Asian Journal of Information Technology 7 (3): 126-129, 2008
ISSN: 1682-3915
© Medwell Journals, 2008

Impact on Quality Attributes for Evaluating Software Architecture Using ATAM and Design Patterns

[1]N. Sankar Ram, [1]B. Rajalakshmi and [2]Paul Rodrigues
[1]Department of CSE, Velammal Engineering College, Chennai, Pin code-600 066, Tamilnadu, India
[2]Department of CSE, A K College of Engineering, Krishnankoil, Pin code-626 190, Tamilnadu, India

Abstract: Architecture evaluation for a large system can be done by using an approach called Architecture Tradeoff Analysis Method (ATAM). In large system the achievement of quality attributes such as maintainability, reusability, extensibility, scalability and Stake Holders Expects (SHE) are not full filled in ATAM approach. This study, presents a system which uses ATAM and design patterns for evaluating the software architecture to identify risk factor and check all the quality attributes have been addressed in the software. By combining the design patterns and the ATAM approach for the evaluation of the software architecture would result in better solutions.

Key words: Software architecture, design patterns, quality attributes, risk factor, ATAM, SHE

INTRODUCTION

The major issue in software development today is quality. The idea predicting the quality of software from a higher level design description is not a new one. Quality of software is bound by basis of its architecture (Kruchten *et al*., 2006). It is recognized that it is not possible to measure the quality attributes of the final system based on SA design. This would imply that detailed design and implementation represents a strict projection of architecture. We analyze methods looking for:

- Their progress towards refinement over time.
- Their main contribution.
- Advantages obtained by them.

Software architecture of a system is defined as "the structure of structures of the system, which comprise software components, the externally visible properties of those components and the relationship among them" (Bass *et al*., 2003; Kruchten *et al*., 2006).

ATAM is a method for evaluating architecture-level designs and identifies trade-off points between attributes, facilities communication between stakeholders (such as user, developer, customer, maintainer) from the perspective of each attribute, clarifies and refines requirements and provides a framework for ongoing, concurrent process of system and analysis.

We could find that ATAM is a risk identification mechanism of quality achievement. Normally, ATAM does not discuss with all possible quality attributes. Efficiency of ATAM depends on the expertise and potential of Stakeholders (SH) and quality attributes (Clements *et al*., 2002; Shaw and Garlan, 2004; Dobrica and Niemela, 2002; Cortellessa *et al*., 2007).

RESEARCH BACKGROUND

The Architecture Trade-off Analysis Method (ATAM): Architecture-based analysis techniques fall into one of two categories, questioning and measuring according to whether they offer qualitative or quantitative results. In complex design situations the effort required to develop models suitable for quantitative analysis and the concentration on one quality at the expense of others tend to dissuade the use of measuring techniques

The adoption of an iterative incremental development process required a method, which could be used throughout the systems lifecycle, as well as provide insight into the design issues and how they relate to the customer objectives (Dobria and Niemela, 2002; Kruchten *et al*., 2006). Consequently, the methods suited to such an approach are those oriented towards application from an early point in the design life-cycle as well as providing the ability to analyze the relationship between multiple quality concerns and design decisions. The only methods found to satisfy these conditions

Corresponding Author: Professor N. Sankar Ram, Department of CSE, Velammal Engineering College, Chennai, Pin code-600 066 Tamilnadu, India

included Software Architecture Assessment using Bayesian Networks (SAABNet) and the Architecture Tradeoff Analysis Method (ATAM) (Dobria and Niemela, 2002; Kruchten *et al*., 2006).

Quality attributes: A quality attribute is a nonfunctional characteristic of a component or a system. A software quality represents the degree to which software possesses a desired combination of attributes. According to this, there are 6 categories of characteristic (functionality, reliability, usability, efficiency, main tainability and portability), which are divided into sub characteristic (Kazman *et al*., 1994; Kruchten *et al*., 2006). The quality attributes are defined as:

Expendability: The degree to which architectural, data or procedural design can be extended.

Simplicity: The degree to which a program can be understood without difficulty.

Generality: The degree to which a software product can perform a wide range of functions.

Modularity: The degree to which the implementation of functions in a program are independent from one another.

Modularity at runtime: The degree to which functions of a program are independent from one another at runtime.

Learnability: The degree to which the code source of a program is easy to learn by new developers.

Understandability: The degree to which the code source of a program is easy to understand.

Reusability: Reusability here is the degree to which a piece of design (or a subset of apiece of design) can be reused in another design.

Scalability: Scalability is the ease with which an application or component can be modified to expand its existing capacities at runtime.

Robustness: The degree to which an executable program continues to function properly under abnormal conditions or circumstances.

PATTERNS

Each pattern describes a problem which occurs over and over again in our environment and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice. An emerging idea in system development under the process can be improved significantly if the system can be analyzed, designed and build from prefabricated and predefined system components.

Originally, patterns addressed common idioms in the world of object oriented design and implementation. Patterns can be used in other areas; in particular *analysis patterns* are being used to describe common idioms at the business analysis level.

Generative patterns: These patterns in our minds are, more or less, mental images of the patterns in the world: they are abstract representations of the very morphological rules, which define the patterns in the world. However, in one aspect they are very different. The patterns in the world merely exist. But the same patterns in our minds are dynamic. They have force. They are generative. They tell us what to do; they tell us how we shall, or may, generate them; and they tell us too, that under certain circumstances, we must create them. Each pattern is a rule, which describes what you have to do to generate the entity which it defines.

Non generative patterns: These are static and passive. They describe recurring phenomena without necessarily saying how to reproduce them.

Components

Name	:	It must have a meaningful name.
Problem	:	A statement of the problem which describes its intent: the goals and objectives it wants to reach within the given context and forces.
Context	:	The preconditions under which the pattern is applicable.
Forces	:	A description of the relevant forces and constraints and how they interact/conflict with one another.
Solution	:	Static relationships and dynamic rules describing how to realize the desired outcome.
Examples	:	One or more sample applications of the pattern.
Resulting Context	:	It describes the post conditions and side-effects of the pattern.
Rationale	:	A justifying explanation of steps or rules in the pattern.
Related Patterns	:	The static and dynamic relationships between this pattern and others.
Known Uses	:	Describes known occurrences of the pattern and its application.

Design patterns: In software engineering, a design pattern is a general repeatable solution to a commonly occurring problem in software design. A design pattern is not a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations (Richard, 1996; Beck, 2007; Freeman *et al.*, 2004). Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved. Algorithms are not thought of as design patterns, since they solve computational problems rather than design problems.

In addition, patterns allow developers to communicate using well-known, well understood names for software interactions. Common design patterns can be improved over time, making them more robust than ad-hoc designs. Considering situations where patterns are used appropriately in a program to solve their corresponding design problems and assuming that the developers have a good knowledge of design patterns.

APPLYING PROPOSED METHOD FOR EVALUATING ARCHITECTURE

Build a search engine for a travel company, which allows searching the resorts, fitting in the required criteria and available for booking (Table 1).

Functional requirements:

- User enters search criteria (multiple resorts IDs, date range, multiple resort amenities, multiple region hierarchy (region/sub-region/market) on the screen and submits the same.
- User defines the result sort order.
- System support multiple levels of validations.
    - User's access control to execute the search.
    - Validations of the search criteria.
- System executes the search criteria.
- System returns the results in the required sort order.
- User can drill down on the returned results and get more details.

Non-functional requirements

- Performance of the system is a critical aspect - with 5 million records in the database, the search should take a maximum of 5 sec.
- Recommended technology is J2EE using Weblogic. Any COTS component can be suggested with proper justification.

Table 1: Sample scorecard for evaluating software architecture features

Requirement	Weight	Score	Total
Book a tour	5	5	25
Search for a tourist spot and routes from South to the spot	5	4	20
Change screen layout to suit the PDA screens	3	3	9
Portability	4	3	12
Maintainability	3	3	9
Reusability	4	2	8
Extensibility	3	3	9
Scalability	4	2	8
Total			100

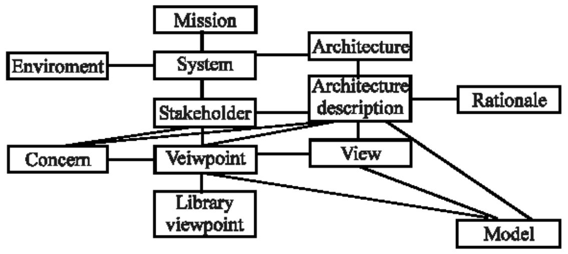

Fig. 1: Architectural description of software intensive system

Expectations: The stress while defining the architecture should be on business components and database rather than presentations. The architecture should identify the following:

- Various layers in the architecture.
- Various business components in each layer.
- Responsibility of the component.
- Data transfer across layers.
- Considerations for the performance for each of the above, as well as database.

Techniques for evaluation and review: Review techniques can be divided into 2 fundamental types:

- Questioning techniques.
- Measuring Techniques.

The Questioning techniques are relatively simple and include methods like a general discussion based qualitative questions, evaluating through questionnaire(s), having a predefined checklist and using it for review or using some well structured methods such as scenario based evaluation methods.

The various measuring techniques are more technical in nature and usually involve quantitative methods like collecting quantitative answers to specific questions, collecting metrics and through simulation and prototypes.

Figure 1 represents architectural description, by applying our method, the typical issues that are under investigations are as follows:

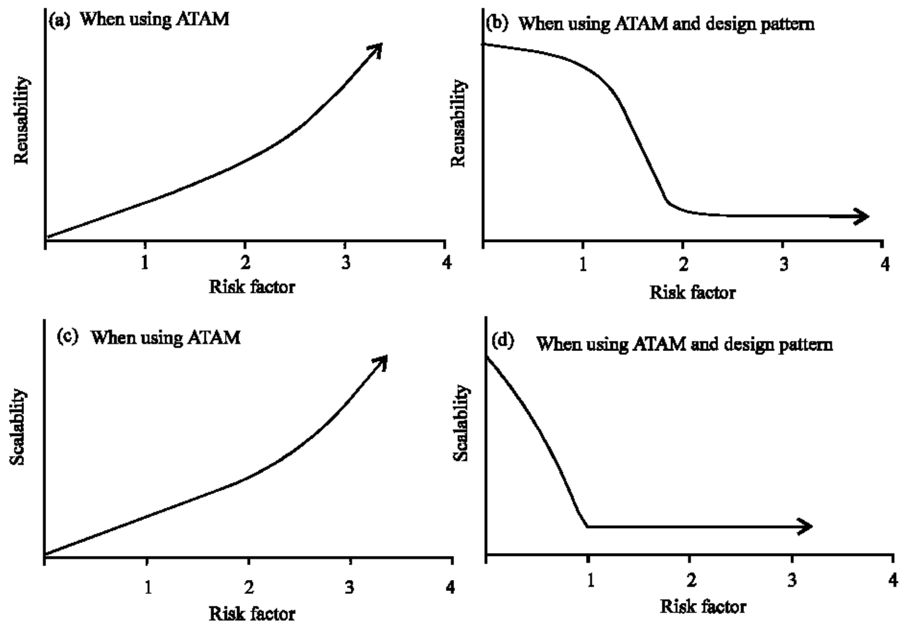

Fig. 2: Performance after applying ATAM amd design patterns

- Are all stakeholders considered?
- Have all requirements been identified?
- Is the architectural solution being provided appears rationale?
- Are the documents well managed?
- Is the architectural solution being provided appears rationale?

The following Fig. 2(a-d) shows the performance after applying ATAM and Design Patterns.

CONCLUSION

The ATAM is the robust method for evaluating software architectures. It works by having these stakeholders articulate a precise list of quality attribute requirements in the form of patterns and scenarios and by illuminating the architecture with respect to our design patterns. ATAM has proven itself as a useful tool hence we use the ATAM architecture to integrate the above mentioned design patterns for better evaluation.

REFERENCES

Bass, L. and P. Clements *et al.*, 2003. Software architecture in practice. 2nd Edn. SEI series in software engineering. Boston: Addison-Wesley.

Beck, K., 2007. Implementation Patterns. Pearson Education, Proceedings of the 18th International Conference on Software Engineering.

Clements, P. and R. Kazman *et al.*, 2002. Evaluating software architectures: Methods and case studies. SEI series in software engineering. Boston: Addison-Wesley.

Cortellessa, V. and P. Pierini *et al.*, 2007. Integrating software models and platform models for Performance Analysis. IEEE. Trans. Software Eng., 33 (6).

Dobrica, L. and E. Niemela, 2002. A Survey on Software Architecture Analysis Methods. IEEE. Trans. Software Eng., 28 (7).

Freeman, Eric and Elisabeth Freeman *et al.*, 2004. Head First Design Patterns. O'Reilly Media.

Kazman, R. and M. Klein *et al.*, 2000. ATAM: Method for Architecture Evaluation. Software Engineering Institute: Pittsburgh.

Kruchten, P. and H. Obbink *et al.*, 2006. The Past, Present and Future of Software Architecture. IEEE. Software.

Richard, G. 1996. Patterns of Software: Tales From the Software Community. Oxford University.

Shaw, M. and D. Garlan, 2004. Software Architecture. Perspectives on an Emerging Discipline. Prentice Hall, India.

Shaw, M. and P. Clements, 2006. The Golden Age of Software Architecture. IEEE Software.