Software Architecture Analysis of Usability

Eelke Folmer, Jilles van Gurp, Jan Bosch

University of Groningen, the Netherlands
mail@eelke.com, jilles@jillesvangurp.com, Jan.Bosch@cs.rug.nl

Studies of software engineering projects [1,2] show that a large number of usability related change requests are made after its deployment. Fixing usability problems during the later stages of development often proves to be costly, since many of the necessary changes require changes to the system that cannot be easily accommodated by its software architecture. These high costs prevent developers from meeting all the usability requirements, resulting in systems with less than optimal usability. The successful development of a usable software system therefore must include creating a software architecture that supports the right level of usability. Unfortunately, no architecture-level usability assessment techniques exist. To support software architects in creating a software architecture that supports usability, we present a scenario based assessment technique that has been successfully applied in several cases. 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.

1 Introduction

One of the key problems with many of today's software is that they do not meet their quality requirements very well. In addition, it often proves hard to make the necessary changes to a system to improve its quality. A reason for this is that many of the necessary changes require changes to the system that cannot be easily accommodated by the software architecture [3] The software architecture, 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 [4] does not support the required level of quality.

The work in this paper is motivated by the fact that this also applies to usability. Usability is increasingly recognized as an important consideration during software development; however, many well-known software products suffer from usability issues that cannot be repaired without major changes to the software architecture of these products. This is a problem for software development because it is very expensive to ensure a particular level of usability after the system has been implemented. Studies [1,2] confirm that a significant large part of the maintenance costs of software systems is spent on dealing with usability issues. These high costs can be explained because some usability requirements will not be discovered until the software has been implemented or deployed. This is caused by the following:

- Usability requirements are often weakly specified.
- Usability requirements engineering techniques have only a limited ability to capture all requirements.
- Usability requirements may change during development.

Discovering requirements late is a problem inherent to all software development and is something that cannot be easily solved. The real problem is that it often proves to be hard and expensive to make the necessary changes to a system to improve its usability. Reasons for why this is so hard:

- Usability is often only associated with interface design but usability does also depend on issues such as the information architecture, the interaction architecture and other quality attributes (such as efficiency and reliability) that are all determined by the software architecture. Usability should therefore also be realized at the architectural level.
- Many of the necessary usability changes to a system cannot be easily be accommodated by the software architecture. Some changes that may improve usability require a substantial degree of modification. For example changes that relate to the interactions that take place between the system and the user, such as undo to a particular function or system wide changes such as imposing a consistent look and feel in the interface.

The cost of restructuring the system during the later stages of development has proven to be an one order of magnitude higher than the costs of an initial development [3]. The high costs spent on usability during maintenance can to an extent be explained by the high costs for fixing architecture-related usability issues. Because during design different tradeoffs have to be made, for example between cost and quality, these high costs may prevent developers from meeting all the usability requirements. The challenge is therefore to cost effectively usable software e.g. minimizing the costs & time that are spent on usability.

Based upon successful experiences [5] with architectural assessment of maintainability as a tool for cost effective developing maintainable software, we developed architectural analysis of usability as an important tool to cost effectively development usable software i.e. if any problems are detected at this stage, it is still possible to change the software architecture with relative cheap costs. Software architecture analysis contributes to making sure the software architecture supports usability. Software architecture analysis does not solve the problem of discovering usability requirements late. However, it contributes to an increased awareness of the limitations the software architecture may place on the level of usability that can be achieved. Explicit evaluation of software architectures regarding usability is a technique to come up with a more usable first version of a software architecture that might allow for more “usability tuning” on the detailed design level, hence, preventing some of the high costs incurring adaptive maintenance activities once the system has been implemented.

In [6] an overview is provided of usability evaluation techniques that can be used during the different stages of development, unfortunately, no usability assessment techniques exists that explicitly focus on the assessment of the software architecture. The contribution of this paper is an assessment technique that assists software architects in designing a software architecture that supports usability called SALUTA (Scenario based Architecture Level UsabiliTy Analysis).

The remainder of this paper is organized as follows. In the next section, the relationship between software architecture and usability is discussed. Section 3 discusses various approaches to software architecture analysis. Section 4 presents an overview of the main steps of SALUTA. Section 5 presents some examples from a case study for performing usability analysis in practice and discusses the validation of the method. Finally the paper is concluded in section 6.

2 Relationship between Usability and Software Architecture

A software architecture description such as a decomposition of the system into components and relations with its environment may provide information on the support for particular quality attributes. Specific relationships between software architecture (such as - styles, -patterns etc) and quality attributes (maintainability, efficiency) have been described by several authors. [7,8,3]. For example [7] describes the architectural pattern layers and the positive effect this pattern may have on exchangeability and the negative effect it may have on efficiency.

Until recently [9,10] such relationships between usability and software architecture had not been described nor investigated. In [10] we defined a framework that expresses the relationship between usability and software architecture based on our comprehensive survey [6]. This framework is composed of an integrated set of design solutions such as usability patterns and usability properties that have a positive effect on usability but are difficult to retrofit into applications because they have architectural impact. The framework consists of the following concepts:

2.1 Usability attributes

A number of usability attributes 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.

2.2 Usability properties

A number of usability properties have been selected from literature that embody the heuristics and design principles that researchers in the usability field consider to have a direct positive influence on usability. They should be considered as high-level

design primitives that have a known effect on usability and most likely have architectural implications. Some examples:

- Providing Feedback - The system should provide at every (appropriate) moment feedback to the user in which case he or she is informed of what is going on, that is, what the system is doing at every moment.
- Consistency - Users should not have to wonder whether different words, situations, or actions mean the same thing. Consistency has several aspects:
  - Visual consistency: user interface elements should be consistent in aspect and structure.
  - Functional consistency: the way to perform different tasks across the system should be consistent.
  - Evolutionary consistency: in the case of a software product family, consistency over the products in the family is an important aspect.

2.3 Architecture sensitive usability patterns

A number of usability patterns have been identified that should be applied during the design of a system's software architecture, rather than during the detailed design stage. This set of patterns has been identified from various cases in industry, modern software, literature surveys as well as from existing (usability) pattern collections. Some examples:

- Actions on multiple objects - Actions need to be performed on objects, and users are likely to want to perform these actions on two or more objects at one time [11].
- Multiple views - The same data and commands must be potentially presented using different human-computer interface styles for different user preferences, needs or disabilities [12].
- User profiles - The application will be used by users with differing abilities, cultures, and tastes [11].

Unlike the design patterns, architecturally sensitive patterns do not specify a specific design solution in terms of objects and classes. Instead, potential architectural implications that face developers looking to solve the problem the architecturally sensitive pattern represents are outlined. For example, to facilitate actions on multiple objects, a provision needs to be made in the architecture for objects to be grouped into composites, or for it to be possible to iterate over a set of objects performing the same action for each. Actions for multiple objects may be implemented by the composite pattern [8] or the visitor pattern [8].

(Positive) relationships have been defined between the elements of the framework that link architectural sensitive usability patterns to usability properties and attributes. These relationships have been derived from our literature survey. The usability properties in the framework may be used as requirements during design. For example, if a requirements species, "the system must provide feedback", we use the framework to identify which usability patterns may be implemented to fulfill these properties by following the arrows in Figure 1. Our assessment technique uses this framework to analyze the architecture's support for usability.

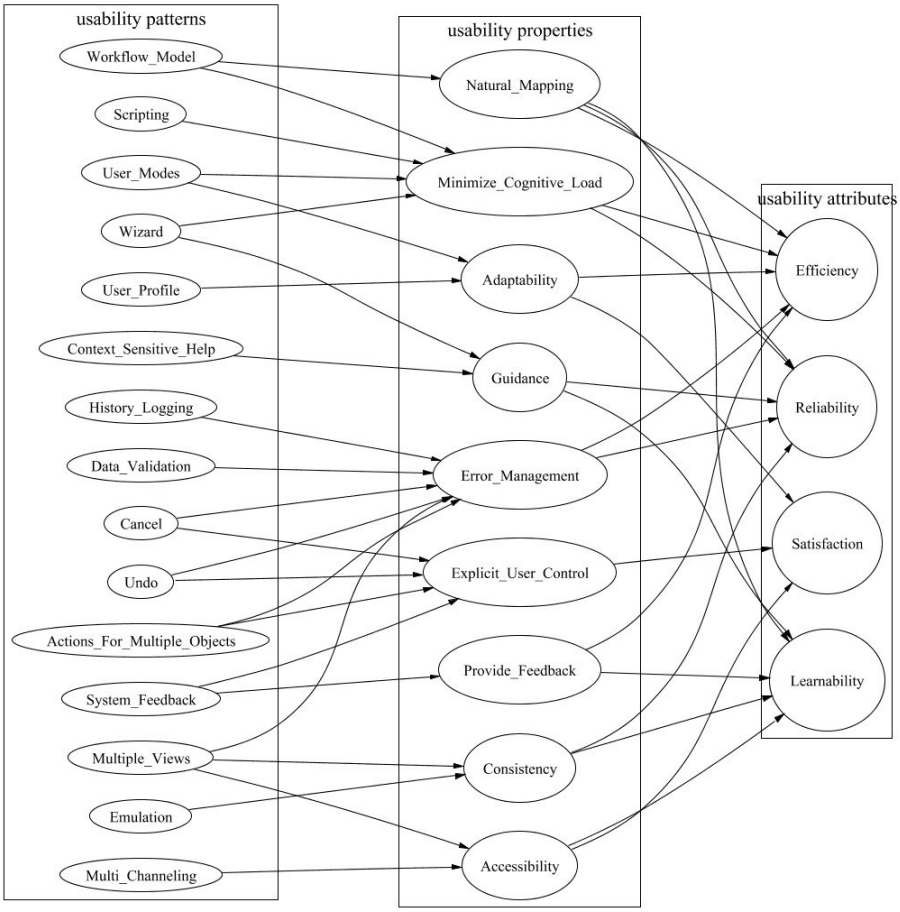

Fig. 1. Usability Framework

3 Software architecture assessment

The design and use of an explicitly defined software architecture has received increasing amounts of attention during the last decade. Generally, three arguments for defining an architecture are used [13]. First, it provides an artifact that allows discussion by the stakeholders very early in the design process. Second, it allows for early assessment of quality attributes [14,3]. Finally, the design decisions captured in the software architecture can be transferred to other systems.

Our work focuses on the second aspect: early assessment of usability. Most engineering disciplines provide techniques and methods that allow one to assess and test quality attributes of the system under design. For example for maintainability

assessment code metrics [15] have been developed. In [6] an overview is provided of usability evaluation techniques that can be used during software development. Some of the more popular techniques such as user testing [16], heuristic evaluation [17] and cognitive walkthroughs [18] can be used during several stages of development. Unfortunately, no usability assessment techniques exist that focus on assessment of software architectures. Without such techniques, architects may run the risk of designing a software architecture that fails to meet its usability requirements. To address to this problem we have defined a scenario based assessment technique (SALUTA).

The Software Architecture Analysis Method (SAAM) [19] was among the first to address the assessment of software architectures using scenarios. SAAM is stakeholder centric and does not focus on a specific quality attribute. From SAAM, ATAM [14] has evolved. ATAM also uses scenarios for identifying important quality attribute requirements for the system. Like SAAM, ATAM does not focus on a single quality attribute but rather on identifying tradeoffs between quality attributes. SALUTA can be integrated into these existing techniques.

3.1 Usability specification

Before a software architecture can be assessed for its support of usability, the required usability of the system needs to be determined. Several specification styles of usability have been identified [20]. One shortcoming of these techniques [17,21,22] is that they are poorly suited for architectural assessment.

- Usability requirements are often rather weakly specified: practitioners have great difficulties specifying usability requirements and often end up stating: “the system shall be usable” [20].
- Many usability requirements are performance based specified [20]. For example, such techniques might result in statements such as “customers must be able to withdraw cash within 4 minutes” or “80% of the customers must find the system pleasant”.

Given an implemented system, such statements may be verified by observing how users interact with the system. However, during architecture assessment such a system is not yet available. Interface prototypes may be analyzed for such requirements however we want to analyze the architecture for such requirements.

A technique that is used for specifying the required quality requirements and the assessment of software architectures for these requirements are scenario profiles [5]. Scenario profiles describe the semantics of software quality attributes by means of a set of scenarios. The primary advantage of using scenarios is that scenarios represent the actual meaning of a requirement. Consequently, scenarios are much more specific and fine-grained than abstract usability requirements. The software architecture may then be evaluated for its support for the scenarios in the scenario profile. Scenario profiles and traditional usability specification techniques are not interfering; scenarios are just a more concrete instance of these usability requirements.

3.2 Usage profiles

A usage profile represents the required usability of the system by means of a set of usage scenarios. Usability is not an intrinsic quality of the system. According to the ISO definition [23], usability depends on:

- The users - who is using the product? (system administrators, novice users)
- The tasks - what are the users trying to do with the product? (insert order, search for item X)
- The context of use - where and how is the product used? (helpdesk, training environment)

Usability may also depend on other variables, such as goals of use, etc. However in a usage scenario only the variables stated above are included. A usage scenario is defined as "an interaction (task) between users, the system in a specific context of use". A usage scenario specified in such a way does not yet 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. For each usage scenario, numeric values are determined for each of these usability attributes. The numeric values are used to determine a prioritization between the usability attributes.

For some usability attributes, such as efficiency and learnability, tradeoffs have to be made. 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 [20]. For example, if for a particular usage scenario learnability is considered to be of more importance than other usability attributes (maybe because of a requirement), 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 (section 4.3) to determine the level of support in the software architecture for the usage scenario.

4 SALUTA

In this section we present SALUTA (Scenario based Architecture Level UsabiliTy Analysis. SALUTA consists of the following four steps:

1. Create usage profile.
2. Describe provided usability.
3. Evaluate scenarios.
4. Interpret the results.

When performing an analysis the separation between these steps is not very strict and it is often necessary to iterate over various steps. In the next subsections, however the steps are presented as if they are performed in strict sequence.

4.1 Create usage profile

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

1. Identify the users: rather than listing individual 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: Instead of converting the complete functionality of the system into 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: In this step, representative contexts of use are identified. (For example. Helpdesk context or disability context.) Deciding what users, tasks and contexts of use to include requires making tradeoffs between all sorts of factors. An important consideration is that the more scenarios are evaluated the more accurate the outcome of the assessment is, but the more expensive and time consuming it is to determine attribute values for these scenarios.
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. Defining specific indicators for attributes may assist the analyst in interpreting usability requirements as will be illustrated in the case study in section 5. To reflect the difference in priority, numeric values between one and four have been assigned to the attributes for each scenario. Other techniques such as pair wise comparison may also be used to determine a prioritization between attributes.
5. Scenario selection & weighing: Evaluating all identified 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 different architectures, scenarios may be selected that highlight the differences between those architectures. If the goal is to predict the level of usability for an architecture, 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. The usage profile that is created using these steps is summarized in a table (See Table 2).

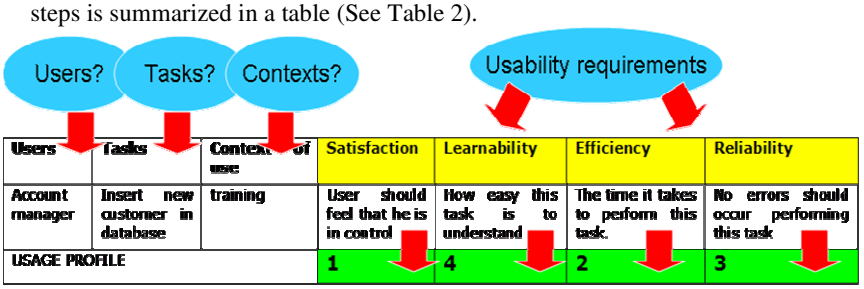

Fig. 2. Example usage profile

This step results in a set of usage scenarios that accurately express the required usability of the system. Usage profile creation is not intended to replace existing requirements engineering techniques. Rather it is intended to transform (existing) usability requirements into something that can be used for architecture assessment. Existing techniques such as such as interviews, group discussions or observations [17,22,24] typically 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 (such as a usability engineer) is required. The usability engineer may fill in the missing information on the usability requirements, because usability requirements are often not explicitly defined.

4.2 Describe provided usability

In the second step of SALUTA, the information about the software architecture is collected. Usability analysis requires architectural information that allows the analyst to determine the support for the usage scenarios. The process of identifying the support is similar to scenario impact analysis for maintainability assessment [5] but is different, because it focuses on identifying architectural elements that may support the 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 [25]. 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 property support that is extracted from the architecture. Some usability properties such as error management may be implemented using architectural patterns such as undo, cancel or data validation. However, 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 [26] 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.

4.3 Evaluate scenarios

SALUTA's next step is to evaluate the support for each of the scenarios in the usage profile. For each scenario, it is analyzed by which usability patterns and properties, that have been identified in the previous step, it is affected. A technique we have used for identifying the provided usability in our cases is the usability framework approach. The relations defined in the framework are used to analyze how a particular pattern or property affects a specific usability attribute. For example if it has been identified that undo affects a certain scenario. Then the relationships of the undo pattern with usability are analyzed (see Figure 1) to determine the support for that particular scenario. Undo in this case may increase reliability and efficiency. 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. See Figure 2 for a snapshot assessment example.

Users	Tasks	Context of use	Satisfaction	Learnability	Efficiency	Reliability
Account manager	Insert new customer in database	training	User should feel that he is in control	How easy this task is to understand	The time it takes to perform this task.	No errors should occur performing this task
USAGE PROFILE			1	4	2	3

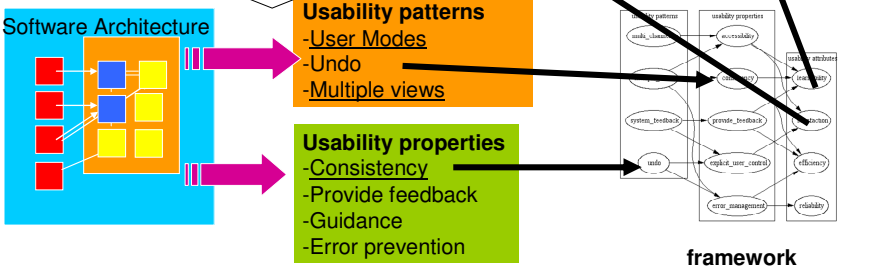

Fig. 3. Snapshot evaluation example

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.

4.4 Interpret the results

Finally, after scenario evaluation, the results need to be interpreted to draw conclusions concerning the software architecture. This interpretation depends on two factors: the goal of the analysis and the usability requirements. 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 analysis is sufficiently accurate the results may be quantified, however even without quantification the assessment can produce useful results. If the goal is to iteratively design an architecture, then if the architecture proves to have sufficient support for usability, 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.

5 Validation

In order to validate SALUTA it has been applied in three case studies:

- eSuite. A web based enterprise resource planning (ERP) system.
- Compressor. A web based e-commerce system.
- Webplatform. A web based content management system (CMS)

The goal of the case studies was twofold: first to conduct a software architecture analysis of usability on each of the three systems and to collect experiences. Our technique had initially only been applied at one case study and we needed more experiences to further refine our technique and make it generally applicable. Second, our goal was to gain a better understanding of the relationship between usability and software architecture. Our analysis technique depends on the framework we developed in [9]. Analyzing architectural designs in the case studies allowed us to further refine and validate the framework we developed. As a research method we used action research [27], we took upon our self the role of external analysts and actively participated in the analysis process and reflected on the process and the results.

These cases studies show that it is possible to use SALUTA to assess software architectures for their support of usability. Whether we have accurately predicted the architecture's support for usability is answered by comparing our analysis with the results of user tests that are conducted when the systems are implemented. These results are used to verify whether the usage profile we created actually matches the actual usage of the system and whether the results of the assessment fits results from the user tests For all three cases, the usage profile and architecture assessment phase is completed. In the case of the Webplatform, a user test has been performed recently. In this article, we limit ourselves to highlighting some examples from the Webplatform case study.

ECCOO develops software and services for one of the largest universities of the Netherlands (RuG). ECCOO has developed the Webplatform. Faculties, departments and organizations within the RuG are already present on the inter/intra/extra –net but because of the current wild growth of sites, concerning content, layout and design, the usability of the old system was quite poor. For the Webplatform usability was

considered as an important design objective. Webplatform has successfully been deployed recently and the current version of the RuG website is powered by the Webplatform. As an input to the analysis of the Webplatform, we interviewed the software architect and usability engineer, examined the design documentation, and looked at the newly deployed RuG site. In the next few subsections, we will present the four SALUTA steps for the Webplatform.

5.1 Usage profile creation

In this step of the SALUTA, we have cooperated with the usability engineer to create the usage profile.

- Three types of users are defined in the functional requirements: end users, content administrators and CMS administrators.
- Several different tasks are specified in the functional requirements. An accurate description of what is understood for a particular task is an essential part of this step. For example, several tasks such as "create new portal medical sciences" or "create new course description" have been understood for the task "make object", because the Webplatform data structure is object based.
- No relevant contexts of use were identified for Webplatform. Issues such as bandwidth or helpdesk only affect a very small part of the user population.

The result of the first three steps is summarized in Table 1.

Table 1. Summary of selected users, tasks for Webplatform

#	Users	Tasks	example
1	End-user	Quick search	Find course X
2	End-user	Navigate	Find employee X
3	Content Administrator	Edit object	Edit course description
4	Content Administrator	Make object	Create new course description
5	Content Administrator	Quick search	Find course X
6	Content Administrator	Navigate	Find phone number for person X
7	CMS Administrator	Edit object	Change layout of portal X
8	CMS Administrator	Make object	Create new portal medical sciences
9	CMS Administrator	Delete object	Delete teacher X
10	CMS Administrator	Quick search	Find all employees of section X
11	CMS Administrator	Navigate	Find section library

The next step is to determine attribute values for the scenarios. This has been done by consulting the usability requirements and by discussing these for each scenario with the usability engineer. In the functional requirements of the Webplatform only 30 guidelines based on Nielsen's heuristics [17] have been specified. Fortunately, the usability engineer in our case had a good understanding of the expected required usability of the system. As an example we explain how we determined attribute values for the usage scenario: "end user performing quick search".

First, we formally specified with the usability engineer what should be understood for each attribute of this task. We have associated reliability with the accuracy of search results; efficiency has been associated with response time of the quick search. Then the usability requirements were consulted. A usability requirement that affects this scenario states: "every page should feature a quick search which searches the whole portal and comes up with accurate search results". In the requirements, it has not been specified that quick search should be performed quickly. However, in our discussions with the usability engineer we found that this is the most important aspect of usability for this task. Consequently, high values have been given to efficiency and reliability and low values to the other attributes. For each scenario, numeric values between one and four have been assigned to the usability attributes to express the difference in priority. Table 2 states the result of the quantification of the selected scenarios for Webplatform.

#	Users	Tasks	S	L	E	R
1	End-user	Quick search	2	1	4	3
2	End-user	Navigate	1	4	2	3
3	Content Administrator	Edit object	1	4	2	3
4	Content Administrator	Make object	1	4	2	3
5	Content Administrator	Quick search	2	1	4	3
6	Content Administrator	Navigate	1	4	2	4
7	CMS Administrator	Edit object	2	1	4	3
8	CMS Administrator	Make object	2	1	4	3
9	CMS Administrator	Delete object	2	1	4	3
10	CMS Administrator	Quick search	2	1	4	3
11	CMS Administrator	Navigate	1	2	3	4

Table 2. Attribute priority table for Webplatform

5.2 Architecture description

For scenario evaluation, a list of usability patterns and a list of usability properties that have been implemented in the system are required to determine the architecture's support for usability. This information has been acquired, by analyzing the software architecture (see Figure 3), consulting the functional design documentation (some specific design decisions for usability had been documented) and interviewing the software architect using the list of patterns and properties defined in our framework.

One of the reasons to develop Webplatform was that the usability of the old system was quite poor; this was mainly caused by the fact that each "entity" within the RuG (Faculties, libraries, departments) used their own layout and their own way to present information and functionality to its users which turned out to be very confusing to users.

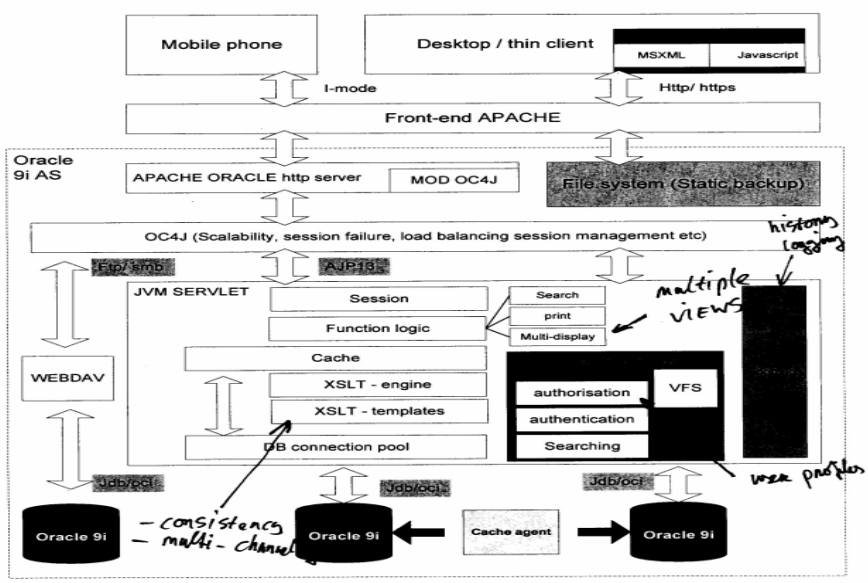

Fig. 4. Webplatform software architecture

A specific design decision that was taken which facilitates several patterns and properties in our framework was to use the internet file system (IFS):

- Multiple views [10]]: The IFS provides an interface that realizes the use of objects and relations as defined in XML. Using XML and XSLT templates the system can provide multiple views for different users and uses on the server side. CSS style sheets are used to provide different views on the client site, for example for providing a "print" layout view but also to allow each faculty their own "skin" as depicted in Figure 3.
- Consistency [10]: The use of XML/ XSLT is a means to enforce a strict separation of presentation from data. This design decision makes it easier to provide a consistent presentation of interface and function for all different objects of the same type such as portals. See for example Figure 6 where the menu layout, the menu items and the position of the quick search box is the same for the faculty of arts and the faculty of Philosophy.
- Multichanneling [8]: By providing different views & control mappings for different devices multichanneling is provided. The Webplatform can be accessed from an I-mode phone as well as from a desktop computer.

Next to the patterns and properties that are facilitated by the IFS several other patterns and properties were identified in the architecture. Sometimes even multiple instances of the same property (such as system feedback) have been identified. Some properties such as consistency have multiple aspects (visual/functional consistency). We need to analyze the architecture for its support of each element of such a property A result of such a detailed analysis for the property accessibility and the pattern history logging is displayed in Table 3.

Fig. 5. Provide multiple views/ & Visual/Functional Consistency.

Table 3.

[pattern]- History Logging	- There is a component that logs every user action. It can be further augmented to also monitor system events (i.e. "the user failed to login 3 consecutive times"). History logging is especially helpful for speeding up the object manipulation process. - Cookies are used to prevent users from having to login again when a connection is lost. Cookies also serve as a backup mechanism on the client site. (To retrieve lost data).
[property] - Accessibility	
• Disabilities	×
• Multi channel	Multi channeling is provided by the web server which can provide a front end to I-Mode or other devices based on specified XLST templates.
• Internationalization	- Support for Dutch / English language, each xml object has different language attribute fields. - Java support Unicode

5.3. Evaluate scenarios

The next step is to evaluate the architecture's support for the usage scenarios in the usage profile. As an example, we analyze usage scenario #4 "content administrator makes object" from table 2. For this scenario it has been determined by which patterns and properties, that have been identified in the architecture it is affected. It is important to identify whether a scenario is affected by a pattern or property that has been implemented in the architecture because this is not always the case. The result of such an analysis is shown in a support matrix in Table 3 for two scenarios. A checkmark indicates that the scenario is affected by at least one or more patterns or properties. Some properties such as consistency have multiple aspects (visual/functional consistency). For a thorough evaluation we need to analyze each scenario for each element of such a property. The support matrix is used together with the relations in the framework to find out whether a usage profile is sufficiently supported by the architecture. The usage profile that we created shows that scenario #4 has high values for learnability (4) and reliability (3). Several patterns and

properties positively contribute to the support of this scenario. For example, the property consistency and the pattern context sensitive help increases learnability as can be analyzed from Figure 1. By analyzing for each pattern and property, the effect on usability, the support for this scenario is determined. Due to the lack of formalized knowledge at the architecture level, this step is very much guided by tacit knowledge (i.e. the undocumented knowledge of experienced software architects and usability engineers). For usage scenario #4, we have concluded that the architecture provides weak support. Learnability is very important for this scenario and patterns such as a wizard or workflow modeling or different user modes to support novice users could increase the learnability of this scenario.

Table 4. Architecture support matrix

Scenario number	**Usability patterns**																**Usability properties**								
	System Feedback	Actions for multiple obj.	Cancel	Data validation	History Logging	Scripting	Multiple views	Multi Channeling	Undo	User Modes	User Profiles	Wizard	Workflow model	Emulation	Context sensitive help	Provide feedback	Error management	Consistency	Adaptability	Guidance	Explicit user control	Natural mapping	Accessibility	Minimize cognitive load
1	✓	×	×	×	×	×	×	✓	×	×	×	×	✓	×	✓	✓	✓	✓	×	✓	×	✓	✓	×
4	✓	×	✓	✓	✓	✓	✓	✓	×	×	×	×	×	×	✓	✓	✓	✓	×	✓	✓	✓	✓	×

5.4. Interpret the results

The result of the assessment of the Webplatform is that three scenarios are accepted, six are weakly accepted and that two scenarios are weakly rejected. The main cause for this is that we could not identify sufficient support for learnability for content administrators as was required by the usage profile. There is room for improvement; usability could be improved if provisions were made to facilitate patterns and properties that have not been considered. The usability requirement of consistency was one of the driving forces of design and our analysis shows that it has positive influence on the usability of the system. Apart from some general usability guidelines [17] stated in the functional requirements no clearly defined and verifiable usability requirements have been specified. Our conclusion concerning the assessment of the Webplatform is that the architecture provides sufficient support for the usage profile that we created. This does not necessarily guarantee that the final system will be usable since many other factors play a role in ensuring a system's usability. Our analysis shows however that these usability issues may be repaired without major changes to the software architecture thus preventing high costs incurring adaptive maintenance activities once the system has been implemented.

5.5. Validation

Whether the usage profile we created is fully representative for the required usability is open to dispute. However, the results from the user test that has recently been completed by the ECCOO is consistent with our findings. 65 test users (students, employees and graduate students) have tested 13 different portals. In the usability tests, the users had to perform specific tasks while being observed. The specific tasks that had to be performed are mostly related to the tasks navigation and quick search in our usage profile. After performing the tasks, users were interviewed about the relevance of the tasks they had to perform and the usability issues that were discovered. The main conclusions of the tests are:

- Most of the usability issues that were detected were related to navigation, structure and content. For example, users have difficulties finding particular information. Lack of hierarchy and structure is the main cause for this problem Although the architecture supports visual and functional consistency, organizations themselves are responsible for structuring their information.
- Searching does not generate accurate search results. This may be fixed by technical modifications. E.g. tuning the search function to generate more accurate search results. (This is also caused by that a lot of meta-information on the content in the system has not been provided yet).

The results of this usability tests fit the results of our analysis: the software architecture supports the right level of usability. 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. Future usability tests will focus on analyzing the usability of the scenarios that involve content administrators. Preliminary results from these tests show that the system has a weak support for learnability as predicted from the architectural analysis.

7. Conclusions

In this paper, we have presented SALUTA, a scenario based assessment technique that assists software architects in designing a software architecture that supports usability. SALUTA consists of four major steps: First, the required usability of the system is expressed by means of a usage profile. The usage profile consists of a representative set of usage scenarios that express the required usability of the system. 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 a framework that has been developed in earlier work. This framework consists of an integrated set of design solutions such as usability patterns and usability properties that have a positive effect on usability but are difficult to retrofit into applications because they have architectural impact. This framework is used to analyze the architecture for its support of usability. The next step is to evaluate the architecture's support of usage profile using the information extracted in the previous step. To do so, we perform support analysis for each of the

scenarios in the set. 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 a case study. The main contributions of this paper are:

- SALUTA is the first and currently the only technique that enables software architects to assess the level of usability supported by their architectures.
- Because usability requirements tend to change over time and may be discovered during deployment, SALUTA may assist a software architect to come up with a more usable first version of a software architecture that might allow for more "usability tuning" on the detailed design level. This prevents some of the high costs incurring adaptive maintenance activities once the system has been implemented.

Future work shall focus on finalizing the case studies, refining the usability framework and validating our claims we make. Preliminary experiences with these three case studies shows the results from the assessment seem reasonable and do not conflict with the user tests. In the future, we will not only focus on assessing with SALUTA but also on using the framework for iteratively designing software architectures with SALUTA.

Acknowledgments

This work is sponsored by the STATUS[1] project under contract no IST-2001-32298. We would like to thank the partners in the STATUS project and ECCOO for their input and their cooperation.

References

[1] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, McGraw-Hill, NY, 1992.

[2] T. K. Landauer, *The Trouble with Computers: Usefulness, Usability and Productivity.*, MIT Press., Cambridge, 1995.

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

[4] IEEE, IEEE Architecture Working Group. Recommended practice for architectural description. Draft IEEE Standard P1471/D4.1, IEEE, 1998.

[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.

[5] 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.
[6] E. Folmer and J. Bosch, *Architecting for usability; a survey*, Journal of systems and software, Elsevier, 2002, pp. 61-78.
[7] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley and Son Ltd, New York, 1996.
[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns elements of reusable object-orientated software.*, Addison -Wesley, 1995.
[9] L. Bass, J. Kates, and B. E. John, Achieving Usability through software architecture, 1-3-2001.
[10] E. Folmer, J. v. Gurp, and J. Bosch, *Investigating the Relationship between Usability and Software Architecture* , Software process improvement and practice, Wiley, 2003, pp. 0-0.
[11] J. Tidwell, Interaction Design Patterns, *Conference on Pattern Languages of Programming 1998*, 1998.
[12] Brighton, *The Brighton Usability Pattern Collection.* http://www.cmis.brighton.ac.uk/research/patterns/home.html
[13] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Addison Wesley Longman, Reading MA, 1998.
[14] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, The Architecture Tradeoff Analysis Method, *Proceedings of ICECCS'98*, 8-1-1998.
[15] W. Li and S. Henry, *OO Metrics that Predict Maintainability*, Journal of systems and software, Elsevier, 1993, pp. 111-122.
[16] J. Nielsen, *Heuristic Evaluation.*, in Usability Inspection Methods., Nielsen, J. and Mack, R. L., John Wiley and Sons, New York, NY., 1994.
[17] J. Nielsen, *Usability Engineering*, Academic Press, Inc, Boston, MA., 1993.
[18] C. Wharton, J. Rieman, C. H. Lewis, and P. G. Polson, *The Cognitive Walkthrough: A practitioner's guide.*, in Usability Inspection Methods, Nielsen, Jacob and Mack, R. L., John Wiley and Sons, New York, NY., 1994.
[19] R. Kazman, G. Abowd, and M. Webb, SAAM: A Method for Analyzing the Properties of Software Architectures, *Proceedings of the 16th International Conference on Software Engineering*, 1994.
[20] S. Lauesen and H. Younessi, Six styles for usability requirements, *Proceedings of REFSQ'98*, 1998.
[21] J. Preece, Y. Rogers, H. Sharp, D. Benyon, S. Holland, and T. Carey, *Human-Computer Interaction*, Addison Wesley, 1994.
[22] D. Hix and H. R. Hartson, *Developing User Interfaces: Ensuring Usability Through Product and Process.*, John Wiley and Sons, 1993.
[23] ISO, ISO 9241-11 Ergonomic requirements for office work with visual display terminals (VDTs) -- Part 11: Guidance on usability., 1994.
[24] B. Shneiderman, *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Addison-Wesley, Reading, MA, 1998.
[25] J. v. Gurp and J. Bosch, *Design Erosion: Problems and Causes*, Journal of systems and software, Elsevier, 3-1-2002, pp. 105-119.
[26] P. B. Kruchten, The 4+1 View Model of Architecture, IEEE Software, 1995.
[27] C. Argyris, R. Putnam, and D. Smith, *Action Science: Concepts, methods and skills for research and intervention*, Jossey-Bass, San Francisco, 1985.