An Architectural Decision Tool Based on Scenarios and Nonfunctional Requirements

Mr. Mahesh Parmar
Department of Computer Engineering
Lakshmi Narayan College of Tech.(LNCT).
Bhobal (MP), INDIA
Email:maheshparmarcse@gmail.com

Prof. W.U. Khan
Department of Computer Engineering
Shri G.S. Institute of Tech. & Science(SGSITS)
Indore (M.P.), INDIA
Email : wukhan@rediffmail.com

Dr. Binod Kumar
HOD & Associate Professor, MCA Department.
Lakshmi Narayan College of Tech.(LNCT).
Bhobal (MP), INDIA
Email : binod.istar.1970@gmail.com

***Abstract*—Software architecture design is often based on architects intuition and previous experience. Little methodological support is available, but there are still no effective solutions to guide the architectural design. The most difficult activity is the transformation from non-functional requirement specification into software architecture. To achieve above things proposed "An Architectural Decision Tool Based on Scenarios and Nonfunctional Requirements". In this proposed tool scenarios are first utilized to gather information from the user. Each scenario is created to have a positive or negative effect on a non-functional quality attribute. The non-functional quality attribute is then computed and compared to other non-quality attributes to relate to a set of design principle that are relevant to the system. Finally, the optimal architecture is selected by finding the compatibility of the design principle.**

Keywords- Software Architecture, Automated Design, Non-functional requirements, Design Principle.

I. INTRODUCTION

Software architecture is the very first step in the software lifecycle in which the nonfunctional requirements are addressed [7, 8]. The nonfunctional requirements (e.g., security) are the ones that are blamed for a system re-engineering, and they are orthogonal to system functionality [7]. Therefore, software architecture must be confined to a particular structure that best meets the quality of interest because the structure of a system plays a critical role in the process (i.e., strategies) and the product (i.e., notations) utilized to describe and provide the final solution.

In this paper, we discuss an architectural decision tool based on a software quality discussed in [14] in order to select the software architecture of a system. In [14], we proposed a method that attempted to bridge the chasm between the problem domain, namely requirement specifications, and the first phase in the solution domain, namely software architecture. The proposed method is a systematic approach based on the fact that the functionality of any software system can be met by all kinds of structures but the structure that also supports and embodies non-functional requirements (i.e., quality) is the one that best meets user needs. To this end, we have developed a method based on nonfunctional requirements of a system. The method applies a scenario-based approach. Scenarios are first utilized to gather information from the user. Each scenario is created to have a positive or negative affect on a non-functional quality attribute. When creating scenarios, we decided to start with some basic scenarios involving only single quality attribute, multiple scenarios were then mapped to each attribute that would have a positive or negative affect when the user found the scenario to be true. Finally, it became clear to us that we needed to allow each scenario to affect an attribute positively or negatively in varying degrees.

In this work, we have studied and classified architectural styles in terms of design principles, and a subset of nonfunctional requirements. These classifications, in turn, can be utilized to correlate between styles, design principles, and quality. Once we establish the relationship between, qualities, design principle, and styles, we should be able to establish the proper relationship between styles and qualities, and hence we should be able to select an architectural style for a given sets of requirements [8], [13].

II. NON-FUNCTIONAL REQUIREMENT

Developers of critical systems are responsible for identifying the requirements of the application, developing software that implements the requirements, and for allocating appropriate resources (processors and communication networks). It is not enough to merely satisfy functional requirements. Non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviours. This should be contrasted with functional requirements that define specific behaviour or functions. Functional requirements define what a system is supposed to do whereas non-functional requirements define how a system is supposed to be. Non-functional requirements are often called qualities of a system. Critical systems in general must satisfy non-functional requirement such as security, reliability, modifiability, performance, and other, similar requirements as well. Software quality is the degree to which software possesses a desired combination of attributes [15].

III. SCENARIOS

Scenarios are widely used in product line software engineering: abstract scenarios to capture behavioral requirements and quality-sensitive scenarios to specify architecturally significant quality attributes. Scenario system specific means translating it into concrete terms for the particular quality requirement. Thus, a scenario is "A request arrives for a change in functionality, and the change must be made at a particular time within the development process within a specified period." A system-specific version might be "A request arrives to add support for a new browser to a Web-based system, and the change must be made within two weeks." Furthermore, a single scenario may have many system-specific versions. The same system that has to support a new browser may also have to support a new media type. A quality attribute scenario is a quality-attribute-specific requirement

The assessment of a software quality using scenarios is done in these steps:

A. Define a Representative set of Scenarios

A set of scenarios is developed that concretizes the actual meaning of the attribute. For instance, the maintainability quality attribute may be specified by scenarios that capture typical changes in requirements, underlying hardware, etc.

B. Analyses the Architecture

Each individual scenario defines a context for the architecture. The performance of the architecture in that context for this quality attribute is assessed by analysis. Posing typical question [15] for the quality attributes can be helpful.

C. Summaries the Rresults

The results from each analysis of the architecture and scenario are then summarized into overall results, e.g., the number of accepted scenarios versus the number not accepted. We have proposed a set of six independent high-level non-functional characteristics, which are defined as a set of attributes of a software product by which its quality is described and evaluated. In practice, some influence could appear among the characteristics, however, they will be considered independent to simplify our presentation. The quality characteristics are used as the targets for validation (external quality) and verification (internal quality) at the various stages of development. They are refined (see Figure 1) into sub-characteristics, until the quality attribute are obtained. Sub characteristics (maturity, fault tolerance, confidentiality, changeability etc) are refined into scenarios. Each non-functional characteristic may have more than one sub characteristics is refined into set of scenarios. When we characterized a particular attribute then set of scenarios developed to describe it.

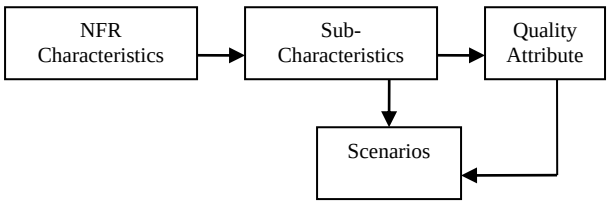

Figure 1. Analysis Scenario Diagram

IV. THE APPROACH

To establish the correct relationship between architectural styles using non functional requirements. The proposed recommendation tool consists of four activities as follows:

- Create a set of simple scenarios relevant to a single nonfunctional requirement.
- Identify those scenarios that may have positive or negative impacts on one or more nonfunctional requirements
- Establish a relationship between a set of quality attributes obtained in step 2 to a set of universally accepted design principles (tactics).
- Select a software architecture style that supports set of design principles identified by step 3.

A. Quality Attribute

Product considerations and market demands require expectations or qualities that must be fulfilled by a system's architecture. These expectations are normally have to do with how the system perform a set of tasks (i.e., quality) rather than what system do (i.e., functionality). Functionality of a system, which is the ability of a system to perform the work correctly for which it was intended, and the quality of a system, is orthogonal to one another.

In general, the quality attributes of a system is divided between two groups: 1) Operational quality attributes such as performance, and 2) non-operational, such as modifiability [8]. In this study, we have selected both operational and non-operational quality attributes as follows:

- Reliability (the extent with which we can expect a system to do what it is supposed to do at any given time)
- Security (the extend by which we can expect how secure the system is from tampering/ illegal access)
- Modifiability (how difficult or time consuming it is to perform change on the system)
- Performance (how fast the system will run, i.e., throughput, latency, number of clock cycles spend finishing a task)
- Usability (the ease by which the user can interact with system in order to accomplish a task),
- Availability (the extend by which we expect the system is up and running)
- Reusability (the extent by which apart or the entire system can be utilized)

Usability involves both architectural and nonarchitectural aspects of a system. Example of nonarchitectural features includes graphical user interface (GUI); examples of architectural features include, undo, cancel, and redo. Modifiability involves decomposition of system functionality and the programming techniques utilized within a component. In general, a system is modifiable if changes involve the minimum number of decomposed units. Performance involves the complexity of a system, which is the dependency (e.g., structure, control, and communication) among the elements of

a system, and the way system resources are scheduled and/or allocated. In general, the quality of a system can never be achieved in isolation. Therefore, the satisfaction of one quality may contribute (or contradict) to the satisfaction of another quality [12]. For example, consider security and availability; security strives for minimally while availability strives for maximally. Or, it is difficult to achieve a high secure system without compromising the availability of that system. In this case security contradicts the availability. This can be easily solved by negotiating with the user to make her/his mind. Another example has to do with security and usability: security inhabits usability because user must do additional things such as creating a password. Table I documents the correlation among quality attributes.

To summaries, five types of quality attributes relationships are identified. These relationships are defined by some numerical values which belong to 0 to 1. These relationships are: Very strong(0.9), Strong(0.7), Average(0.5), Below average(0.3), Very low(0.1), Not available(0.0).

TABLE I. QUALITY VS QUALITY

S.N.	Quality Attribute	Quality Attribute	Relationship	values
1)	Reliability	Performance	Very Strong	0.9
		Security	Very Strong	0.9
2)	Performance	Reliability	Very Strong	0.9
		Security	Below Average	0.3
3)	Security	Reliability	Average	0.5
		Performance	Very Low	0.1

B. Design Principle

According to [1, 2], a design can be evaluated in many ways using different criteria. The exact selection of criteria heavily depends on the domain of applications. In this work, we adopted what is known as commonly accepted design principles [3, 6, 7], and a set of design decisions known as tactics [3, 8, 13, 14]. Tactics are a set of proven design decisions and are orthogonal to particular software development methods. Tactics and design principle have been around for years and originally advocated by people like Parnas and Dijkstra. Our set of design principles and tactics includes: 1) Generality (or abstractions), 2)Locality and separation of concern, 3) Modularity, 4)Concurrency 5) Replicability, 6) Operability, and 7) Complexity.

Examples of design principles and tactics include a high degree of parallelism and asynchronized communication is needed in order to partially meet the performance requirement; a high degree of replicability (e.g., data, control, computation replicability) is needed in order to partially meet availability; a high degree of locality, modularity, and generality are needed in order to achieve modifiability and understandability; a high degree of controllability, such as authentication and authorization, is needed in order to achieve security and privacy.; and a high degree of locality, operatability (i.e., the efficiently by which a system can be utilized by end-users) is needed in order to achieve usability. Table II shows the correlation among qualities and tactics.

TABLE II. TACTICS VS QUALITIES

S.N	Tactics	Quality Attribute	Relationship	values
1)	Generality	Reliability	Very Strong	0.9
		Security	Average	0.5
		Performance	Very Strong	0.9
2)	Locality	Reliability	Very Strong	0.9
		Security	Not Available	0.0
		Performance	Very Strong	0.9
3)	Modularity	Reliability	Very Strong	0.9
		Security	Very Strong	0.9
		Performance	Strong	0.7

C. Architecture Styles

In order to extract the salient features of each style we have compiled its description, advantages and disadvantages. This information was later utilized to establish a link among styles and design principles. We have chosen, for the sake of this work, main/subroutine, object-oriented, pipe/filter, blackboard, client/server, and layered systems.

A main/subroutine (MS) architectural style advocates top-down design strategy by decomposing the system into components (calling units), and connectors (caller units). The coordination among the units is highly synchronized and interactions are done by parameters passing.

An Object-oriented (OO) system is described in terms of components (objects), and connectors (methods invocations) components are objects. Objects are responsible for their internal representation integrity. The coordination among the units is highly asynchronized and interactions are done by method invocations. The style supports reusability, usability, modifiability, and generality.

A Pipe/filter (P/F) style advocates bottom-up design strategy by decomposing a system in terms of filters (data transformation units) and pipes (data transfer mechanism). The coordination among the filters are asynchronized by transferring control upon the arrival of data at the input. Upstream filters typically have no control over this behavior.

A Client/server (C/S) system is decomposed into two sets of components (clients or masters), and (servers or slaves). The interactions among components are done by remote procedure calls (RPC) type of communication protocol. The coordination and control transformation among the units are highly synchronized.

A Blackboard (BKB) system is similar to a database system; it decomposes a system into components (storage and computational units known as knowledge sources (KSs). In a Blackboard system, the interaction among units is done by shared memory. The coordination among the units, for most parts, is asynchronized when there is no race for a particular data item, otherwise it is highly synchronized. The blackboard

style enjoys some level of replications such data (e.g., the distributed database and the distributed blackboard systems) and computation.

A Layered (LYR) system typically decomposes a system into a group of components (subtasks). The communication between layers is achieved by the protocols that define how the layers will interact. The coordination and control transformation among the units (or subtasks) is highly synchronized and interactions are done by parameters passing. A Layered system incurs performance penalty stems from the rigid chain of hierarchy among the layers. Table III illustrates the relationships among design principles/tactics.

V. PROPOSED WORK

The implementation of our tool consists of six different modules to perform its functions. Average weight module calculates average weight corresponding to selected scenarios. Effective weight module calculates effective weight of each nonfunctional requirement and each nonfunctional requirement has list of scenarios. Scenarios and its corresponding weight are selected by user. Quality attribute weight module calculates quality attribute weight. It depends on average and effective module response. Quality attribute rank module calculates the rank of quality attribute. It depends on quality attribute weight module. Tactics rank module calculates tactics rank and architecture style rank module calculates the architecture rank.

TABLE III. ARCHITECTURAL STYLES VS TACTICS

S.N	Architecture Style	Tactics	Relationship	values
1)	Pipe & Filter	Generality	Very Strong	0.9
		Locality	Very Strong	0.9
		Modularity	Average	0.5
2)	Black Board	Generality	Very Strong	0.9
		Locality	Very Strong	0.9
		Modularity	Very Strong	0.9
3)	Object Oriented	Generality	Very Strong	0.9
		Locality	Very Strong	0.9
		Modularity	Very Strong	0.9

These are described in detail as follows.

- Calculate average weight of each quality attribute that is selected by user. In this step user first selects scenarios corresponding to non-functional requirement and chooses weight according to his choice. Then calculate the average weight for each non functional requirement by using following formula

$$AQAi = \frac{\sum_{1}^{n} QWtn}{N}$$

AQA_i = Average weight of i^{th} quality attribute
QWt_n = Weight of n^{th} selected scenario.
n = Number of scenarios
N = Total number of selected scenarios

- Calculate effective weight of each quality attribute. Each scenario may affect more than one scenarios. All effective scenarios questions for each scenario are stored in the effect table in the database. Effect table maintains the list of effected scenarios questions. Calculate effective weight for each quality attribute by using following formula.

$$EWtQAi = \sum_{1}^{m}\sum_{1}^{e} EQn * QWtn$$

$EWtQA_i$ = Effective weight of i^{th} quality attribute
EQ_n = n^{th} Effective scenario.
e = Number of effective scenarios.
m = Number of scenarios

- Calculate quality attribute weight for each quality attribute. Using the output of step1 and step 2 we calculate the quality attribute weight by the following formula.

$$QAWti = AQAi + EWtQAi$$

$QAWt_i$ = i^{th} quality attribute weight.

- Calculate quality attribute rank. Quality to quality relationship table is stored in the database which maintains relationship values of quality to quality attribute. Calculate quality attribute rank using quality to quality relationship table by following formula.

$$QARi = \sum_{1}^{q} QAWti * QtoQq$$

QAR_i = i^{th} quality attribute rank.
q = Number of quality attribute.
$QtoQ_q$ = qth Quality to quality relationship

- Calculate tactics rank. Quality to tactics relationship table is stored in the database which maintains relationship values of quality to tactics. Calculate tactics rank using quality to tactics relationship table by following formula.

$$TRi = \sum_{1}^{t} QARi * QtoTt$$

TR_i = i^{th} Tactics rank.
$QtoT_t$ = t^{th} Quality to tactics relationship.
t = Number of tactics.

- Calculate architecture style rank. Tactics to architecture style relationship table is stored in the database which maintains relationship values of tactics to architecture style. Calculate architecture style rank using tactics to architecture style relationship table by flowing formula.

$$ASRi = \sum_{1}^{a} TRi * TtoASa$$

ASR_i = i^{th} Architecture style rank.
$TtoAS_a$ = ith Tactics to architecture style relationship.
a = Number of architecture styles

First user will click the main page then main page opens and he will select non functional requirements. Now he has to select scenario questions and corresponding weight according to his requirement. A single user can select more than one non functional requirement

Now the user will select the submit button and Result page will open and he would be able to see the Average Weight, Effective Weight, Quality Attribute Weight, Quality Attribute rank, Tactics Rank and Architectural Style and Rank.

VI. Related Work

The work in this paper is inspired by the original work in the area of architectural design guidance tool by Thomas Lane [4], and it is partially influenced by the research in [2, 5], [6], of [7], [8], [9], [11], [12], [13], and [14]. In [5], NFRs, such as accuracy, security, and performance have been utilized to study software systems.

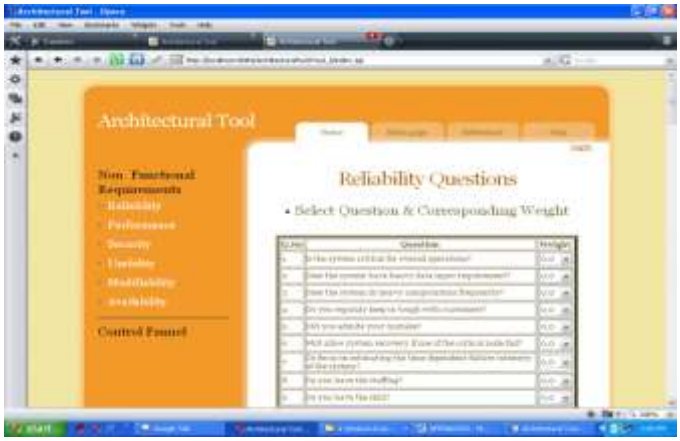

Figure 2 . A simple form-based Scenario

Figure 3. The evaluation results

In [6], the authors analyzed the architectural styles using modality, performance, and reusability. Their study provided preliminary support for the usefulness of architectural styles the work by Bass et al. [8] introduces the notion of design principle and scenarios that can be utilized to identify and implement quality characteristics of a system. In [7], the discussed the identification of the architecturally significant requirements its impact and role in assessing and recovering software architecture. In [9], the authors proposed an approach to elicit NFRs and provide a process by which software architecture to obtain the conceptual models.

In [14], the authors proposed a systematic method to extract architecturally significant requirements and the manner by which these requirements would be integrate into the conceptual representation of the system under development. The method worked with the computation, communication, and coordination aspects of a system to select the most optimal generic architecture. The selected architecture is then deemed as the starting point and hence is subjected to further assessment and/or refinement to meet all other user's expectations.

In [13], the authors developed a set of systematic approaches based on tactics that can be applied to select appropriate software architectures. More specifically, they developed a set of methods, namely, ATAM (architecture Tradeoff Analysis Method, SAAM (Software Architecture Analysis Method, and ARID (Active Reviews for Intermediate Designs. Our approach has been influenced by [13]; we did applied tactics and QAs to select an optimal architecture. However, the main differences between our approach and the methods developed by Clements *et al.* [13] are 1) our method utilizes different set of design principle and proven design, 2) establishes the correlation within QAs, tactics using tables, 3) establishes the proper correlation between QAs, tactics, and architectural styles using a set of tables and 4) the implementation of scenarios, which meant to increase the accuracy of the evaluation and architectural recommendations.

VII. Conclusions and Future Work

In this paper, we created a tool based on a set of scenarios that allows the user to select an architecture based on non-functional requirements. Non-functional requirements are then mapped to tactics using weighting. The architecture is then selected by its compatibility with the high-scoring design principle. We believe this approach has a lot of merits. However, more research work will be required to create a complete set of scenarios having a closer coupling with quality attributes. Additional work may also be required in fine-tuning the mappings between nonfunctional and functional requirements.

Currently, our tool can be utilized to derive and/or recommend architectural styles based on NFR. To validate the practicality and the usefulness of our approach, we plan to conduct a series of experiments in the form of case studies in which the actual architectural recommendations from our tool will be compared to the design recommendations by architects. We have discussed some quality attributes, some design tactics and some architecture styles. This needs some more research work on other quality attributes, tactics and architecture styles. New research works on non functional requirements might be

done by project members in the future. Our tool provides facility for addition, deletion and modification of new non-functional requirements, new tactics, and new architecture styles.

REFERENCES

[1] R. Aris. Mathematical modeling techniques. London; San Francisco: Pitman (Dover, New York), 1994.

[2] N. Medvidovic, P. Gruenbacher, A. Egyed, and B. Boehm. Proceedings of the 13th International Conference on Software Engineering and Knowledge Engineering (SEKE'01), Buenos Aires, Argentina, June 2001.

[3] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. Pattern-Oriented Software Architecture: A System of Patterns. John Wiley, 1996.

[4] T. Lane. User Interface Software Structure, Ph.D. thesis. Carnegie Mellon University, May 1990.

[5] L. Chung, and B. Nixon. Dealing with Non Functional Requirements: Three experimental Studies of a Processes oriented approach. Process dings of the International Conference on Software Engineering (ICSE'95), Seattle, USA, 1995.

[6] M. Shaw, and D. Garlan. Software Architecture: Perspective on an Emerging Discipline. Prentice Hall, 1996.

[7] M. Jazayeri, A. Ran, and F. Linden. Software Architecture for Product Families., Addison/Wesley, 2000.

[8] L. Bass, P. Clements, and R. Kazman. Software Architecture in Practice, second edition, Addison/Wesley. 2003.

[9] L. Cysneiros, and J. Leite. Nonfunctional Requirements: From elicitation to Conceptual Models. IEEE Transaction on Software Engineering, vol.30, no.5, May 2004.

[10] B. Boehm, A. Egyed, J. Kwan, D. Port, A. Shah, and R. Madachy. Using the WinWin spiral model: a case study IEEE Computer, 1998.

[11] A. Lamsweerde. From System Goals to Software Architecture. In Formal Methods for Software Architecture by LNCS 2804, Springer-Verlag, 2003.

[12] L. Chung, B. Nixon, E. Yu, and J. Mylopoulos. Nonfunctional Requirements in Software Engineering. Kluwer Academic, Boston, 2000.

[13] P. Clements, R. Kazman, and M. Klein. Evaluating Software Architectures: Methods and Case Studies. Addison Wesley, 2002.

[14] H. Reza, and E. Grant. Quality Oriented Software Architecture. The IEEE International Conference on Information Technology Coding and Computing (*ITCC'05*), Las Vegas, USA, April 2005.

[15] J.A. McCall, Quality Factors, *Software Engineering Encyclopedia*, Vol 2, J.J. Marciniak ed., Wiley, 1994, pp. 958 – 971

[16] B. Boehm and H. Hoh, "Identifying Quality-Requirement Conflicts," IEEE Software, pp. 25-36, Mar. 1996.

[17] M. C. Paulk, "The ARC Network: A case study," IEEE Software, vol. 2, pp. 61-69, May 1985.

[18] M. Chen and R. J. Norman, "A framework for integrated case," IEEE Software, vol. 9, pp. 18-22, March 1992.

[19] S. T. Albin, The Art of Software Architecture: Design Methods and Techniques, John Wiley and Sons, 2003

[20] P. Bengtsson, Architecture-Level Modifiability Analysis, Doctoral Dissertation Series No.2002-2, Blekinge Institute of Technology, 2002.

AUTHORS PROFILE

Mr. Mahesh Parmar is Assistant Professor in CSE Dept. in LNCT Bhopal and having 2 years of Academic and Professional experience. He has published 5 papers in International Journals and Conferences. He received M.E. degree in Computer Engineering from SGSITS Indore in July 2010. His other qualifications are B.E.(Computer Science and Engineering, 2006). His area of expertise is Software Architecture and Software Engineering

Dr.W.U.Khan, has done PhD (Computer Engg) and Post Doctorate (Computer Engg). He is Professor in Computer Engineering Department at, Shri G.S. Institute of Technology and Science, Indore, India.

Dr. Binod Kumar is HOD and Associate professor in MCA Dept. in LNCT Bhopal and having 12.5 years of Academic and Professional experience. He is Editorial Board Member and Technical Reviewer of Seven (07) International Journals in Computer Science. He has published 11 papers in International and National Journals. He received Ph.D degree in Computer Science from Saurastra Univ. in June 2010. His other qualifications are M.Phil (Computer Sc, 2006), MCA(1998) and M.Sc (1995). His area of expertise is Data Mining, Bioinformatics and Software Engineering.