Scenario-based Assessment of Software Architecture Usability

Eelke Folmer, Jilles van Gurp, Jan Bosch
Department of Mathematics and Computing Science
University of Groningen, PO Box 800, 9700 AV the Netherlands
mail@eelke.com, Jilles@cs.rug.nl, Jan.Bosch@cs.rug.nl

Abstract

Over the years the software engineering community has increasingly realized the important role software architecture plays in fulfilling the quality requirements of a system. The quality attributes of a software system are, to a large extent determined by the system's software architecture. In recent years various tools and techniques have been developed that allow for design for quality attributes, such as performance or maintainability, at the software architecture level. We believe this design approach can be applied not only to non-operational quality attributes such as performance or maintainability, but also to operational quality attributes such as usability. This paper presents and describes a scenario based assessment method to assess whether a given software architecture (provided usability) meets the usability requirements (required usability).

1. Introduction

The quality attributes of a software system are to a considerable extent defined by its software architecture. In addition, design decisions in the beginning of the design process are the hardest to revoke. Therefore it is important to have an explicit and objective design process. Various researchers in the software engineering research community have proposed software architecture design methods: SAAM [1], ATAM [2] and QASAR [3]. The latter, the Quality Attribute-oriented Software ARchitecture design method (QASAR), is a method for software architecture design that employs explicit assessment of, and design for the quality requirements of a software system.

The architecture design process depicted in Figure 1 can be viewed as a function that transforms a requirement specification to an architectural design. The requirements are collected from the stakeholders; the users, customers, technological developments and the marketing departments. These groups often provide conflicting requirements and have to agree on a specific set of requirements before the design process can start. The design process starts with a design of the software

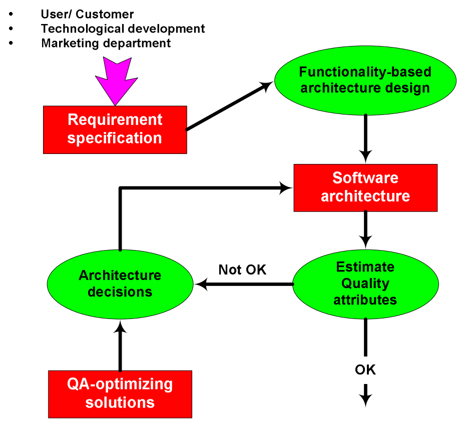
• User/ Customer
• Technological development
• Marketing department
Requirement specification
Functionality-based architecture design
Software architecture
Estimate Quality attributes
Not OK
Architecture decisions
QA-optimizing solutions
OK

Figure 1: Software architecture design method

architecture based on the functional requirements. Although software engineers will not design a system on purpose that is unreliable or performs poorly, most non-functional requirements are typically not explicitly defined at this stage.

The design process results in a preliminary version of the software architecture design. This design is evaluated with respect to the quality requirements by using a qualitative or quantitative assessment technique. During assessment the provided quality attributes are compared to the required quality attribute specifications. If these are satisfactory, then the design process is finished. Otherwise, the architecture transformation or improvement stage is entered. This stage improves the software architecture by selecting appropriate quality attribute optimizing or improving design solutions.

When applying architecture design solutions, generally one or more quality attributes are improved whereas other attributes may be affected negatively. By applying one or more architectural design solutions, a new architectural design is created. The design is evaluated again and the process is repeated, if necessary, until all non-functional requirements have been satisfied as much as possible. Other design methods such as SAAM or ATAM take a similar approach with respect to iterative refinement of the design. Generally some compromises are necessary with respect to conflicting non-functional requirements. The design process described here depends on two requirements:

- It is required to determine when the software design process is finished. Therefore, assessment techniques are needed to provide quantitative or qualitative data, to determine if the architecture meets the non-functional requirements.
- Development or identification of architectural design solutions that improve quality attributes.

As of yet, no architectural assessment techniques for usability exist. The goal of this paper is to outline and present an assessment technique for usability that fulfills one of the requirements to be able to design for usability on the architectural level.

2. Architecture assessment of usability

Most usability issues are only discovered late in the development process, during testing and deployment. This late detection of usability issues is largely due to the fact that in order to do a usability evaluation, it is necessary to have both a working system and a representative set of users present. This evaluation can only be done at the end of the design process. It is therefore expensive to go back and make changes at this stage. Most usability improving modifications are structural and can hence not be implemented because of their cost

One of the goals of the STATUS[1] project is to develop techniques and methods that can assess software architectures for their support of usability. The reason for developing such techniques is because the quality attributes of a software system are, to a large extent determined by a system's software architecture. We believe this not only holds for non-operational quality attributes such as maintainability or modifiability but also for usability.

Being able to assess the quality attributes such as usability during early development therefore is very important. Three types of architecture assessment have been identified [3]

- Scenario based assessment: In order to assess a particular architecture, a set of scenarios is developed that concretizes the actual meaning of a requirement. For instance, the maintainability requirements may be specified by defining change profiles that captures typical changes in the requirements, underlying hardware and so on. For each scenario the architecture is assessed for its support of this scenario.
- Simulation: Simulation of the architecture uses an executable model of the application architecture. This comprises models of the main components of the system composed to form an overall architecture model. It is possible to check various properties of such a model in a formal way and to animate it to allow the user or designer to interact with the model as they might with the finished system.
- Mathematical modeling: By using mathematical models developed by various research communities such as high performance computing, operational quality attributes can be assessed. Mathematical modeling is closely related to, or an alternative to simulation.

It is our conjecture that a scenario based approach for developing an assessment method for usability is the most promising candidate technique. Mathematical modeling and simulation are better suited to assess operational quality attributes because such quality attributes, for example performance, are easier and more accurately estimated than usability. A scenario is similar to a use case for those familiar with object oriented modeling. A scenario is defined as a short statement describing and interaction of one of the stakeholders with the system in a particular context. The usage of scenarios is motivated by the consensus it brings to the understanding of what a particular software quality really means. Scenarios are a good way of synthesizing individual interpretations of a software quality into a common view. This view is both more concrete than the general software quality definitions [4] and also incorporates the uniqueness of the system to be developed, i.e. it is more context sensitive.

Traditionally, scenario based assessment has been applied to development related software qualities [5]. Software qualities such as maintainability can be expressed very naturally through change scenarios. The safety quality attribute may be specified by hazard scenarios In [1] the use of scenarios for assessing architectures is also identified. It is our conjecture that scenario based assessment can also be applied for usability assessment. Usability is often defined in a very abstract fashion. Scenarios can make abstract usability requirements more specific. For example a usability requirement like "the system should be learnable" is much harder to evaluate for a system than a usage scenario defined as: "For a novice user operating on a helpdesk context, inserting a new customer in the sales database should be learnable", which is a more concrete statement.

Before we developed an assessment technique for usability the relationship between usability and architecture was investigated. The results of that research are presented in the next section.

3. Usability Framework

[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.

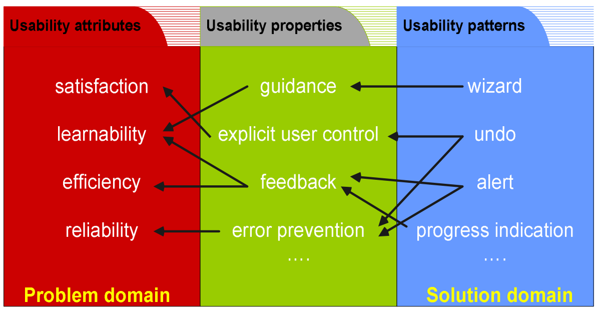

Figure 2: Usability framework

One of the first goals of the STATUS project was to investigate the relationship between usability and software architecture. A framework has been developed [6] which illustrates this relationship. This framework provides the basis for developing assessment tools for usability. The framework is used for extracting information regarding the architectural information related to usability required for the assessment (this will be discussed when presenting our assessment technique). The framework consists of the following concepts:

- Usability attributes.
- Usability properties.
- Usability patterns.

Figure gives some examples of attributes, properties and patterns, and shows how these are related to illustrate the relationship between usability and software architecture. The concepts used are defined below.

3.1. Usability attributes

A comprehensive survey of the literature [7] revealed that different researchers have different definitions for the term usability attribute, but the generally accepted meaning is that a usability attribute is a precise and measurable component of the abstract concept that is usability. After an extensive search of the work of various authors, the following set of usability attributes has been identified for which software systems in our work are assessed. No innovation was applied in this area, since abundant research has already focused on finding and defining the optimal set of attributes that compose usability. Therefore, merely the set of attributes most commonly cited amongst authors in the usability field has been taken. The four attributes that are chosen are:

- Learnability - how quickly and easily users can begin to do productive work with a system that is new to them, combined with the ease of remembering the way a system must be operated.
- Efficiency of use - the number of tasks per unit time that the user can perform using the system.
- Reliability in use - this attribute refers to the error rate in using the system and the time it takes to recover from errors.
- Satisfaction - the subjective opinions that users form in using the system.

These attributes can be measured directly by observing and interviewing users of the final system using techniques that are well established in the field of usability engineering.

3.2. Usability properties

Essentially, usability properties embody the heuristics and design principles that researchers in the usability field have found to have a direct influence on system usability. Usability properties cannot be observed when evaluating usability for an implemented system. These properties can be used as requirements at the design stage, for instance by specifying "the system must provide feedback", however they are not strict requirements in way that they are requirements that should be fulfilled at all costs. Usability properties should be considered as higher-level design primitives, which have a known effect on usability. It is up to the software engineer to decide how and at which levels these properties are implemented by using usability patterns of which it is known they have an effect on this usability property. The following properties have been defined:

- Providing feedback - the system provides continuous feedback as to system operation to the user.
- Error management - includes error prevention and recovery.
- Consistency - consistency of both the user interface and functional operation of the system.
- Guidance - on-line guidance as to the operation of the system.
- Minimize cognitive load - system design should recognize human cognitive limitations, short-term memory etc.
- Natural mapping - includes predictability of operation, semiotic significance of symbols and ease of navigation.
- Accessibility - includes multi-mode access, internationalization and support for disabled users.

3.3. Usability patterns

The term usability pattern refers to a technique or mechanism that can be applied to the design of the architecture of a software system in order to address a need identified by a usability property at the requirements stage (or iteration thereof). Various pattern collections have been defined [8], [9], the difference with other collections is that our collection considers only patterns which should be applied during the design of a system's software architecture, rather than during the detailed design stage. There is not a one-to-one mapping between usability patterns and the usability properties that they affect. A pattern may be related to any number of properties, and each property may be improved (or impaired) by a number of different patterns. The choice of which pattern to apply may be made on the basis of cost and the trade off between different usability properties or between usability and other quality attributes such as security or performance. 20 patterns have been identified and a detailed analysis of each usability pattern and the relationship between this patterns, usability properties and usability attributes can be found on http://www.designforquality.com/

The next section presents our proposal of a scenario based assessment method for architectural assessment of usability. The assessment method uses the framework described in this section as a source of input for extracting the information required for assessment.

4. Usability assessment technique: SALUTA

What is Saluta? The Scenario based Architecture Level UsabiliTy Analysis method (SALUTA) comprises the following steps:

1) Determine the goal of the assessment.
2) Create usage profile.
3) Describe the software architecture.
4) Evaluate usage scenarios: determine the support for the scenarios.
5) Interpret the results: draw conclusions from the analysis results.

The steps are discussed and defined in detail in the following subsections:

4.1 Determine the goal of the assessment

The first step in the analysis method is to determine the type of results that will be delivered by its analysis. The following goals are distinguished:

- Predict the level of usability: give an accurate indication of the support of usability for an architecture.
- Risk assessment: detect usability issues for which the software architecture is inflexible.
- Software architecture selection: compare two candidate software architectures and select the optimal candidate which has the best support for usability

4.2 Create usage profile

Before an architecture can be assessed for its support of usability, first a way to describe the required usability is required. Preece [10] and Hix [11] suggest various techniques for the specification of usability. The way traditional techniques specify usability such as proposed by Preece and Hix are not suited for architectural assessment because of the following reasons:

- Very little is mentioned about usability requirements in scientific literature. In addition, real-life examples are rarely provided. Preece for example, presents much advice on usability requirements, but in a rather abstract setting without real-life examples. Traditionally usability specifications are rather defined in an abstract fashion and therefore not suited for architectural assessment.
- Traditionally usability requirements have been specified such that these can be verified for an implemented system. However, such requirements are largely useless in a forward engineering process. For example, we could say that a goal for a system is that it should be easy to learn, or that new users should require no more than 30 minutes instruction, however, requirements at such a level are hard to assess on an architectural level, because those can only be measured when the system is in use. Such statements are therefore useless for architectural assessment of usability.

A more suitable format (as argued in section 1.1) for specifying required usability for architectural assessment is by using scenarios. Scenario profiles are increasingly often used for the assessment of quality attributes during the architectural design of software systems [12]. A scenario profile describes the semantics of software quality factors such as maintainability or safety for a particular system. A usage profile is defined as: “a description of the semantics of usability for a particular system in terms of scenarios”. Scenario profiles are created using the following steps:

1) Identify the context of scenario profile (SP) generation.
2) Identify all scenario entities for usability
3) Create usage scenarios
4) Scenario elicitation

4.2.1 Identify the context of SP generation. A scenario profile can, basically, be defined in one of two contexts [13]: the Greenfield and the experienced context. If a scenario profile is defined in an organization using the technique for the first time, for a new system and no historical data is available about similar systems, the profile definition fully depends on the experience, skill and creativeness of the individuals defining the profile. The resulting scenario profile is the only input to the architecture assessment. The lack of alternative data sources in this case and the lack of knowledge about the representativeness of scenario profiles defined by individuals and groups, indicates that there is a need to increase our understanding of profiles in this situation. In the second situation, there is either an earlier release of the system or historical data of similar systems available. Since, in this case, empirical data can be collected, this data can be used as an additional input for the next prediction and a more accurate result is achieved.

4.2.2 Identify all scenario entities for usage scenarios. A scenario is defined as: "Scenarios refer to interactions between independent entities [14]. Entities for example can be stakeholders, the system (or possibly parts of it such as hardware, software, subsystems, objects) and the environment" There are different ways to interpret the definition of scenario. In object oriented design methods a scenario generally refers to use case scenarios; scenarios that describe system behavior. However it is also possible to use the definition above so it describes actions or sequence of actions that might occur in relation to the system. For example usage scenarios describe system behavior whereas change scenarios describe an action (a modification task) in relation to the system.

Entities as defined in the definition above play an essential role in defining scenarios. For example the entity stakeholders is taken into account because different stakeholders in the software lifecycle take different viewpoints when expressing their concerns about a software system. These viewpoints reflect the stakeholders' differing needs with respect to the software architecture [15]. Because of that the different needs of each stakeholder directly relate to the different needs concerning quality attributes and hence to different needs concerning its software architecture. The same argumentation holds for other entities such as the context in which the stakeholder operates or the hardware that impose requirements on the quality attributes.

Identification of all entities that influence a particular quality attribute is essential for defining a scenario for that quality attribute. For usage scenarios the following entities have been identified that define a usage scenario:

- The user (as a stakeholder)
- The context in which the user operates (as part of the environment)
- The tasks that a user can perform (as part of the system).

4.2.3. Create usage scenarios. The way scenarios are created or defined for a quality attribute largely depends on the entities that define the scenario for that particular quality attribute. For usability the following activities are defined:

1. Identify the users: A representative list of distinct users has to collected and defined. Examples: Novice users, expert users or system administrators.

2. Identify the tasks: The next step is identification and selection of distinct tasks. Most systems have a lot of different tasks; therefore a representative selection of these tasks that are distinct has to be made. For example a task could be: insert new customer in database.

3. Identify the context: The third step is determination of the unique contexts in which each user operates. Examples: helpdesk context or training environment.

4. Create attribute preference table: The attribute preference table (APT) is defined to relate a scenario to usability. Because a scenario consisting of a user, a task and a context only describes interaction we have defined a way to relate it to usability. To express the usability issues a user has while performing a task in a specific context the scenarios are related to our usability attributes as defined in [6]. To relate a scenario to usability we determine the usability attribute values for that particular scenario. For example for a novice user performing a task “insert order” in a “learning environment” the learnability attribute of usability may be important. The APT expresses the required usability for that scenario. An example of an APT can be found in Table 1 By defining the APT the required usability is quantified by stating the users’ preference concerning usability for that scenario. For each type of user, task and context, the user’s preferences concerning usability is determined.

There are various ways to determine quantitative values for the preference to usability. It can be done as part of requirements collection process: typical users or experts assign values, for example they assign values between 1 to 5 to each attribute for each task and context. The assigning of values can also be done as a post requirements process (during assessment), where an expert (or a team of experts) determine values for the usability preferences, the usability requirements that are collected during requirements analysis can then be used as an informative source for assigning the values.

4.2.4. Scenario selection. The attribute preference table that was created in combination with a descriptive list of users, tasks and contexts of operation can be used to summarize and describe the different scenarios that have been created. From this table, which holds all scenarios a scenario, profile is created by selecting scenarios that are representative. Scenario selection is the process of selecting those scenarios that are to be used in the assessment step of the analysis. Scenario selection results in a scenario profile which holds the set of relevant scenarios which will be evaluated. A scenario profile is a

Table 1: Example APT

Scenario attribute preference table						
User	**Task**	**Context**	**Learn-ability**	**Efficien-cy of use**	**Relia-bility**	**Satis-faction**
A	T1	C1	5	2	4	3
A	T2	C2	5	5	3	2
A	T3	C1	1	1	3	3
A	T4	C2	1	…	3	3
B	T1	C1	….		…	…
B	T1	C2				
B	T2	C1				
…	…	…				

set of scenarios that form the context for a quality requirement posed on the system. Different profiles may be defined depending on criteria for selecting the scenarios into the profile. The selection criteria influence the representativeness of the scenario profile, since in essence it is a kind of population sampling strategy. Two types of general scenario profiles have been identified:

- Complete scenario profile: "all scenarios that can potentially occur"
- Selected scenario profile: "a representative subset of the population of all possible scenarios

Scenarios may be assigned additional properties, such as an associated weight, priority or probability of occurrence within a certain time. The selection of usage scenarios also depends on the goal of the analysis, if the goal is to:

- Predict the level of a quality attribute: Select scenarios that have high probability of occurring.
- Risk assessment: select scenarios that expose those risks.
- Software architecture selection: select scenario that highlight differences

The process of identifying scenario entities, scenario creation and scenario selection are often combined performed in one process called scenario elicitation.

4.3 Describe the software architecture

The third step, architecture description, concerns the information about the software architecture that is needed to perform the analysis. Generally speaking, usability analysis requires architectural information that allows the analysis to evaluate the scenarios. The result of this step is a description of the provided usability. Information related to the architecture; for example, box and line diagrams or documented design decisions, may provide data about various quality attributes but since our interest lies in usability only the information that is related to usability is required. To achieve this, the information required is extracted using our framework described in section 3. Different types of assessment techniques have been defined depending on the amount of architectural information that is available for assessment or what information one is willing to acquire to get a more accurate result from the assessment. The following subsection discusses the different assessment types defined and the architectural information necessary to perform that type of assessment.

4.4 Evaluate Scenarios

Assessing an architecture for its support of a particular quality attribute basically comes down to a comparison between the required values of that particular quality attribute versus the provided value of that quality attribute. For usability assessment the required usability 'levels' are compared to the provided usability 'levels'. The levels are specified by scenarios. In section 2.2 a technique is discussed for capturing and describing the required usability using scenario profiles. For each scenario in the scenario profile the architecture is analyzed for its support of that scenario. The process that identifies the support for the scenarios is defined as architectural support analysis. Eventually the results from the analysis are summarized into an overall result. For example the number of supported scenarios versus the number scenario not supported. This number will be an indication of the support of the architecture for its support of usability. Three different types of assessment have been defined (as depicted in Figure):

- **Pattern based**
- **Design decision based**
- **Use case map based**

The framework is used to extract the architectural information required for each assessment technique.

4.4.1. Pattern based. By analyzing an architectural description of the system an expert assesses the architectures support of usability. The architecture designs present within the development are used as a source of input for this type of assessment. The architectural design can be a simple box and line diagram

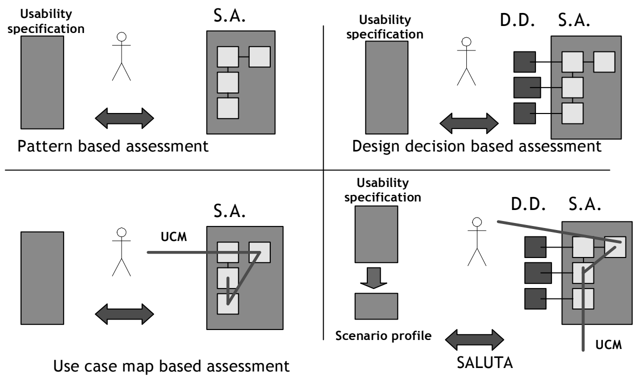

Figure 3: Scenario based assessment techniques

or for example a 4+1 view on the architecture. These designs can provide a lot of information about quality attributes and since the subject of our evaluation is usability we are only interested in those parts of architecture information that are related to usability. To acquire this information we use the framework to extract the required information. For expert based analysis an identification of patterns that influence usability in the system is required. By heuristically evaluating the system using the list of patterns identified in [6] a list of patterns or possible derivatives of those patterns implemented can be identified. The list usability patterns present in the software system should provide the information necessary for the software engineer to decide if a scenario will be supported by the architecture. For each scenario the software engineer will determine which patterns are involved and whether the usage scenario is sufficiently supported.

4.4.2. Design decision based. Not only a description of the structure of a system as it is decomposed into components and relations with its environment may be used for analysis. The design decisions that led to that particular architecture are also very important. The earliest design decisions may have a considerable influence on various quality attributes of the resulting system. However such design decisions which are made during design are most often not documented. If they have been however they may be used as a source of input for this type of assessment. For design decision based analysis it needs to be determined which design decisions have been made with regard to usability. By heuristically evaluating the design decisions made during design, using the list of usability properties defined in our framework the required information for the assessment (the design decisions that relate to usability) is extracted. This type of assessment heavily depends on the amount of information documented during or after initial architectural design. If no design decisions have been documented, this information could be retrieved by interviewing the system architect(s). For design decision based analysis, the list of design decisions that have been extracted using the framework is used to determine the support for each usage scenario. For each scenario we analyze if a scenario is affected by the design decisions and whether this has resulted in sufficient support for that scenario.

4.4.3. UCM based. An even more detailed way of assessing is to use use case maps (UCM) for describing the architecture. Using UCM for describing the architecture has the following benefits:

- Use case maps describe behavioral and structural aspects of systems at a high (architectural level) of abstraction
- UCM are easy to learn & understand but precise.
- Use case maps can show multiple scenarios in one diagram and the interaction amongst them. (which allows reasoning about a system as a whole)
- Use case maps are an informal abstract notation suited to our purposed

Architecture designs and design decisions made during design can be provided by the software architect who assists the analysis. Use case maps in case not present can be constructed with the assisting software architect. Based on the scenarios in the scenario profile for each scenario a use case map is build. Some tasks may have the similar or the same use case maps. The use case map allows us to analyze various static properties that relate to the usability attributes layer in our framework. For example a use case map may visualize the number of steps or time it takes to perform a task. The number of steps may be an indication to the efficiency or learnability attribute. Next to providing static information use case maps allow close analysis of architectural components (such as a patterns) involved in that particular scenario. The information gathered during this analysis is an extra source of input for the architectural support analysis of the scenarios.

4.4.4. Summarize: The types of assessment techniques presented here are complementary as shown in Figure . In general expert based assessment can be applied in most cases, assuming that at least some basic form of

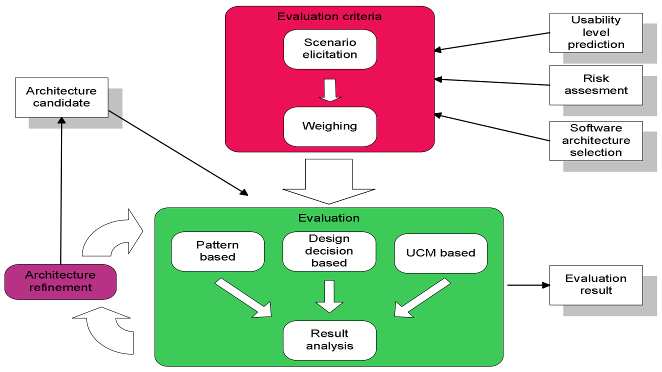

Figure 4: Assessment process

architectural description has been made for design which allows for identification of patterns. Design decision and use case map based assessment may give additional information for the architectural support analysis. However because these types of required information are not always present these can be retrieved or created by interviewing the system architects, which has its costs.

4.5 Interpret the results

When the scenario evaluation has been finished we need to interpret the results to draw our conclusions concerning the software architecture. At this stage we go back to our architecture design stage (see Figure) where we wondered if this architecture had sufficient support for usability. The interpretation of the results depends entirely on the goal of the analysis and the system requirements. If the architecture proves to have sufficient support for all quality attributes the design process is ended. Otherwise we need to apply architecture transformations or design decisions to improve certain quality attribute(s). The choice to use particular transformations may be based upon results from the analysis. For example: Consider a system, which proves to have a low support for usability, for example learnability for some usage scenarios is not supported. To improve learnability we could use the design primitive of guidance, to address guidance we could implement for example a wizard pattern or provide context sensitive help.

5. Conclusions

The work presented in this paper is motivated by the increasing realization in the software engineering community of the importance of software architecture for fulfilling quality requirements. We have presented a provisional assessment technique for usability based on scenarios, which has potential to improve current design for usability. Future case studies should determine the validity of our approach to refine it, possibly redefine and elaborate the steps that should be taken to make it generally applicable. Several issues need to be resolved during case studies, which have been summarized below:

- Relevance of framework: The relationships depicted in our framework indicate potential relationships. Further work is required to substantiate these relationships.
- Use case maps: may provide information about static properties of usability. More research is required to determine whether use case maps can provide that kind of information.

The main contribution of this paper is the formulation and derivation of an architectural assessment approach for usability.

6. References

[1] 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, pp. 81-90

[2] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson and J. Carriere, "The Architecture Tradeoff Analysis Method", *Proceedings of ICECCS'98*, 1998

[3] J. Bosch, *Design and Use of Software Architectures: Adopting and Evolving a Product Line Approach*, Pearson Education (Addison-Wesley and ACM Press).2000.

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

[5] P. O. Bengtsson; N. Lassing; J. Bosch and H. van Vliet ,"Architecture-Level Modifiability Analysis (ALMA),", *Conditionally Accepted for the Journal of Systems and Software*, 2002.

[6] E. Folmer and J. Bosch, "Usability patterns in Software Architecture", *Accepted for HCI International 2003*, 2003

[7] E. Folmer and J. Bosch ,"Architecting for usability; a survey", *Accepted for the Journal of systems and software*, 2002.

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

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

[10] J. Preece, Y. Rogers, H. Sharp, D. Benyon, S. Hollandand T. Carey, *Human-Computer Interaction*, Addison Wesley.1994.

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

[12] J. Bosch and P. O. Bengtsson, "Assessing optimal software architecture maintainability", *fifth European Conference on Software Maintainability and Reengineering*, 2002

[13] P. O. Bengtsson and J. Bosch ,"An Experiment on Creating Scenario Profiles for Software Change", *special issue on Software maintenance in Annals of Software Engineering (ISSN: 1022-7091)*, vol. 9 59-78, 2000.

[14] J. M. Caroll, 1995. The Scenario Perspective on System Development, in *Scenario Based Design: Envisioning Work and Technology in System Development*. Caroll, J. M., John Wiley and Sons.

[15] C. Gacek, A. Abd-Allah, B. Clark and B. Boehm, "On the Definition of Software System Architecture", 1995