Case studies on analyzing software architectures for usability

Conference Paper · January 2005

DOI: 10.1109/EUROMICRO.2005.17 · Source: IEEE Xplore

CITATIONS
11

READS
335

2 authors, including:

Jan Bosch
Chalmers University of Technology
619 PUBLICATIONS **22,257** CITATIONS

SEE PROFILE

Case studies on Analyzing Software Architectures for Usability

Eelke Folmer, Jan Bosch
Department of Mathematics and Computing Science
University of Groningen, the Netherlands

Abstract

Studies of software engineering projects reveal that a large number of usability related change requests are made after its deployment. Fixing certain usability problems during the later stages of development has proven to be costly, since some of these changes require changes to the software architecture i.e. this often requires large parts of code to be completely rewritten. Explicit evaluation of usability during architectural design may reduce the risk of building a system that fails to meet its usability requirements and may prevent high costs incurring adaptive maintenance activities once the system has been implemented. In this paper, we demonstrate the use of a scenario based architecture analysis technique for usability we developed, at two case studies.

1. Introduction

Usability is increasingly recognized as an important consideration during software development; however, many well-known software products suffer from usability problems that cannot be repaired without major changes to the software architecture of these products. Usability, similar to other qualities, is, to a certain extent, restricted and fulfilled by software architecture design. The quintessential example that is always used to illustrate this restriction is adding undo to an application. Undo is the ability to reverse certain actions (such as reversing deleting some text in Word). Undo may significantly improve usability as it allows a user to explore, make mistakes and easily go some steps back; facilitating learning the application's functionality. However from experience, it is also learned that it is often very hard to implement undo in an application because it requires large parts of code to be completely rewritten e.g. undo is hard to retrofit. Because of this high cost certain usability improving solutions are not added during late stage.

The problem is that software architects and HCI engineers are often not aware of the impact of usability improving solutions on the software architecture. Nor are there any techniques that explicitly focus on analyzing software architectures for their support of such solutions. As a result, avoidable rework is frequently necessary. This rework leads to high costs, and because tradeoffs need to be made during design, for example between cost and quality, this leads to systems with less than optimal usability.

To improve upon this situation we developed a Scenario based Architecture Level UsabiliTy Assessment technique called SALUTA [1]. A method provides the structure for understanding and reasoning about how particular design decisions may affect usability. However, to assess a software architecture for its support of usability we need to know which usability improving design solutions are hard to retrofit so that these may be considered during architecture design. In [2] we presented the software-architecture-usability (SAU) framework, which consists of an integrated set of usability improving design solutions that have been identified to be hard to retrofit. SALUTA uses the SAU framework for identifying the architecture's support for usability. In this paper we report upon two case studies performed with our technique.

The remainder of this paper is organized as follows. In the next section, the SAU framework is presented. Section 3 presents an overview of the main steps of SALUTA. Section 4 presents the results of the case studies performed and provides some examples for performing usability analysis in practice and discusses the validation of the method. Related work is discussed in Section 5. Finally the paper is concluded in Section 6.

2. SAU framework

In order to be able to assess an architecture for its support of usability, we need to know which usability improving design solutions are hard to retrofit. Several approaches [3,2] aim at capturing this relevant design knowledge. In [2] we defined a framework that captures this relevant design knowledge. This framework is composed of a set of design solutions such as usability patterns and usability properties that have a positive effect on usability but that have been

Proceedings of the 2005 31st EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA'05)
0-7695-2431-1/05 $20.00 © 2005 IEEE

identified in practice and literature to be difficult to retrofit. The framework consists of the following concepts:

2.1 Architecture sensitive usability patterns

Interaction design (ID) patterns such as undo [4], user profiles [5] or multiple views [6], most commonly provide solutions to specific usability problems in interface and interaction design. Several pattern collections [7,8] are freely accessible on the web providing user interface designers with examples of good practice. Most existing ID pattern collections however, refrain from providing or discussing implementation details. This is not surprising since most ID patterns are written by user interface designers that have little interest in the implementation part and usually lack the knowledge to define the implementation details. This is a problem since without knowing these implementation details it becomes very hard to assess whether such a pattern can be easily added during late stage design. Architecture sensitive usability patterns (ASUP) are an extension of existing ID patterns and we focused on identifying architectural implications for implementing these patterns.

Table 1: Undo ASUP

Problem	Users do actions they later want reverse because they realized they made a mistake or because they changed their mind.
Use when	You are designing a desktop or web-based application where users can manage information or create new artifacts. Undo is not suitable for systems with actions that are not reversible such as emailing.
Solution	Maintain a list of user actions and allow users to reverse selected actions.
Why	Offering the possibility to always undo actions gives users a comforting feeling.
Usability properties affected:	+ **Error management**: providing the ability to undo an action helps the user to correct errors if the user makes a mistake. + **Explicit user control**: allowing the user to undo actions helps the user feel that they are in control of the interaction.
Architectural Considerations	There are two approaches to implementing Undo. The first is to capture the entire state of the system after each user action. The second is to capture only relative changes to the system's state. Since changes are the result of an action, the implementation is based on using Command objects [9] that are then put on a stack. Each specific command is a specialized instance of an abstract class Command. Consequently, the entire user-accessible functionality of the application must be written using Command objects.

Our set of patterns has been identified from various cases in industry, modern software, literature surveys as well as from existing (usability) pattern collections. Table 1 shows part of the undo ASUP.

2.2. Usability properties

In order to be able to "guide" architectural design we identified a set of usability properties that embody the heuristics and design principles that researchers in the usability field consider to have a direct positive influence on usability. Usability properties most likely have architectural implications. Table 2 shows the error management usability property.

Table 2: Error management property

Intent:	The system should provide a way to manage user errors. This can be done in two ways: - By preventing errors from happening, so users make fewer mistakes. - By providing an error correcting mechanism, so that errors made by users can be corrected.
Usability attributes affected:	+ **Reliability**: error management increases reliability because users make fewer mistakes. + **Efficiency**: efficiency is increased because it takes less time to recover from errors or users make fewer errors.

Relationships have been defined between usability properties and ASUP's. For example undo is connected to error management as undo provides the ability to correct errors. The usability properties in the framework can then be used as requirements during design. For example, if a requirements species, "the system must provide feedback", we can use the framework to identify which usability patterns may be implemented to fulfill these properties.

2.3. Usability attributes

In order to identify how an ASUP implemented in the architecture may improve usability, an additional layer in our framework was needed. A number of usability attributes (e.g. a decomposition of usability into more concrete measurable terms) have been selected from literature that appear to form the most common denominator of existing notions of usability:

- **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** the error rate in using the system and the time it takes to recover from errors.
- **Satisfaction** - the subjective opinions of the users of the system.

These attributes have been related to usability properties to be able to identify how an ASUP that has been implemented may improve or impair certain aspects of usability. For example, if undo has been implemented in the architecture, we can analyze the relationships in the SAU framework to identify that undo improves explicit user control and error management. Explicit user control increases satisfaction and error management improves reliability and efficiency. SALUTA uses this framework to analyze an architecture's support for usability. Figure 1 shows some relationships defined. Table 3 shows a list of ASUP's and properties we identified.

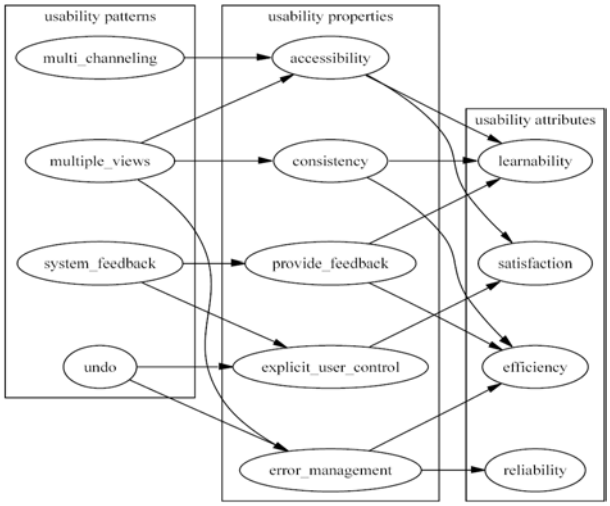

Figure 1: SAU framework

Table 3: ASUP's and properties identified

ASUP	Usability Properties
Workflow modeling	Natural mapping
Scripting	Minimize cognitive load
User Modes	Adaptability
Wizard	Guidance
User profiles	Provide feedback
Context Sensitive Help	Explicit user control
System Feedback	Error management
Actions for multiple objects	Consistency
Cancel	Accessibility
Undo	
History Logging	
Data validation	
Multiple views	
Emulation	
Multi Channeling	

3. SALUTA

SALUTA consists of the following four steps:

1. Create usage profile; describe required usability.
2. Analyze architecture; describe provided usability.
3. Evaluate scenarios; determine support.
4. Interpret the results; take actions.

3.1. Create usage profile

Before a software architecture can be assessed, the required usability of the system needs to be determined. Several specification styles of usability have been identified [10]. One shortcoming of these techniques [11,12,13] is that they are poorly suited for architectural assessment. A format more suitable for architecture assessment are scenario profiles [14]. Scenario profiles describe the semantics of software quality attributes by means of a set of scenarios. Scenarios are much more specific and fine-grained than abstract usability requirements.

For the assessment we defined our own type of scenario profile called usage profile, which consists of usage scenarios. According to the ISO definition [15], usability depends on: the users (who is using the product?), the tasks (what are the users trying to do?), the context of use (where and how is the product used). Usability may also depend on other variables, such as goals of use. In our usage scenario definition only these aforementioned variables are included. However, a usage scenario specified as a valid combination of (user, task, context of use), does not specify anything about the required usability of the system.

In order to do that, the usage scenario is related to the four usability attributes defined in our framework. This is because for some usability attributes, such as efficiency and learnability, tradeoffs have to be made as it is often impossible to design a system that has high scores on all attributes. A purpose of usability requirements is therefore to specify a necessary level for each attribute [10].

For each usage scenario, numeric values are determined for each of the attributes to determine a prioritization between the usability attributes. For example, if for a particular usage scenario learnability is considered to be of more importance than other usability attributes, then the usage scenario must reflect this difference in the priorities for the usability attributes. The analyst interprets the priority values during the analysis phase to determine whether the architecture sufficiently supports particular usability attributes for that usage scenario.

The steps that need to be taken for usage profile creation are the following:

1. Identify the users: Users that are representative for the use of the system should be categorized in types

or groups (for example system administrators, end-users etc).
2. Identify the tasks: Representative tasks are selected that highlight the important features of the system. For example, a task may be "find out where course computer vision is given".
3. Identify the contexts of use: Representative contexts of use are identified. (For example. Helpdesk context or disability context.)
4. Determine attribute values: For each valid combination of user, task and context of use, usability attributes are quantified to express the required usability of the system, based on the usability requirements specification. To reflect the difference in priority, for example numeric values between one and four can be assigned to the attributes for each scenario.
5. Scenario selection & weighing: Evaluating all scenarios may be a costly and time-consuming process. Therefore, the goal of performing an assessment is not to evaluate all scenarios but only a representative subset. Different profiles may be defined depending on the goal of the analysis. For example, if the goal is to compare two architectures, scenarios may be selected that highlight the differences between those architectures. If the goal is to analyze the level of support for usability, scenarios may be selected that are important to the users. To express differences between usage scenarios in the usage profile, properties may be assigned to scenarios, for example: priority or probability of use within a certain time. The result of the assessment may be influenced by weighing scenarios, if some scenarios are more important than others, weighing these scenarios reflect these differences.

A usage profile that is created using these steps is summarized in a table (see Table 5 for an example). Usage profile creation is not intended to replace existing requirements engineering techniques. Rather it transforms (existing) usability requirements into something that can be used for architecture assessment. Existing techniques such as interviews, group discussions or observations [11,13,16] already provide information such as representative tasks, users and contexts of use that are needed to create a usage profile. Close cooperation between the analyst and the person responsible for the usability requirements is required. The usability engineer may fill in the missing information on the usability requirements.

3.2. Describe provided usability

SA of usability requires architectural information that allows the analyst to determine the support for the usage scenarios. This process focuses on identifying architectural elements that may support the usage scenario. Two types of analysis techniques are defined:

- Usability pattern based analysis: using the list of architectural sensitive usability patterns defined in our framework the architecture's support for usability is determined by the presence of these patterns in the architecture design.
- Usability property based analysis: The software architecture can be seen as the result of a series of design decisions [17]. Reconstructing this process and assessing the effect of such individual decisions with regard to usability attributes may provide additional information about the intended quality of the system. Using the list of usability properties defined in our framework, the architecture and the design decisions that lead to this architecture are analyzed for these properties.

The quality of the assessment very much depends on the amount of evidence for patterns and properties support that can be extracted from the architecture. Some usability properties such as error management are fulfilled by ASUP's such as undo, cancel or data validation. In addition to patterns there may be additional evidence in the form of other design decisions that were motivated by usability properties. The software architecture of a system has several aspects (such as design decisions and their rationale) that cannot easily be captured or expressed in a single model. Different views on the system [18] may be needed access such information. Initially the analysis is based on the information that is available, such as diagrams etc. However due to the non explicit nature of architecture design the analysis strongly depends on having access to both design documentation and software architects. The architect may fill in the missing information on the architecture. SALUTA does not address the problem of properly documenting software architectures and design decisions. The more effort is put into documenting the software architecture the more accurate the assessment is.

3.3. Evaluate scenarios

In this step we evaluate the support for each of the scenarios in the usage profile. For each scenario, we analyze by which usability patterns and properties, that have been identified in the previous step, it is affected. To determine the support we use the relations defined in the SAU framework to analyze how a particular pattern or property may improve or impair a specific usability attribute. This step is repeated for each pattern or property that affects the scenario. The analyst then determines the support of the usage scenario based on the acquired information.

For each scenario, the results of the support analysis are expressed qualitatively using quantitative measures. For example the support may be expressed on a five level scale (++, +, +/-,-,--). The outcome of the overall analysis may be a simple binary answer (supported/unsupported) or a more elaborate answer (70% supported) depending on how much information is available and how much effort is being put in creating the usage profile.

3.4. Interpret the results

After scenario evaluation, the results need to be interpreted to draw conclusions concerning the software architecture. This interpretation depends on the goal of the analysis i.e. based on the goal of the analysis; a certain usage profile is selected.

If the goal of the analysis is to compare two or more candidate software architectures, the support for a particular usage scenario must be expressed on an ordinal scale to indicate a relation between the different candidates. (Which one has the better support?).

If the goal is to iteratively design an architecture, then if the architecture provides sufficient support, the design process may be ended. Otherwise, architectural transformations need to be applied to improve usability. Qualitative information such as which scenarios are poorly supported and which usability properties or patterns have not been considered may guide the architect in applying particular transformations. The framework may then be used as an informative source for design and improvement of the architecture's support of usability.

4. Case studies

In order to validate SALUTA it has been applied in three case studies. The first case study (a web based CMS system) has been published in [1] and in this paper we report upon two other case studies performed at our industrial partners in the STATUS[1] project that sponsored this research. All case studies have been performed in the domain of web based enterprise systems; an increasingly popular application format. From a usability point of view this is a very interesting domain: as anyone with an internet connection may be a potential user and a lot of different types of users and usages must be supported.

The Compressor catalogue application is a product developed by the Imperial Highway Group (IHG) for a client in the refrigeration industry. It is an e-commerce application, which makes it possible for potential customers to search for detailed technical information about a range of compressors; for example, comparing two compressors. There was an existing implementation as a Visual Basic application, but the application has been redeveloped in the form of a web application. The system employs a 3-tiered architecture and is built upon an in-house developed application framework. The application is being designed to be able to work with several different web servers or without any. The independence of the database is developed through Java Database Connectivity (JDBC). The data sources (either input or output) can also be XML files. The application server has a modular structure, it is composed by a messaging system and the rest of the system is based on several connectable modules (services) that communicate between them.

The eSuite product developed by LogicDIS is a system that allows access to various ERP (Enterprise Resource Planning) systems, through a web interface. ERP systems generally run on large mainframe computers and only provide users with a terminal interface. eSuite is built as an web interface on top of different ERP systems. Users can access the system from a desktop computer but also from a mobile phone. The system employs a tiered architecture commonly found in web applications. The user interfaces with the system through a web browser. A web server runs a Java servlet and some business logic components, which communicate with the ERP.

Table 4: Comparison of both cases

Aspect	Compressor	eSuite
Type	E-commerce	ERP
No. of users	> 100	> 1000
Goal of the analysis	Selection: Compare old versus new version of Compressor.	Design: iteratively design & improve an architecture.
types of users	3	2
Type of interaction	Information browsing; Comparing and analyzing data of different types of compressors and compressor parts.	Typical ERP functionality. (e.g. insert order, get client balance sheet)
Usage contexts	Mobile/Desktop/Standalone	Mobile/Desktop

[1] STATUS is an ESPRIT project (IST-2001-32298) financed by the European Commission in its Information Society Technologies Program. The partners are Information Highway Group (IHG), Universidad Politecnica de Madrid (UPM), University of Groningen (RUG), Imperial College of Science, Technology and Medicine (ICSTM), LOGICDIS S.A.

As an input to both case studies, we interviewed the software architects and several other individuals involved in the development of these systems. We analyzed the results from usability tests with the old systems and with interface prototypes of the new systems and examined the available design documentation such as architectural designs and usability requirements specifications. Table 4 provides a comparison between both cases.

In the next subsections we present the results of applying the steps of SALUTA at each case.

4.1. Usage profile creation

First we created usage profiles for both cases to express the required usability of these systems. We first identified relevant users, tasks and contexts of use. In both cases this information could be retrieved from analyzing requirements specification. In the Compressor case the analysis of users, tasks and context of use resulted in 58 unique (user, task, usage context) combinations. In order to keep the assessment at a reasonable size the number of scenarios was reduced. Some tasks that were quite similar were combined and certain tasks with low execution frequencies were left out. We tried to minimize the number of scenarios without losing any representativeness of the required usability of the system. Eventually 14 scenarios were distilled.

In the next step we determined attribute priority values for the scenarios. In the requirements specification some usability requirements were explicitly specified for certain user groups or for certain tasks and contexts of use. For example in the eSuite case it was specified that learnability was important for simple users. Analyzing these requirements provided us with an indication which usability attributes were important for which scenario, however translating these sometimes abstract or weakly specified usability requirements to concrete attribute values was difficult. In addition some (obvious) usability requirements were not specified at all. As a result a close cooperation with those responsible for specifying the usability requirements was necessary to be able to make this translation.

For scenario 4 in Table 5 we give the argumentation that led to assigning specific values to the attributes. Learnability is not important for administrators since they are quite familiar with the application. Inserting an order is error prone therefore reliability is important. Manipulating data and interaction on a mobile phone can be very difficult and time consuming if the system is slow and unreliable; consequently, high values have been given to reliability and efficiency and low values to the other attributes for this specific scenario. Table 5 lists the usage profile we created for eSuite.

Table 5: eSuite usage profile

#	User	Context	Task	e	l	r	s
1	Simple users	Desktop	Insert Order	1	4	3	2
2	Simple users	Mobile	Insert Order	4	3	2	1
3	Administrators	Desktop	Insert Order	4	1	3	2
4	Administrators	Mobile	Insert Order	4	1	3	2
5	Simple users	Desktop	Search Order	1	3	4	2
6	Administrators	Desktop	Search Order	3	1	4	2
7	Simple users	Desktop	Get balance	1	4	3	2
8	Administrators	Desktop	Get balance	3	1	4	2
9	Simple users	Desktop	View Stock	1	4	2	3
1	Administrators	Desktop	View Stock	4	1	2	3
1	Simple users	Desktop	Get Statistics	1	4	2	3
1	Administrators	Desktop	Get Statistics	4	1	3	2

4.2. Architecture description

Table 6: ASUP / Properties present in SA

		Implementation
History Logging		All user actions go through basic servlet,; these can be logged.
Multi Channeling		Through the use of XML/ XSLT different views / controllers can be defined for different devices such as Mobile/ WAP.
Undo		Not implemented / Not relevant
Adaptability	Matching user preferences	Stylesheets / customization.
	Matching user expertise	Using xml and a user profile database user roles could be defined.
Guidance		No specific design decisions.
Accessibility	Disabilities	No specific design decisions.
	Multi access	Multi channeling is provided by the web server which can provide a front end to for example WAP based devices.
	Internationalization	Greek/English support (language in xml files) / java support Unicode /locale, Xml placeholder for languages.

In the next step we analyzed the software architecture designs and documentation for the presence of ASUP's and usability properties. We did this by interviewing the architect using our SAU framework. This provided us with a list *if* particular patterns and properties had been implemented. We then got into more detail by analyzing available architecture designs and documentation for evidence of *how* these patterns and properties had been implemented. Table 6 show ASUP's and properties we identified in the new Compressor architecture.

4.3. Evaluate scenarios

Using the information acquired in the previous step we determined the architecture's support for the usage profiles. By analyzing for each pattern and property, the effect on usability, the support for this scenario was determined. Table 7 shows some of the results of the analysis.

Table 7: example usage evaluation

	Scenario #	# patterns	# properties	support
Compressor	1 (old)	0/15	0/15	--
	2 (old)	1/15	1/15	-/+
	1 (new)	6/15	5/10	-/+
	2 (new)	10/15	5/10	+
eSuite	3	8/15	7/10	+/-
	5	8/15	7/10	+
	6	8/15	7/10	++

To illustrate this step we provide an example analysis for the support of scenario #3 of the eSuite case study. The scenario requires high values for efficiency (4) and reliability (3) (See Table 5). Several patterns and properties positively contribute to the support of this scenario (see Table 7). However this number does only provide an indication. We have to look at each pattern and property to determine the support; for example data validation positively contributes to reliability. But apart from data validation no other ASUPs support reliability or efficiency. This task could benefit from adding specific ASUPS such as undo which will improve reliability as users can correct their mistakes or guidance which will also improve reliability. In the context of this specific system we can conclude that the architecture has only medium support for this scenario.

4.4. Interpret the results

Table 8 lists the results of the assessments of both cases. The table lists the number of scenarios analyzed and lists whether these scenarios are strongly rejected (--), weakly rejected (-), medium accepted (-/+), weakly accepted (+) or strongly accepted (++).

Table 8: evaluation results assessments

Nr of scenarios ↓		--	-	-/+	+	++
Old Compressor	14	2	2	8	-	-
New Compressor	14	-	-	5	6	3
eSuite	12	-	-	3	4	3

In the Compressor case we identified that the new architecture has significantly improved over the old architecture, however there was still room for improvement. Unfortunately we were not involved during architecture design. However, the assessment did provide the architect with valuable information on which usability improving design solutions could still be supported by the architecture without incurring great costs.

In the eSuite case we were involved during architecture design. Based on the assessment results the eSuite architecture was improved[2] by applying certain patterns (actions for multiple objects, undo, and contexts sensitive help) from the SAU framework.

Our general impression was that the assessments were well received by the participating architects. The assessments emphasized and increased the understanding of the important relationship between software architecture and usability. The results of the assessments were documented and taken into account for future releases and redevelopment of the products.

4.5. Validation

The case studies show that it is possible to use SALUTA to assess software architectures for their support of usability, but whether we accurately predicted the architecture's support for usability is only answered when comparing the results of the analysis with the results of usability tests when the system has been finished. Several user tests have been performed. The results of these tests fit the results of our analysis. Some usability issues came up that where not predicted during our architectural assessment. However, these do not appear to be caused by problems in the software architecture. Validating SALUTA is difficult as any improvement in the usability of the final system should not be solely accounted to our method. More focus on usability during development in general is in our opinion the main cause for an increase in usability. To validate SALUTA we should measure the decrease in costs spent on usability during maintenance. However in the organizations that participated in the case studies such figures have not been recorded nor is any historical data available.

Future case studies shall focus on designing the architecture based on the usage profile e.g. an attribute/property-based architectural design, where the SAU framework is used to suggest patterns that should be used rather than identify their absence post-hoc.

5. Related work

Several scenario based architecture assessments techniques have been developed e.g. SAAM [19],

[2] The decision to apply certain patterns was not solely based on the result of the assessment but also as result of user tests with prototypes were these patterns were present.

ATAM [20] and QASAR[21]. These techniques provide a generic method and steps for the assessment for different quality attributes. Certain specific quality-attribute assessment techniques have been developed. SAAMER [22] is an extension to SAAM and addresses quality attributes such as maintainability, modifiability and reusability. In [23], ALMA is proposed which can be used to assess modifiability.

6. Conclusions

Fixing certain usability problems during the later stages of development has proven to be costly, since some of these changes require changes to the software architecture, which is expensive to modify during late stage. The goal of an architecture analysis method is to understand and reason about the effect of design decisions on the quality of the final system, at a time when it is still cheap to change these decisions.

In this paper, we have presented the results of applying SALUTA, a scenario based assessment technique for usability, at two case studies. SALUTA consists of four major steps: First, the required usability of the system is expressed by means of a usage profile. The following sub-steps are taken for creating a usage profile: identify the users, identify the tasks, identify the contexts of use, determine attribute values, scenario selection & weighing. In the second step, the information about the software architecture is collected using the SAU framework which consists of an integrated set of design solutions that have a positive effect on usability but are difficult to retrofit into applications because of their architectural impact.

The next step is to evaluate the architecture's support of usage profile using the information extracted in the previous step. The final step is then to interpret these results and to draw conclusions about the software architecture. The result of the assessment for example, which scenarios are poorly supported or which usability properties or patterns have not been considered, may guide the architect in applying particular transformations to improve the architecture's support of usability. We have elaborated the various steps in this paper, discussed the issues and techniques for each of the steps, and illustrated these by discussing some examples from the case studies.

6. References

[1] E. Folmer, J. v. Gurp, and J. Bosch, Software Architecture Analysis of Usability , *The 9th IFIP Working Conference on Engineering for Human-Computer Interaction*, 2004.

[2] E. Folmer, J. v. Gurp, and J. Bosch, *A framework for capturing the relationship between usability and software architecture*, Software Process: Improvement and Practice, Wiley, 2003, pp. 67-87.

[3] L. Bass, J. Kates, and B. E. John, Achieving Usability through software architecture, Technical Report CMU/SEI-2001-TR-005, 1-3-2001.

[4] M. Welie and H. Trætteberg, Interaction Patterns in User Interfaces, *7th Conference on Pattern Languages of Programming (PloP)*, 2000.

[5] J. Tidwell, Interaction Design Patterns, *Conference on Pattern Languages of Programming 1998*, 1998.

[6] Brighton, *The Brighton Usability Pattern Collection.* http://www.cmis.brighton.ac.uk/research/patterns/home.html

[7] M. Welie, *GUI Design patterns*, http://www.welie.com/

[8] J. Tidwell, *Common ground: Pattern Language for Human-Computer Interface Design*, http://www.mit.edu/~jtidwell/interaction_patterns.html

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns elements of reusable object-orientated software.*, Addison-Wesley, Reading, Massachusetts, 1995.

[10] S. Lauesen and H. Younessi, Six styles for usability requirements, *Proceedings of REFSQ'98*, 1998.

[11] J. Nielsen, *Usability Engineering*, Academic Press, Inc, Boston, MA., 1993.

[12] J. Preece, Y. Rogers, H. Sharp, D. Benyon, S. Holland, and T. Carey, *Human-Computer Interaction*, Addison Wesley, Reading, MA, 1994.

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

[14] N. Lassing, P. O. Bengtsson, H. van Vliet, and J. Bosch, *Experiences with ALMA: Architecture-Level Modifiability Analysis*, Journal of systems and software, Elsevier, 2002, pp. 47-57.

[15] ISO, ISO 9241-11 Ergonomic requirements for office work with visual display terminals (VDTs) -- Part 11: Guidance on usability., 1994.

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

[17] J. v. Gurp and J. Bosch, *Design Erosion: Problems and Causes*, Journal of systems and software, Elsevier, 3-1-2002, pp. 105-119.

[18] P. B. Kruchten, The 4+1 View Model of Architecture, IEEE Software, 1995.

[19] R. Kazman, G. Abowd, and M. Webb, SAAM: A Method for Analyzing the Properties of Software Architectures, *16th International Conference on Software Engineering*, 1994.

[20] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, The Architecture Tradeoff Analysis Method, *International Conference on Engineering of Complex Computer Systems*, 8-1-1998.

[21] J. Bosch, *Design and use of Software Architectures: Adopting and evolving a product line approach*, Pearson Education (Addison-Wesley and ACM Press), Harlow, 2000.

[22] C. Lung, Bot.S., and K. K. R. Kaleichelvan, An Approach to Software Architecture Analysis for Evolution and Reusability, *CASCON Proceedings* , 1997.

[23] P. O. Bengtsson and J. Bosch, Architecture Level Prediction of Software Maintaince, *EuroMicro Conference on Software Engineering*, 1999.