Gathering Usability Information through Elicitation Patterns

Natalia Juristo[1], Ana Moreno[1], Maribel Sanchez-Segura[2], Al Davis [3]

[1]*School of Computing*
Universidad Politécnica de Madrid, Spain
natalia@fi.upm.es
ammoreno@fi.upm.es

[2]*Department of CS*
Universidad Carlos III de Madrid, Spain
misanche@inf.uc3m.es

3 College of Business
University of Colorado at Colorado Springs, CO, USA
adavis@uccs.edu

Abstract

Like any other quality attribute, usability cannot be achieved at the last development moment. On the contrary, usability has to be taken into account right from the earliest stages by considering features that raise the usability level of the software system as well as usability requirements that enhance the performance of other functional requirements. However, existing HCI usability heuristics are not sufficient to serve as proper requirements from a software engineering perspective. HCI heuristics for incorporating usability features lead to incomplete and ambiguous requirements. For example, the classical approach for reducing requirements ambiguity, related to incorporating new information, cannot be used for usability requirements, since the information that would need to be added is likely to be beyond the usability knowledge of most requirements writers, developers and users. One possible solution to this problem is to bring HCI experts into the software development process. However, this is not a straightforward approach and may have serious implications for the development process. In this paper, we propose an alternative solution based on the definition of usability elicitation patterns. These patterns capitalise upon the know-how in usability requirements elicitation by specifying fundamental elements recurrently intervening in usability requirements elicitation. We have focused on usability features with major implications in functionality.

Keywords: usability requirements, usability requirements elicitation, requirements elicitation

1. Introduction

Usability is considered a critical aspect for interactive software systems [1][2]. The benefits related to cost savings and revenue increases thanks to usability improvements are also well recognised in the software community [3][4][5].

However, usability is still not a prominent feature of software systems.

Several reasons can help to explain this situation. One possible factor relates to business concerns, such as time-to-market pressure, cost limitations, or even low awareness of the relevance of usability. Another factor, and one we consider to be even more significant relates to technical issues such as the inherent difficulty of incorporating usability features into a software system.

In this paper, we focus on some of the technical problems that crop up when dealing with usability requirements and, particularly, the difficulties software developers find in gathering all the information needed to fully specify these requirements properly.

Section 2 discusses how usability requirements are related to functional requirements and, as such, should be dealt with from the beginning of the development process. Section 3 presents ambiguity problems that occur when trying to incorporate usability features as functional requirements and discusses how the traditional approaches for settling ambiguities are hard to apply. Section 4 presents a pattern-based solution for dealing with usability feature requirements that will support usability feature elicitation and specification by software practitioners. Finally, section 5 shows some validation results collected after applying the proposed solution.

2. Implications of Usability for Functional Requirements

The generally accepted proposal for dealing with usability features at the requirements stage is to treat them as non-functional requirements [6][7], establishing what usability values the system should achieve. These are then used as a reference standard at the subsequent product evaluation stage. For example, task X should be performed by a novice user in less than Y minutes, or end user satisfaction with the application should be higher than Z on a 1-to-5 scale. This way of dealing with usability in requirements has traditionally led to the widespread assumption that usability has few, if any, implications for system functionality.

Although some HCI authors have illustrated the close relationship between usability and software functionality [2], the widespread belief in software engineering (SE) is that the implications of usability can be detached from primary system functionality, as generally accepted design strategies like the separation of the presentation layer from the application layer (see, for example, well-known architectural models, like MVC architecture [8]) appear to back.

Among the heuristics for building usable software [1][2][7][9] (see [10] for a detailed analysis), we find particular features that represent concrete functionalities in a software system (undo, cancel, feedback, aggregation of commands, definition of user profiles, wizards, etc.). Several SE authors have studied the implications of these usability features in a software design [11][12] concluding that their inclusion in a software system requires such a hard design rework that, like any other functionality, they should be dealt with starting at the requirements stage.

So, although it might be true that some particular usability issues, mainly the ones related to the pure interface (colours, fonts, distribution of elements in the screen, etc.) can be modified with low cost and therefore do not need to be considered right from the beginning of the development process, other usability features have serious implications on the software functionality and therefore must be addressed earlier.

In other words, usability is not a software attribute that can be achieved at the last development moment. We claim that usability has important implications for system functionality and, consequently, the system's functional requirements, and should, therefore, not be treated as just non-functional requirements. Non-functional usability requirements may be useful for evaluation purposes, but are not enough for developing a usable software system. Features that raise the usability level of software systems need to be contemplated explicitly, and should be treated just like other functional requirements.

This task is, however, not void of difficulties, as the next section shows.

3. Problems with Treating Usability as Functional Requirements

Once we know that particular usability features should be considered as functional requirements, one might think that such requirements could be specified by just stating the corresponding usability features, for example, the system should provide users with the ability to cancel actions or the system should provide feedback to the user. In fact, this is the level of advice that HCI heuristics provide, assuming that it is sufficient for developers to incorporate a given usability feature into their designs. For example, one of the most commonly recurring rules of thumb is related to the feedback feature. Nielsen [9] describes the feedback design heuristic as: "Visibility of system status: The system should always keep users informed about what is going on through appropriate feedback within reasonable time".

However from a SE perspective, the above description provides very limited information, and nowhere near enough to properly specify the feedback functionality, let alone design and implement it correctly. In fact, there are four types of feedback: *interaction feedback* to let users know 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, or *warnings* to prevent users from taking irreversible action. Although they roughly match the above description, a developer with no expertise in usability or HCI will find it difficult or, perhaps, impossible, to use this description as a satisfactory specification for describing the all the functional requirements relating to feedback. In other words, the descriptions provided by HCI for usability features that have a big impact on system functionality lead to ambiguous and incomplete requirements.

In fact, much more information is needed to be able to properly incorporate the feedback feature into a software system. Besides the different kinds of feedback that can be provided and to which type of functionality they affect, a lot more information needs to be elicited to completely and unambiguously specify each feedback type. For example, HCI experts [13] suggest that system status information should always be presented to the user in the same place (so users know where to look to find this information) or that the information displayed should be obtrusive or unobtrusive depending on the criticality of the situation. So, issues like:

- which status changes in the software will the user want to be notified of (e.g., software failures, external resources

failures, changes in the internal status of the software, which changes, which states, …)
- how critical are each of the above situations,
- what kind of information needs to be displayed to the user in each case,
- etc.

need to be discussed with the different stakeholders and elicited in order to properly specify the system status feedback. Otherwise, the specification generated will be incomplete and ambiguous.

Note that, generally, the problem of reducing ambiguity in functional requirements is solved by adding more domain information to the requirements [14][15] from either the user, client or software engineer.

However, this solution is hard to apply to functional usability requirements because neither the users nor the developers are, in most cases, likely sources of the information to be added. Users know that they want feedback. What they do not know is, for example, what kind of feedback can be provided, and which one is most suitable for each situation (information that HCI experts do have). On the other hand, software engineers with design and development skills do not have the necessary HCI knowledge either. Even if they are given the requirement that the software system should be usable or, better still, that the software system should have undo, feedback or any other usability features, their HCI inexperience means that they do not know all the details that need to be specified to properly incorporate such features into a requirements specification. As Kazman et al. report, developers do not learn HCI-related information on their own [16].

In sum, although ambiguity in requirements is hard to deal with, it can be addressed in the case of functional domain requirements by gathering new information from experts in the domain. However, this is not a viable solution for functional usability requirements, as their proper and full specification is beyond the usability knowledge of average developers and users.

One way to address this shortage of HCI expertise could be to incorporate HCI experts into the software development team (as suggested by the HCI field [32][33]). However, there are at least two problems with this solution. The first is that as HCI is a different discipline from SE, difficulties in communication between the software team and HCI experts may arise. As Sheffard and Metzker point out [17], the causes of this communication breakdown are to be found in the use of different vocabulary, notations, software development strategies, etc. Such difficulties can be a considerable obstacle having a big impact on software construction. The second impediment is the cost of this solution. Many small- to medium-sized software development companies cannot afford to engage an HCI expert to work on their development teams.

We propose an alternative approach that uses a pattern-based solution to support information elicitation and specification for usability features with an impact on functional requirements.

4. A Pattern-Based Solution For Gathering Functional Usability Requirements

4.1. Usability Elicitation Patterns

The concept of pattern was introduced by Alexander in 1977 in the building architecture domain [18] and is considered a solution to a recurrent problem in a context. In software development, the concept of pattern has been used extensively for different purposes. For example, we can find design patterns [19] that propose solutions to common design problems; architectural patterns [20] that propose skeletons of well-known architectural models; or business patterns that set out classical business processes and organizational relationships [21]. Also patterns have been proposed in the requirements field to document user needs and specify generic system behaviour at a high level of abstraction [22][23], as well as to record best requirements engineering practices [24]. In all these approaches, the pattern concept is used to represent information to be reused at different development stages.

In this paper, we propose the notion of *elicitation patterns* that capitalize upon the know-how in requirements elicitation so that fundamental elements intervening recurrently throughout requirements elicitation can be specified and reused in different projects by requirements engineers. Our aim is to propose artefacts (patterns) enabling the reuse of usability knowledge to support software developers during usability requirements elicitation so that they can use these patterns to extract all the information they need to completely and unambiguously specify a usability feature.

Note that the solution we are proposing is different from the HCI or usability patterns provided by the HCI community ([25][26][13][27][28][29]). HCI patterns deal with interface recommendations about the visible part of usability features. For example, Welie [13] talks about different ways to present system status data – obtrusively or unobtrusively; and Tidwell [27] suggests the use of animated indicators to display the progress of tasks with particular characteristics. The information provided by these patterns cannot be used as requirements (because they mainly focus on interface issues), although they can be used to generate issues to be discussed with the user in order to fully specify the usability feature in question; for example, the criticality of the different tasks or situations about which status feedback is to be provided.

We will use HCI patterns, as well as the HCI literature, as sources of knowledge to be reused in our elicitation patterns. HCI sources need to be carefully examined to extract useful information from a SE perspective to be used by software developers to discuss and then properly specify usability features with the user and other stakeholders.

4.2. Functional Usability Features

Table 1 shows the usability features that we have selected to develop usability elicitation patterns. We have chosen features that have a significant impact on usability according to the usability literature (column 1), a significant impact on system functionality (column 2), and about which there is enough HCI information (column 3) to derive the essentials to be elicited and specified.

According to HCI experts, each usability feature in Table 1 includes a variety of subtypes. Therefore, we have developed a usability elicitation pattern for each subtype, which we have then termed a usability mechanism (to distinguish from a usability feature).

Table 1. Usability features addressed by usability elicitation patterns

Usability feature	HCI authors who claim that the feature is relevant for software usability	High impact on software functionality	HCI authors who provide information about this feature
Feedback	Nielsen [9], Constantine [1], Shneiderman [2], Hix [7]	Bass et al. [11], Juristo et al.[12]	Tidwell [27][13], Welie [26], Laasko [29], Brighton [25], Coram [28]
Undo/Cancel	Nielsen [9], Hix [7], Shneiderman [2]	Bass et al. [11], Juristo et al.[12]	Brighton [25], Tidwell [13], Welie [26], Laasko [29]
Form/field validation	Shneiderman [2], Hix [7], Constantine [1]	Bass et al. [11], Juristo et al.[12]	Brighton [25], Tidwell [27][13]
Wizard	Constantine [1]	Juristo et al.[12]	Welie [26]
User profile	Hix [7]	Bass et al. [11], Juristo et al.[12]	Welie [26], Tidwell [13]
Help	Nielsen [9]	Bass et al. [11], Juristo et al.[12]	Tidwell [27]

Table 2 shows the specific usability mechanism for which we have developed a usability elicitation pattern. However, as the usability and HCI fields are continually evolving, we plan to add other mechanisms to the list as new information appears.

4.3. An Example of a Usability Elicitation Pattern

Table 3 provides one example of a usability elicitation pattern for the System Status Feedback. The complete list of usability elicitation patterns is available at http://www.ls.fi.upm.es/udis/usability-elicitation-patterns) We describe a usability elicitation pattern by means of the *identification* of the usability feature addressed by the usability elicitation pattern, the *problem* tackled, the *context* in which this pattern will be useful, and the *solution* to the problem. Let us look at these fields in more detail.

The first information that appears in the pattern is its *identification*. This includes information like the *name* of the usability mechanism under consideration, the *family* of usability features to which it belongs (that is, the usability feature of which this mechanism is a subtype) and possible *aliases* by which this usability mechanism may be known.

The *problem* addressed by each pattern is how to elicit and specify the information needed to incorporate in a software system the corresponding usability mechanism.

The *usability context* provides information related to the situation that makes this mechanism useful for the application to be built, and therefore makes the usability elicitation pattern useful for a requirements engineer. This information can act as a general usability requirement statement so a software practitioner knows whether or not this feature is applicable to the software under development.

The *solution* part of the pattern that we propose is composed of two elements: the usability mechanism elicitation guide, and the usability mechanism specification guide.

Table 2. Specific usability mechanisms for which a usability elicitation pattern has been developed

Usability Feature	Usability Mechanism	Goal
Feedback	System Status [13] [28]	To inform users about the internal status of the system
	Interaction [25] [28]	To inform users that the system has registered a user interaction, that is, that the system has heard users
	Warning [26] [25]	To inform users of any action with important consequences
	Long Action Feedback [27][13] [26] [28] [25]	To inform users that the system is processing an action that will take some time to complete
Undo/Cancel	Global Undo [27][13] [26] [29] [25]	To undo system actions at several levels
	Object-Specific Undo [29]	To undo several actions on an object
	Abort Operation [25] [13]	To cancel the execution of a command or an application
	Go Back to a Safe State [13]	To go back to a particular state in a command execution sequence
Form/Field Validation	Structured Text Entry [13] [25]	To help prevent the user from making data input errors
Wizard	Step-by-Step Execution [27][13] [26]	To help do tasks that require different steps with user input
User Profile	Preferences [13] [26]	To record each user's options for working with the system at the functional level
	Personal Object Space [13]	To record each user's options for working with the system at the interface level.
	Favourites [13] [26]	To record certain places of interest for the user
Help	Multilevel Help [26]	To provide different help levels for different users

The *usability mechanism elicitation guide* provides knowledge for eliciting and gathering information to fully specify the usability mechanism. It lists issues that stakeholders (users, developers, HCI experts if available, etc.) should discuss to properly define how the usability mechanism is to be considered in a particular software system. For example, if a user wants to be notified when a change in system status occurs [13], one of the first issues to be discussed for each particular application is for what kind of status changes this notification requires. The example in Table 3 lists some specific questions (about system failures, internal resources, external resources) to be addressed when discussing this issue, along with other questions for dealing with related issues.

The *usability mechanism specification guide* provides a template to be instantiated for each application. In the particular case of Table 3, this template calls for the instantiation of the System Status Feedback for the application under development. So, for each application, status X, XI and XII will be a particular status of the software system developed, whereas faults I, II and III will be particular faults that can occur while the system is executing tasks A, B, and C, respectively, etc.

Table 3. System Status Feedback Requirements Information

IDENTIFICATION

Name:	System Status Feedback
Family:	Feedback
Alias:	Status Display [13] Modelling Feedback Area [28]

PROBLEM

Which information needs to be elicited and specified in order to provide users with system status information.

CONTEXT

When changes that are important to the user occur or
When failures that are important to the user occur, for example:

- During task execution
- Because there are not enough system resources
- Because external resources are not working properly.

Examples of status feedback can be found in status bars on windows applications; train, bus or airline schedule systems; VCR displays; etc.

SOLUTION

Usability Mechanism Elicitation Guide:

1. HCI experts argue that the user wants to be notified when a change of status occurs [13]

   So, the issues to be discussed with stakehorlders include:

   - *Will the system have the **capability to report** system status?*
   - *If so, changes in system status can be triggered by user-requested or other actions or when there is a problem with an external or system resource. So, which kind of changes will the system need to manage?*
     - *What **system statuses** are there and about which does the user need to be informed?*
     - *Do stakeholders want the system to provide notification of system **failures**? If so, which ones?*
     - *Do stakeholders want the system to provide notification if there are not **enough resources** to execute the ongoing commands? If so, which resources?*
     - *Do stakeholders want the system to provide notification if there is a problem with an **external resource** or device with which the system interacts? If so, which ones?*

2. Well-designed displays of information to be shown should be chosen. They need to be unobtrusive if the information is not critically important, but obtrusive if something critical happens. Displays should be arranged to emphasize the important things, de-emphasize the trivial, not hide or obscure anything, and prevent one piece of information from being confused with another. They should never be re-arranged, unless users do so themselves. Attention should be drawn to important information with bright colours, blinking or motion, sound or all three – but a technique appropriate to the actual importance of the situation to the user should be used [13].

   So, for each situation identified above under item 1, discuss with stakeholders:

   - *Which information will be shown to the user?*
   - *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 salient information is closed.*
   - *Which of this information will have to be highlighted because it is related to an important but non-critical situation? Using different colours and sound or motion, sizes, etc.*
   - *Which of this information will be simply displayed in the status area? Locating some kind of indicator in the system status area.*

   For each piece of system status information to be displayed according to its importance, the range will be from obtrusive indicators (for example, a window in the main display area which prevents the user from continuing until it has been closed), through highlighting (with different colours, sounds, motions or sizes) to the least eye-catching 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) will be provided does need to be discussed at this stage.

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 [28]]. 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 [1]. 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 [13].

   So, the issues to be discussed with the user include:

   - *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.*
   - *Which is the best place to locate the feedback information for each situation?*

Usability Mechanism Specification Guide:

The following information will need to be instantiated in the requirements document.

- The system statuses that 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.

5. Proof of Concept

We have worked with the usability elicitation patterns in different contexts. A validation has been performed at the UPM with final-year undergraduate students. Additionally, case studies have also been carried out with software companies.

To test whether software developers could produce usable software using HCI-type information as requirements, we ran a survey at the UPM with final-year undergraduate students. The students had taken three, 90-hour, SE-related subjects as part of their computing degree courses. In addition, most students were working part time at software companies and could, therefore, be considered as junior developers. We worked with pairs of students. Each pair of students developed the same system from a software requirement specification (SRS) document using the IEEE-830 format [30]. We asked students to include a particular usability requirement (like feedback or undo) and to generate the corresponding software. We gave one of the students of the pair the respective usability elicitation patterns and the other student only the definitions of the usability feature according to the usability heuristics found in the literature and encourage him/her to complete this description with information to be found in internet or in other sources. For example, for the feedback feature, we gave one student the four usability elicitation patterns of the family listed in Table 2, and we gave the other student the description of this feature according to usability literature, namely, "the system should always keep users informed about what is going on through appropriate feedback within reasonable time" [9].

Students modified the original SRS to include this new requirement. They examined how each function to be performed by the system would be affected by each feature. They recorded the different tasks they performed during the development process (reading and understanding the original SRS, extracting information about feedback, working on the analysis models, etc.), as well as the time taken for each one.

Comparing the two SRSs generated by the student pairs, we found important differences in completeness and ambiguity. As expected, students who did not use the pattern generated incomplete SRSs, because they had little usability knowledge and could not extract this knowledge from users. The students did not consider all variants of the usability features to be incorporated. Also the usability features that they did identify were not fully specified. This means that many usability issues arose at late development stages (mainly coding and evaluation) and significant rework was required. On the other hand, SRSs developed using the patterns had complete and relatively unambiguous descriptions of the different tasks related to the usability features to be considered, and the subsequent development tasks were more straightforward.

We also found that usability elicitation patterns helped students avoid misconceptions related to the usability features. For example, some students who did not use the patterns confused feedback messages with cancellation or error messages.

Regarding the case studies at software companies, we applied the usability elicitation patterns in the context of a European research project (STATUS project, IST–2001–32298). In particular, we gave these patterns to the industrial partners participating in the project for them to use to develop pilot applications.

The industrial partners found the patterns useful. They included much of the usability knowledge covered by the patterns, most of which was unfamiliar to practitioners and which they had, therefore, not used in earlier development projects. They also mentioned that the patterns provided support for the elicitation process by guiding them through the discussion with the users about particular usability features. Usability tests were conducted by the industrial partners to get an indicator of user satisfaction and found out whether the developed software improved the average level of usability of the applications they develop. One of the companies, LogicDIS (Greece) compared the results of the tests of existing applications developed without the patterns with the results of the tests of the new versions of these applications developed with the patterns [30]. An improvement in usability of almost 25% was found in the new applications. Although preliminary because the number of applications was quite low (3 case studies), these results are indicative of a positive tendency and of the potential benefits the use of usability elicitation patterns can provide.

6. Conclusions

We have discussed some usability issues that stand in the way of creating usable software systems. Specifically, we have established that:

1. Late incorporations of particular usability issues into a software system may involve a lot of rework. Therefore, these features should be incorporated at the requirements stage.
2. Usability has implications for the functional requirements, and the classical

approach of dealing with usability as non-functional requirements falls short of the mark.

3. Usability features are more difficult to specify clearly than may appear at first glance, as a lot of details may need to be explicitly discussed with stakeholders, and typically neither software practitioners nor users have the HCI expertise to know these details.

We have discussed this problem as a vehicle for better understanding what implications usability has for software development. Having placed usability into the right perspective within software development, it is evidently necessary to provide developers with support to satisfactorily deal with usability features during the requirements process.

To this end, we have proposed usability elicitation patterns that provide software practitioners with knowledge that guides them through the process of eliciting and specifying particular usability features. In other words, the use of such patterns helps developers to determine whether and how a usability feature applies to a particular system. Several tests have been run on this issue. The preliminary results are encouraging. However, we are in the process of performing more empirical studies that should contribute to confirming our hypothesis that these elicitation patterns can be particularly useful for organizations with no HCI experts on their software development team, as these patterns have been developed to cover much of the existing public HCI expertise.

We have worked on a pattern-based solution because the information provided in the different usability elicitation patterns can be reused to guide the usability elicitation process for a particular feature across different projects. Nevertheless as is the general rule with pattern use, each individual application is likely to have its particularities, to which the pattern-based solution will need to be adapted. We have not yet had the opportunity to demonstrate whether or not pattern-based elicitation guidelines will work as well with other traditional types of non-functional requirements.

References

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

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

[3] G. M. Donahue. “Usability and the Bottom Line.” *IEEE Software*, vol. 18 (11) 2001, p. 22-30.

[4] J. Nielsen. “Return on Investment for Usability”. Alertbox, January 2003. Http://www.useit.com

[5] M. Chrusch. “Seven Great Myths of Usability. Interactions.” *Journal Name?* September/October 2000, pg. 13-16.

[6] J. Whiteside, J. Bennett, and K. Holtzblatt. "Usability Engineering: Our Experience and Evolution", in *Handbook of Human-Computer Interaction* (editor M. Helander). pp. 791-817. North-Holland, 1988

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

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

[9] J. Nielsen. *Usability Engineering. AP Professional*, 1993., John Wiley & Sons, New York, NY.

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

[11] L Bass, B. John, J Kates. *Achieving Usability Through Software Architecture*. Technical Report. CMU/SEI-2001-TR-005, March 2001.

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

[13] J. Tidwell. The Case for HCI Design Patterns. Visited February 2004 Http://www.mit.edu/jdidwell/common_ground_onefile.htm

[14] B. Kovitz. “Ambiguity and What to Do about It.” *IEEE Joint International Conference on Requirements Engineering (RE’02)* pp. 213 (Key talk).

[15] D.Berry. “The importance of Ignorance in Requirements Engineering.” *Journal of Systems and Software*, vol 28 (2), pp. 179-184.

[16] R. Kazman, J. Gunaratne, B. Jerome. "Why Can't Software Engineers and HCI Practitioners Work Together?", *Human-Computer Interaction Theory and Practice* - Part 1 (Proceedings of HCI International '03), (Crete, Greece), June 2003.

[17] A. Sheffah, E. Metzker. “The Obstacles and Myths of Usability and Software Engineering.” *Communications of the ACM*. December 2004, vol 47 (12), pp. 71-76.

[18] C. Alexander. *A Pattern Language: Towns, Building, Construction*. Oxford University Press, 1977.

[19] E. Gamma. R. Helm, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[20] M. Shaw. “Some Patterns for Software Architectures.” in *Pattern Languages of Program Design 2*. J. Vlissides, J. Colien and N. Kerth Eds. Pp. 255-269, 1996.

[21] H.E. Eriksson, M. Penker. *Business Modeling with UML . Business Patterns at Work*. John Wiley and Sons, 2000.

[22] S. Robertson. Requirements Patterns Via Events/Use Cases. The Atlantic Systems Guild. http://www.systemsguild.com/GuildSite/SQR/Requirements_Patterns.html

[23] S. Konrad, B. Cheng. “Requirements Patterns for Embedded Systems.” *IEEE International Conference on Requirements Engineering*. 2002.

[24] L. Hagge. K. Lappe. "Sharing Requirements Engineering Experience Using Patterns.” *IEEE Software*. Jan-Feb 2005. Pg 24-31.

[25] *Usability Pattern Collection*. Visited January 2004 Http://www.cmis.brighton.ac.uk.research/patterns

[26] M. van Welie. *The Amsterdam Collection of Patterns in User Interface Design*. Visited January 2004. Http://www.welie.com .

[27] J. Tidwell. UI Patterns and Techniques. Visited February 2004. Http://time-tripper.com/uipatterns

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

[29] S. A. Laasko. User Interface Designing Patterns, 2003. http://www.cs.helsinki.fi/u/salaakso/patterns/index_tree.html Visited October 2004.

[30] IEEE, *Recommended Practice for Software Requirements Specifications*, Standard 830, 19xx.

[31] N. Juristo, A. Moreno, D. Tsirikos *Analysis and Comparison of Usability of Old and New Developments* Deliverable.6.3. STATUS project. Http://www.ls.fi.upm.es/status December 2004.

[32] ISO. ISO 13407.Human-Centred Design Processes for Interactive Systems. ISO, 1999.1

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