Experiences with Software Architecture Analysis of Usability

Eelke Folmer, University of Nevada - Reno, USA
Jan Bosch, Intuit Inc. - Mountain View, USA

ABSTRACT

Software engineers and human computer interaction engineers have come to the understanding that usability is not something that can be “added” to a software product during late stage, since to a certain extent it is determined and restricted by architecture design. Cost effectively developing a usable system must include developing an architecture, which supports usability. Because software engineers in industry lacked support for the early evaluation of usability, we defined a generalized four-step method for software architecture level usability analysis called SALUTA. In this article, we report on a number of experiences and problems we observed when performing architecture analysis of usability at three industrial case studies performed in the domain of Web-based enterprise systems. Suggestions or solutions are provided for solving or avoiding these problems so organizations facing similar problems may learn from our experiences.

Keywords: Human/Computer Interaction; Software Architecture; User Requirements; User Satisfaction; Web Applications

INTRODUCTION

A key problem facing software engineers today is meeting quality requirements, such as maintainability, reliability, and performance. Quality is not something that can be “added” to a software product during late stage, since to a certain extent it is determined and restricted by software architecture design (Bosch, 2000), that is, the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment and the principles guiding its design and evolution (IEEE, 1998). Significantly improving the quality of a software system often requires major changes to the software architecture. For example, using layers (Buschmann, Meunier, Rohnert,
Sommerlad, & Stal, 1996) may improve maintainability, but requires a significant amount of restructuring if imposed upon an existing software design and puts restrictions on how components can access each other.

A complicating problem is that most of these qualities can only be measured during deployment. In addition, trying to restructure the architecture during late stage is prohibitively expensive as it affects large parts of the existing source code. The problem boils down to making sure you initial architecture design supports the right amount of quality, since fixing it during the later stages of development is expensive. Software architects have been aware of these constraints and much research effort has been put into the development of architecture analysis methods. A number of methods (see related work section) have been developed, allowing an architect to predict the support for maintainability, performance, or reliability for any given architecture design.

A quality that has previously largely been ignored by software engineers—but which is increasingly recognized as one of the most important qualities—is usability. Providing interaction different from how the user would expect it or requires it is usually detrimental to the commercial success of any software application. Similar to other qualities, usability is restricted by software architecture design. The quintessential example used to illustrate this restriction is adding undo to an application. Undo allows you to reverse actions (such as reversing making a text bold in Word). Undo improves usability as it allows a user to explore, make mistakes, and easily go some steps back; facilitating learning the application’s functionality. Experiences with implementing undo to an existing application show that it is very expensive to do since implementations of undo are usually based upon the command pattern (Gamma, Helm, Johnson, & Vlissides, 1995). Similar to using the layers pattern, this requires some significant restructuring and is expensive to retrofit into an existing software design. Several other usability features such as user profiles, visual consistency, and actions for multiple objects have also proven (Bass, Kates, & John, 2001; Folmer, Gurp, & Bosch, 2003) to be architecture “sensitive.”

Studies (Pressman, 1992; Landauer, 1995) reveal that a significant large part of the maintenance costs of software systems is spent on dealing with usability issues and we have reason to believe that part of these costs are explained by making expensive changes to an architecture design to support particular usability features. In order to be able to cost effectively develop a usable application we need to make sure that our software architecture design can support such usability features. Yet few software engineers and human computer interaction engineers are aware of this constraint and as a result avoidable rework is frequently necessary.

In order to improve upon this situation we first captured the relevant design knowledge. In Folmer et al. (2003) we describe the software-architecture-usability (SAU) framework, which identifies a number of usability features that are hard to retrofit. Next we developed a method providing architects with a number of steps for performing architecture analysis of usability. In Folmer and Bosch (2002) we provide an overview of usability evaluation techniques. Unfortunately, no architecture assessment techniques have been identified which focus on usability. Based upon successful experiences (Lassing, Bengtsson,
The EduOntoWiki Project for Supporting Social, Educational, and Knowledge Construction Processes with Semantic Web Paradigm
www.igi-global.com/chapter/eduontowiki-project-supporting-social-educational/37704?camid=4v1a

Becoming Anonymous: A Politics of Masking
Maria-Carolina Cambre (2018). The Dark Web: Breakthroughs in Research and Practice (pp. 290-317).
www.igi-global.com/chapter/becoming-anonymous/185878?camid=4v1a