Software architecture analysis of usability

Folmer, Eelke

Chapter 3

Usability Patterns in SA

Published as: Usability Patterns in Software Architecture, Eelke Folmer, Jan Bosch, Proceedings of the Human Computer Interaction International 2003, Pages 93-97, June 2003.

Abstract: Over the years the software engineering community has increasingly realized the important role software architecture plays in fulfilling the quality requirements of a system. Practice shows that for current software systems, most usability issues are still only detected during testing and deployment. To improve the usability of a software system, usability patterns can be applied. However, too often software systems prove to be inflexible towards such modifications which lead to potentially prohibitively high costs for implementing them afterwards. The reason for this shortcoming is that the software architecture of a system restricts certain usability patterns from being implemented after implementation. Several of these usability patterns are "architecture sensitive", such modifications are costly to implement due through their structural impact on the system. Our research has identified several usability patterns that require architectural support. We argue the importance of the relation between usability and software architecture. Software engineers and usability engineers should be aware of the importance of this relation. The framework which illustrates this relation can be used as a source to inform architecture design for usability.

3.1 Introduction

In the last decades it has become clear that the most challenging task of software development is not just to provide the required functionality, but rather to fulfill specific properties of software such as performance, security or maintainability, which contribute to the quality of software (Bosch, 2000). Usability is an essential part of software quality; issues such as whether a product is easy to learn to use, whether it is responsive to the user and whether the user can efficiently complete tasks using it may greatly affect a product's acceptance and success in the marketplace. Modern software systems are continually increasing in size and complexity. An explicit defined architecture can be used as a tool to manage this size and complexity. The quality attributes of a software system however, are to a large extent determined by a system's software architecture. Quality attributes such as performance or maintainability require explicit attention during development in order to achieve the required levels (Bosch and Bengtsson, 2002). It is our conjecture that this statement also holds for usability. Some changes that affect usability, for example changes to the appearance of a system's user interface, may easily be made late in the development process without incurring great costs. These are changes that are localized to a small section of the source code. Changes that relate to the interactions that take place between the system and the user are likely to require a much greater degree of modification. Restructuring the system at a late stage will be extremely and possibly prohibitively, expensive. To improve on this situation, it would be beneficial for knowledge pertaining to usability to be captured in a form that can be used to inform architectural design, so that engineering for usability is possible early in the design process. The usability engineering community has collected and developed various design solutions such as usability patterns that can be applied to a system to improve usability. Where these

prescribe sequences or styles of interaction between the system and the user, they are likely to have architectural implications. For example, consider the case where the software allows a user to perform a particularly complex task, where a lot of users make mistakes. To address this usability issue a wizard pattern can be employed. This pattern guides the users through the complex task by decomposing the task into a set of manageable steps. However implementing such a pattern as the result of a design decision made late on proves to be very costly. There needs to be provision in the architecture for a wizard component, which can be connected to other relevant components, the one triggering the operation and the one receiving the data gathered by the wizard. The problem with this late detection of usability issues is that sometimes it is very difficult to apply certain usability patterns after the majority of a system has been implemented because these patterns are 'architecture sensitive'. The contribution of this paper is to make software engineers aware that certain 'design solutions' that may improve usability are extremely difficult to retro-fit into applications because these patterns require architectural support. Therefore being able to design architectures with support for usability is very important. The framework that we present in the next section can be used as an informative source during design.

3.2 Usability Framework

Through participation in the STATUS project we have investigated the relationship between usability and software architecture. Before the relationship between usability and software architecture was investigated an accurate definition of usability was tried to obtain by surveying existing literature and practice. Initially a survey was undertaken to try and find a commonly accepted definition for usability in terms of a decomposition into usability attributes. It was soon discovered that the term usability attribute is quite ambiguous. People from industry and academia have quite different perceptions of what they consider to be a useful usability attribute. The number of "usability attributes" obtained in this way grew quite large therefore we needed a way to organize and group the different attributes. Next to the need for organizing these different interpretations of usability attributes, a relation between usability and software architecture was tried to discover. The only 'obvious' relation between usability and architecture is that there are some usability patterns that have a positive effect on usability and that are architecture sensitive. However, it was soon discovered that it was extremely difficult to draw a direct relationship between usability attributes and software architecture. An attempt was made to decompose the set of usability attributes into more detailed elements such as: "the number of errors made during a specific task", which is an indication of reliability, or "time to learn a specific task" which is an indication of learnability, but this decomposition still did not lead to a convincing connecting relationship with architecture.

The reason is that traditionally usability requirements have been specified such that these can be verified for an implemented system. However, such requirements are largely useless in a forward engineering process. For example, it could be stated that a goal for the system could be that it should be easy to learn, or that new users should require no more than 30 minutes instruction, however, a requirement at this level does not help guide the design process. Usability requirements need to take a more concrete form expressed in terms of the solution domain to influence architectural design. To address to these problems discussed a framework has been developed. Figure 30 shows the framework developed so far. It shows a collection of attributes, properties and patterns and shows how these are linked to give the relationship between usability and software architecture. This relation is illustrated by means of an example.

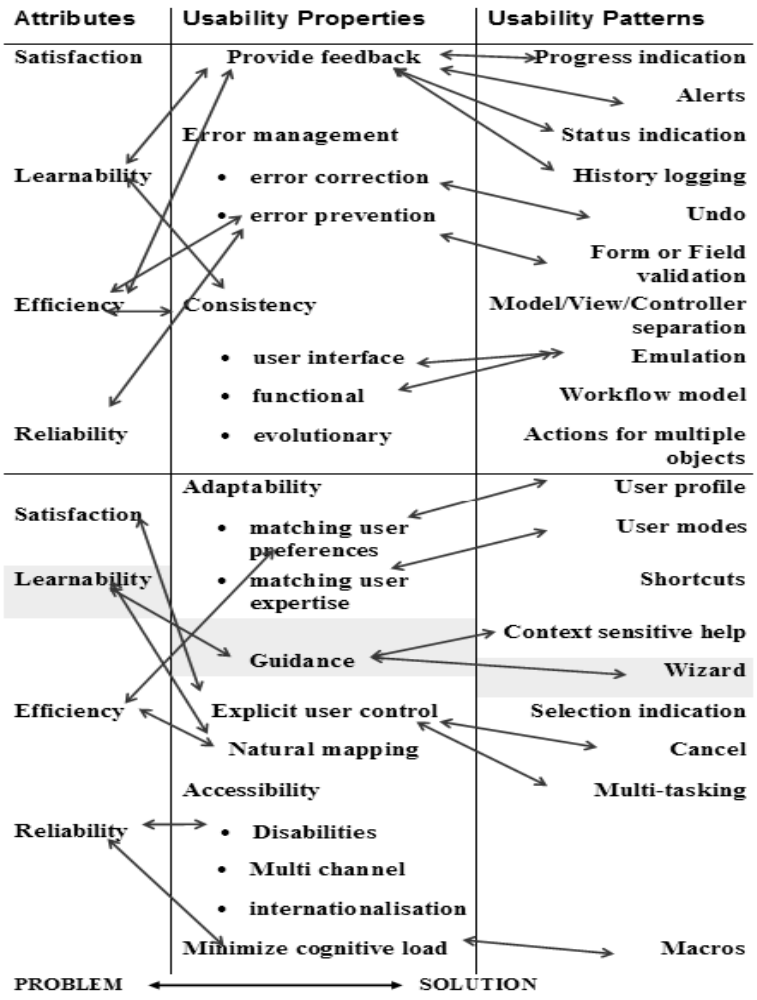

Figure 30: Usability Framework

Figure 30 shows the wizard pattern linked to the "guidance" usability property which in turn is linked to the "learnability" usability attribute. The wizard pattern guides the user through a complex task by decomposing the task into a set of manageable subtasks. To implement a wizard a provision is needed in the architecture for a wizard component, which can be connected to other relevant components: the one triggering the operation and the one receiving the data gathered by the wizard. The wizard is related to usability because it uses the primitive of guidance to "guide" the user through the task. Guidance on its turn has two "obvious" relations with usability. Guidance has a positive effect on learnability and a negative effect on efficiency. The concept of "guidance" is defined as a usability property; a usability property is a more concrete form of a usability requirement specified in terms of the solution domain. Patterns relate to one or more of these usability properties. Properties on their turn relate to one or more usability attributes. This relation is not necessarily a one to one mapping. The relationship can be positive as well as negative. To avoid the table becoming too cluttered, and the risk of possibly producing a fully connected graph, only the links thought to be strongest and positive are indicated in the table. The framework relates the problem to the solution domain; a usability attribute can be measured on a

completed solution, whereas a usability property exists in the problem domain, and can be used as a requirement for system design. A usability pattern bridges the gap between problem and solution domains, providing us with a mechanism to fulfil a requirement, providing us with a solution for which the corresponding usability attribute can be measured. The next sections enumerate the concepts of usability attributes, properties and patterns which comprise our framework.

3.3 Usability Attributes

A comprehensive survey of the literature (Folmer and Bosch, 2004) revealed that different researchers have different definitions for the term usability attribute, but the generally accepted meaning is that a usability attribute is a precise and measurable component of the abstract concept that is usability. After an extensive search of the work of various authors, the following set of usability attributes is identified for which software systems in our work are assessed. No innovation was applied in this area, since abundant research has already focused on finding and defining the optimal set of attributes that compose usability. Therefore, merely the set of attributes most commonly cited amongst authors in the usability field has been taken. The four attributes that are chosen are: Learnability - how quickly and easily users can begin to do productive work with a system that is new to them, combined with the ease of remembering the way a system must be operated. Efficiency of use - the number of tasks per unit time that the user can perform using the system. Reliability in use - this attribute refers to the error rate in using the system and the time it takes to recover from errors. Satisfaction - the subjective opinions that users form in using the system. These attributes can be measured directly by observing and interviewing users of the final system using techniques that are well established in the field of usability engineering.

3.4 Usability Properties

Essentially, our usability properties embody the heuristics and design principles that researchers in the usability field have found to have a direct influence on system usability. These properties can be used as requirements at the design stage, for instance by specifying: "the system must provide feedback". They are not strict requirements in a way that they are requirements that should be fulfilled at all costs. It is up to the software engineer to decide how and at which levels these properties are implemented by using usability patterns of which it is known they have an effect on this usability property. For instance providing feedback when printing in an application can be very usable, however if every possible user action would result in feedback from the system it would be quite annoying and hence not usable. Therefore these properties should be implemented with care. The following properties have been identified: Providing feedback - the system provides continuous feedback as to system operation to the user. Error management - includes error prevention and recovery. Consistency - consistency of both the user interface and functional operation of the system. Guidance - on-line guidance as to the operation of the system. Minimize cognitive load - system design should recognize human cognitive limitations, short-term memory etc. Natural mapping - includes predictability of operation, semiotic significance of symbols and ease of navigation. Accessibility - includes multi-mode access, internationalization and support for disabled users.

3.5 Usability Patterns

One of the products of the research on this project into the relationship between software architecture and usability is the concept of a usability pattern. The term "usability pattern" is chosen to refer to a technique or mechanism that can be applied to the design of the architecture of a software system in order to address a need identified by a usability property at the requirements stage. Various usability pattern collections have been defined (Tidwell 1998, Welie and Trætteberg, 2000). Our collection is different from those because we only consider patterns which should be applied during the design of a system's software architecture, rather than during the detailed design stage. (Bass et al, 2001) have investigated the relationship between the usability and software architecture through the definition of a set of 26 scenarios. These scenarios are in some way equivalent to our properties and usability patterns. However there are some differences. They have used a bottom up approach from the scenarios whereas we have taken a top down approach from the definition of usability. Our approach has in our opinion resulted in a more clearly documented and illustrated relationship between those usability issues addressed by the design principles and the software architecture design decisions required to fulfill usability requirements. Another difference is that our patterns have been obtained from an inductive process from different practical cases (e-commerce software developed by the industrial partners in this project) whereas their scenarios result from personal experience and literature surveys. Their work has been useful to support our statement that usability and software are related through usability patterns. A full catalogue of patterns identified is presented on http://www.designforquality.com.

There is not a one-to-one mapping between usability patterns and the usability properties that they affect. A pattern may be related to any number of properties, and each property may be improved (or impaired) by a number of different patterns. The choice of which pattern to apply may be made on the basis of cost and the trade off between different usability properties or between usability and other quality attributes such as security or performance. This list of patterns presented in Figure 30 is not intended to be exhaustive, and it is envisaged that future work on this project will lead to the expansion and reworking of the set of patterns presented here, including work to fill out the description of each pattern to include more of the sections which traditionally make up a pattern description, for instance what the pros and cons of using each pattern may be.

3.6 Summary and Conclusions

Our research has argued the importance of the relation between usability and software architecture. A framework has been developed which illustrates this relation. The list of usability patterns and properties identified/defined in our framework is substantial but incomplete, new usability patterns or properties that are developed or discovered can be fitted in the existing framework. Future research should focus on verifying the architectural sensitiveness of the usability patterns that have been identified. For validation only e-commerce software provided by our industrial partners in this project has been considered. To achieve more accurate results our view should be expanded to other application domains. The usability properties can be used as requirements for design, it is up to the software architect to select patterns related to specific properties that need to be improved for a system. It is not claimed that a particular usability pattern will improve usability for any system because many other factors may be

involved that determine the usability of a system. The relationships in the framework indicate potential relationships. Further work is required to substantiate these relationships and to provide models and assessment procedures for the precise way that the relationships operate. This framework provides the basis for developing techniques for assessing software architectures for their support of usability. This technique allows for iteratively designing for usability on the architectural level.