8%	MEDIUM 37%	MEDIUM	LOW 10%
Help	LOW 7%	LOW 6%	LOW	HIGH 68%
Commands aggregation	LOW (10%)	LOW (5.8%)	MEDIUM	LOW 15%

Table 2. Mean values for design impact of FUF

4. Limitations of Usability Requirements

The idea of dealing with usability at the requirements phase is not new. Both HCI (Jokela, 2005) and SE (Swebok, 2004) have considered usability as a non-functional requirement. In this context, 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 “End user satisfaction with the application should be higher than Z on a 1-to-5 scale”. Dealing with usability in the shape of non-functional 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 (Cysneiros, 2005). We propose a complementary approach in which usability features with major implications for software functionality, FUFs, are incorporated as 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. However, descriptions like these provide 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 ((Tidwell, 1996)(Welie, 2003)(Laasko, 2003)(Brighton, 1998)(Coram, 1996)(Benson, 2002)) 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 during 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 (Kovitz, 2002)(Benson, 2002). However, in this case, 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, what is best for each situation, and less still 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 (Kazman et al, 2003).

The HCI literature suggests that HCI experts should join software development teams to provide this missing expertise (ISO, 1999)(Mayhew, 1999). 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 (Seffah & Metzker, 2004). 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 Usability Elicitation Patterns

Our approach consists of packaging guidelines that empower developers to capture functional usability requirements using the information provided by the HCI literature as input. We have analyzed this information from a software development point of view and have elaborated elicitation and specification guidelines that have been packaged in a pattern format.

The first task was to analyze the different varieties of usability features identified by HCI authors. We denoted these subtypes as usability mechanisms, and gave them a name that is indicative of their functionality (see Table 3)

Then we defined the elicitation and specification guides for the usability mechanisms, focusing on the information provided by HCI authors. We analyzed and combined all the recommendations on the same mechanism, and then removed redundancies. The resultant 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 stakeholders to properly specify such usability features.

The outcome of the previous tasks is packaged in what we call a usability elicitation pattern. Other authors have already used patterns 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 (Hagge, 2005)(Repare, 2005), for example. In (Whitenak, 1995), the author proposes twenty patterns to guide the analyst through the application of the best techniques and methods for the elicitation process.

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

We have developed one usability elicitation pattern for each usability mechanism in **Pogreška! Izvor reference nije pronađen.**³ (second column). They are available at <http://is.ls.fi.upm.es/research/usability/usability-elicitation-patterns>. Table 4 shows an example of the elicitation pattern for the System Status Feedback mechanism.

The developer can use the *identification* part of the pattern to find out the basics of the usability mechanism 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, it will not be used. Otherwise, the respective usability functionality will be elicited and specified using the *solution* part of the pattern.

The solution part of the pattern 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 should be considered, alongside the respective HCI rationale (i.e. the HCI recommendation used to derive the respective issues). Developers should read and understand the HCI rationales in the guide. This will help them to understand why those issues need to be discussed with stakeholders.

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. An example of the application of this usability elicitation pattern is given in (Juristo, et al, 2007a) .

Usability Feature	Usability Mechanism	HCI Authors' Label	Goal
Feedback	System Status	Modeless Feedback Area (Coram, 1996) Status Display (Tidwell, 1996)	To inform users about the internal status of the system
	Interaction	Interaction Feedback (Brighton, 1998) Modeless Feedback Area (Coram, 1996) Let Users Know What is Going On (Benson, 2002)	To inform users that the system has registered a user interaction, i.e. that the system has heard users
	Warning	Think Twice (Brighton, 1998) Warning (Welie, 2003)	To inform users of any action with important consequences
	Long Action Feedback	Progress Indicator (Tidwell, 1996) (Tidwell, 2005) Show Computer is Thinking (Brighton, 1998) Time to Do Something Else (Brighton, 1998) Progress (Welie, 2003) Modeless Feedback Area (Coram, 1996) Let Users Know What is Going On (Benson, 2002)	To inform users that the system is processing an action that will take some time to complete
Undo Cancel	Global Undo	Multi-Level Undo (Tidwell, 1996) (Tidwell, 2005) Undo (Welie, 2003) Global Undo (Laasko, 2003) Allow Undo (Brighton, 1998) Go Back One Step (Tidwell, 1996)	To undo system actions at several levels
	Object-Specific Undo	Object-Specific Undo (Laasko, 2003)	To undo several actions on an object
	Abort Operation	Go Back One Step (Tidwell, 1996) Emergency Exit (Brighton, 1998) Cancellability (Tidwell, 2005) □	To cancel the execution of an action or the whole application
	Go Back	Go Back to a Safe Place (Tidwell, 1996) Go Back One Step (Tidwell, 1996)	To go back to a particular state in a command execution sequence
User Input Error Prevention/Correction	Structured Text Entry	Forms, Structured Text Entry (Tidwell, 1996) Structured Format (Tidwell, 2005) Structured Text Entry (Brighton, 1998)	To help prevent the user from making data input errors
Wizard	Step-by-Step Execution	Step-by-Step (Tidwell, 1996) Wizard (Welie, 2003) (Tidwell, 2005) □	To help users to do tasks that require different steps with user input and correct such input
User Profile	Preferences	User Preferences (Tidwell, 1996) Preferences (Welie, 2003)	To record each user's options for using system functions
	Personal Object Space	Personal Object Space (Tidwell, 1996)	To record each user's options for using the system interface.
	Favorites	Favorites (Welie, 2003) Bookmarks (Tidwell, 1996)	To record certain places of interest for the user
Help	Multilevel Help	Multilevel Help (Tidwell, 2005)	To provide different help levels for different users
Command Aggregation	Command Aggregation	Composed Command (Tidwell, 1996) Macros (Tidwell, 2005)	To express possible actions to be taken with the software through commands that can be built from smaller parts.

Table 3. Usability mechanisms for which usability elicitation and specification guides have been developed

IDENTIFICATION	
Name: System Status Feedback	
Family: Feedback	
Alias: Status Display Modeling Feedback Area (Coram, 1996)	
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 (Tidwell, 1996)	<i>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.</i> 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	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
being confused with another. They should never be re-arranged, unless users do so themselves. Attention should be drawn to important information with bright colors, blinking or motion, sound or all three – but a technique appropriate to the actual importance of the situation to the user should be used (Tidwell, 1996).	<i>highlighted because it is related to an important but non-critical situation? Using different colors and sound or motion, sizes, etc.</i> 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

HCI Rationale	Issue to discuss with stakeholders
<p>being confused with another. They should never be re-arranged, unless users do so themselves. Attention should be drawn to important information with bright colors, blinking or motion, sound or all three - but a technique appropriate to the actual importance of the situation to the user should be used (Tidwell, 1996).</p>	<p><i>highlighted because it is related to an important but non-critical situation? Using different colors and sound or motion, sizes, etc.</i></p> <p><i>2.4. Which of this information will be simply displayed in the status area? For example, providing some indicator.</i></p> <p>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 colors, sounds, motions 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.</p>

Table 4. (a) System status feedback usability elicitation pattern

SOLUTION (Cont.)	
Usability Mechanism Elicitation Guide (Cont.):	
HCI Rationale (Cont.)	Issue to discuss with stakeholders (Cont.)
<p>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 (Coram, 1996). 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 (Constantine, 1998). 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 (Tidwell, 1996).</p>	<p><i>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.</i></p> <p><i>3.2. Which is the best place to locate the feedback information for each situation?</i></p>

Usability Mechanism Specification Guide:
<p>The following information will need to be instantiated in the requirements document.</p> <ul style="list-style-type: none"> - 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 PATTERNS¹:

Table 4. (b) System status feedback usability elicitation pattern (cont.)

5. Preliminary Evaluation of Usability Elicitation Patterns

The potential benefits of the usability elicitation patterns have been evaluated at different levels.

We studied how useful the patterns were for building 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 SE Master students. In particular, we worked with five groups of three students. Each group was given a different software requirement specification document (for a theatre tickets sale system, for a PC storage and assembly system, for a temping agency job offers management system, for a car dealer vehicle reservation and sale system, and for a travel agency bookings and sale system). All the systems were real applications,

¹ *Related patterns* refer to other usability elicitation patterns whose contexts are related to the one under study and could also be considered in the same application. In this case, no related patterns have being identified. However, readers are referred to other patterns, like Long Action Feedback or Abort Operation, at the above-mentioned web site.

and each one was randomly allocated to a group. 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 corresponding usability functionality.
- Another student was given reduced patterns. See Appendix, including the reduced pattern for System Status Feedback, to get a taste of the difference between the reduced and full patterns. This short pattern is just a compilation of information from the HCI literature about the usability mechanisms. We have not elaborated this information from a development perspective, i.e. the reduced patterns do not include the "Issues to be discussed with stakeholders" column in Table 3. 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, no patterns) to prevent student characteristics from possibly biasing the final result.

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 (Constantine, 1998) (Shneiderman, 1999)(Nielsen, 1993).

6.1. Users' usability evaluation

The usability evaluations conducted by users are based on usability tests in which the users state their opinion about the system. We used an adaptation of the QUIS usability test (QUIS, 2007). Each test question is scored on a scale of 1 (lowest usability) to 5 (highest usability). The final usability score is the mean of the responses to each question. We worked with three representative users for each system. Each user evaluated the three versions of each application (the one developed with the full patterns, with the reduced patterns and with no patterns) in different order.

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, 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 $p\text{-value} < 0.01$). 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.

6.2. 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 5 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 the case of feedback, for example, the developers that used the respective elicitation patterns included, on average, 94% of the functionalities associated with this mechanism. Developers that used the reduced patterns incorporated 47% of the respective functionalities. Finally, developers that used no pattern included only 25%.

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 last column of Table 5 with $p\text{-value} < 0.01$ in all cases). Again the Tamhane test showed that all 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.

	Full usability elicitation patterns	Reduced patterns	No pattern	Kruskal-Wallis(chi-square; p-value)
Feedback	94%	47%	25%	12,658; 0,002*
Undo/Cancel	90%	66%	43%	12,774; 0,002*
User Profile	95%	80%	65%	12,597; 0,002*
Users Input Errors Prevention/Correction	97%	85%	72%	12,727; 0,002*

	Full usability elicitation patterns	Reduced patterns	No pattern	Kruskal-Wallis(chi-square; p-value)
Wizard	100%	89%	71%	13,109; 0,001*
Help	100%	81%	74%	13,109; 0,001*

* Statistically significant at 99% of confidence

Table 5. Mean percentage of functionality added for each usability mechanism by each information type

Note that the functionality added using the full elicitation patterns is less than 100% 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% 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.

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.

7. Conclusions

The goal of this chapter was first to provide some data about the impact of including particular usability recommendations in a software system. The data gathered show that building certain usability components into a software system really does entail significant changes to the software system design. Therefore, it is important to move usability issues

forward to the early development phases, i.e. to requirements time (like any other functionality). However, this is not a straightforward objective primarily due to the fact that development stakeholders are not acquainted with HCI.

We propose a possible solution to overcome these snags. To do this, 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. 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.

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, then, a balance has to be struck between user availability, time and cost constraints, on the one hand, and usability results, on the other, at the beginning of development.

8. References

- Andrés, A., Bosch, J., Charalampos, A., Chatley, R., Ferre, X., Folmer, E., Juristo, N., Magee, J., Menegos, S., Moreno, A. 2002. Usability Attributes Affected by Software Architecture. Deliverable. 2. STATUS project. Available at: <http://www.ls.fi.upm.es/status>
- Barbacci M, Ellison R., Lattanze A., Stafford J.A, Weinstock C.B., Wood W.G. 2003. Quality Attribute Workshop, 3rd ed. CMU/SEI-2003-TR-016, Pittsburgh, Software Engineering Institute, Carnegie Mellon University, USA.
- Bass, L. and John, B.E. 2003. Linking usability to software architecture patterns through general scenarios. *Journal of Systems and Software*, Volume 66, Issue 3, pp. 187-197
- Bass, L., et al. 1999. Architecture-Based Development. CMU/SEI-1999-TR-007. SEI/CMU.
- Bass, L., John, B., Kates, J. 2001. Achieving Usability Through Software Architecture. Technical Report. CMU/SEI-2001-TR-005
- Battey, J. 1999. IBM's redesign results in a kinder, simpler web site. Available at: <http://www.infoworld.com/cgi-bin/displayStat.pl?pageone/opinions/hotsites/hotextr990419.htm> .
- C. Benson, A. Elman, S. Nickell, C. Robertson. GNOME Human Interface Guidelines. <http://developer.gnome.org/projects/gup/hig/1.0/index.html>
- Bias, R.G., Mayhew D.J. 2005. Cost-Justifying Usability. An Update for the Internet Age. Elsevier.
- Black, J. 2002. Usability is next to profitability. *BusinessWeek Online*. Available at: http://www.businessweek.com/technology/content/dec2002/tc2002124_2181.htm
- Boehm B., et al. 1978. Characteristics of Software Quality. North Holland, New York.

- Bosch, J., Lundberg, L. 2003. Software architecture - Engineering quality attributes. *Journal of Systems and Software*, Volume 66, Issue 3, pp. 183-186.
- Brighton. 2003. Usability Pattern Collection. <http://www.cmis.brighton.ac.uk/research/patterns/>
- Constantine, L., Lockwood, L. 1999. *Software for Use. A Practical Guide to the Models and Methods of Usage-Centered Design*. Addison-Wesley.
- F. Buschmann, R. Meuneir, H. Rohnert, P. Sommerland, M. Stal. *Pattern-Oriented Software Architecture, A System of Patterns*. J. Wiley and Sons, 1996
- Coram, T., Lee, L. 1996. *Experiences: A Pattern Language for User Interface Design*. <http://www.maplefish.com/todd/papers/experiences/Experiences.html>
- Chidamber, S., Kemerer, C. 1994. A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*, June, pp. 476-492.
- Donahue G. M. 2001. Usability and the Bottom Line. *IEEE Software*, vol. 16, n. 1. pp 31-37.
- Eskenazi, E.M., Fioukov, A.V., Hammer, D.K., Obbink, H., 2002. Performance Prediction for Software Architectures, in *Proceedings of PROGRESS 2002 workshop*, Netherlands.
- Ferre, X., Juristo, N., Moreno, A. 2006. Integration of HCI Practices into Software Engineering Development Processes: Pending Issues. *Encyclopedia of Human-Computer Interaction*. pp. 422-428. C. Ghaoui (ed.). Idea Group Inc.
- Fetcke, T., Abran, A., Nguyen, T., 1997. Mapping the OO - Jacobson. Approach into Function Point Analysis. *Software. Technology of Object-Oriented Languages and Systems. Proceedings of TOOLS 1997*.
- Folmer, E., Group, J., Bosch, J. 2004.. Architecting for usability: a survey. *Journal of Systems and Software*, vol 70. pp. 61-78.
- Griffith, J. 2002. Online transactions rise after bank redesigns for usability. *The Business Journal*. 2002. Available at: <http://www.bizjournals.com/twincities/stories/2002/12/09/focus3.html>
- L. Hagge. K. Lappe. "Sharing Requirements Engineering Experience Using Patterns". *IEEE Software*. Jan-Feb 2005, pp. 24-31.
- Heckel, P. 1991. *The elements of friendly software design*. (2nd ed.) CA: Sybex Inc.
- Hix, D., Hartson, H. R.. 1993. *Developing User Interfaces: Ensuring Usability Through Product and Process*. John. Wiley & Sons, New York.
- IBM, 2005. Cost Justifying Ease of Use. Available at: http://www-3.ibm.com/ibm/easy/eou_ext.nsf/Publish/23 (current 18 May. 2005).
- IEEE. 1998. IEEE Std 1061: Standard for a Software Quality Metrics Methodology.
- ISO. 1998. ISO 9241-11, 98: Ergonomic Requirements for Office work with Visual Display Terminals. Part 11: Guidance on Usability. ISO.
- ISO. 2000. ISO 18529, 00: Human-Centered Lifecycle Process Descriptions. ISO.
- ISO/IEC, 1991. ISO 9126: Information Technology - Software quality characteristics and metrics.
- ISO/IEC. 1999, ISO14598-1, 99: Software Product Evaluation: General Overview. ISO/IEC.
- ISO Std 13407: Human-Centred Design Processes for Interactive Systems. ISO, 1999.1
- T. Jokela. "Guiding Designers to the World of Usability: Determining Usability Requirements through Teamwork". In *Human-Centered Software Engineering*. A. Seffah, J. Gulliksen and M. Desmarais, Kluwer 2005

- Juristo, N., Moreno A.M., Sánchez-Segura, M. Guidelines for Eliciting Usability Functionalities (pp. 744-758). IEEE Transactions on Software Engineering. November 2007, vol 33 (11).
- Juristo N., Moreno A.M., Sánchez-Segura M. Analysing the Impact of Usability on Software Design (pp:1506 - 1516). Journal of System and Software. Vol. 80(9) September 2007.
- R. Kazman, J. Gunaratne, B. Jerome. "Why Can't Software Engineers and HCI Practitioners Work Together?" In Human-Computer Interaction Theory and Practice. C. Stephanidis, L. Erlbaum (Eds.). Elsevier, 2003.
- Klein, M., et al. 1999. Attribute-Based Architectural Styles. CMU/SEI-99-TR-022. SEI/CMU.
- B. Kovitz. "Ambiguity and What to Do about It". IEEE Joint International Conference on Requirements Engineering 2002 (Key talk).
- Laasko, S. A. 2003. User Interface Designing Patterns. http://www.cs.helsinki.fi/u/salaakso/patterns/index_tree.html Visited October 2004.
- Mayhew, D.J. 1999. The Usability Engineering Lifecycle. Morgan Kaufmann.
- McKay, E.N. 1999. Developing User Interfaces for Microsoft Windows, Microsoft Press
- Nielsen, J. 1993. Usability Engineering, AP Professional, Boston, Mass.
- Perry, D., Wolf, A. 1992. Foundations for the Study of Software Architecture. ACM Software Engineering Notes, vol 17 (4), pp. 40-52.
- Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., Carey, T. 1994. Human-Computer Interaction. Addison Wesley.
- QUISTM Questionnaire For User Interaction Satisfaction. <http://lap.umd.edu/QUIS/REPAIRE> <http://repare.desy.de/Repare/RepareController>.
- Rubinstein, R, Hersh, H. 1984. The Human Factor. Digital Press, Bedford, MA.
- Scapin, D. L., Bastien, J.M. C. 1997. Ergonomic criteria for evaluating the ergonomic quality of interactive systems, Behaviour & Information Technology, vol 16, no 4/5, pp.220-231.
- Seffah, A., Metzker E. 2004. The Obstacles and Myths of Usability and Software Engineering. Communications of the ACM, Vol. 47(12), pp. 71-76.
- Shneiderman, B. 1998. Designing the User Interface: Strategies for Effective Human-Computer Interaction. (3rd ed. ed.). Menlo Park, CA: Addison Wesley.
- SWEBOK. Guide to the Software Engineering Body of Knowledge. 2004 Version. <http://www.swebok.org>
- Thibodeau, P. 2002. Users Begin to Demand Software Usability Tests. ComputerWorld. Available at: <http://www.computerworld.com/softwaretopics/software/story/0,10801,76154,00.html>
- Tidwell, J. 2005. Designing Interfaces. Patterns for Effective Interaction Design. O'Reilly, USA.
- Tidwell, J. 1999. Common Ground: A Pattern Language for Human-Computer Interface Design. http://www.mit.edu/~7Ejtidwell/interaction_patterns.html
- Trenner L., et al. 1998. The Politics of Usability. Springer, London, UK.
- Welie M. 2003. Amsterdam Collection of Patterns in User Interface Design. <http://www.welie.com/>

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