Innovative Patterns for Finding Enchanced Solutions to your Architecture

N. SANKAR RAM [1], PAUL RODRIGUES [2]
[1] Velammal Engineering College, Chennai – 600 066.

[2] Anna University, Chennai – 600 025

ABSTRACT

The paper discuss the innovative patterns such as subtraction, multiplication, division, task unification and attribute dependency change for evaluating the software architecture to identify the risk factor, check all the quality attributes have been addressed in the software. Architecture evaluation for a system can be done by using an approach called Architecture Tradeoff Analysis Method (ATAM). The achievement of quality attributes such as maintainability, reusability, extensibility, scalability and Stake Holders Expects (SHE) are not fulfilled in ATAM approach. We have proposed a method called Enhanced Architecture Tradeoff Analysis Method (EATAM) by combining the innovative patterns and the ATAM for the evaluation of the software architecture would result in better solutions. The innovative patterns are therefore useful not for categorizing new software ideas but also for generating them.

KEYWORDS:

Software Architecture, Innovative Patterns, Quality Attributes, Risk Factor, ATAM, EATAM, SHE.

1. INTRODUCTION

Most ideas for new patterns are either uninspired or impractical. A systematic process based on five simple patterns, can generate ideas that are both ingenious and viable. 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. It is recognized that it is not possible to measure the quality attributes of the final system based on software architecture design. This would imply that detailed design and implementation represents a strict projection of architecture.

Analyzing the software 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".

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. The modules or templates are therefore useful not just for categorizing new pattern ideas but also for generating them.

2. RESEARCH BACKGROUND

2.1 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. 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 included Software Architecture Assessment using Bayesian Networks (SAABNet) and the Architecture Tradeoff Analysis Method (ATAM)

2.2 Quality Attributes

A quality attribute is a nonfunctional characteristic of a component or a system. A software

Manuscript received July 5, 2008
Manuscript revised July 20, 2008

quality represents the degree to which software possesses a desired combination of attributes. According to this, there are six categories of characteristic (functionality, reliability, usability, efficiency, maintainability, and portability), which are divided into sub characteristic [10, 5].
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.
Learn ability: 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.

3. INNOVATIVE PATTERNS

At the core of our process are the five innovation patterns. These "templates of innovation" have emerged from our historical analysis of product development trends. Our research indicates that more successful product innovation fit into at least one of these five patterns. Indeed, we have found that the patterns can help predict the emergence of new products before the appearance of signals indicating market demand.

3.1 Subtraction Pattern

While introducing new patterns, the marketers tend to eliminate the complexities in the old version thereby adding some interactive and innovative Add-on features that would enhance its performance and at the same time satisfying the customer needs better . The subtraction pattern outweighs the former by removing some of the unwanted components and replacing it by a better component in the "closed world" of the pattern and its immediate environment.

Everyone would have browsed the Job Portal. Users found it difficult because each and every detail of the resume had to be typed. Due to this there was wastage of time and unwanted errors occurred. Now these demerits were analyzed and a better replacement was made by new website which is used currently. Here applicants can upload their resumes directly in a jiffy instead of wasting their time in typing the resume. The subtraction patterns got a tremendous response and satisfied the needs of the customers. While this is a perfectly logical approach, it can result in those incremental improvements that have an impact on customers.

3.2 Multiplication Pattern

The second pattern represents a very different approach to innovation. This is the prime logic behind multiplication pattern: here the existing components or features are untouched, but another copy of these features are made. The objective is to go beyond a mere quantitative change and achieve a qualitative change.

Google search engine serves to be a classic example for this. Initially it had just a simple search engine, where the users used to while away their time in searching for information. Now the search engine contains a new feature called "Advanced Search" where one can filter his/her query and reduce the time by specifying appropriate fields such as date, file type, range etc. The user can find what he is looking for by this method.

3.3 Division Pattern

One can use the division pattern to split an existing product into many component modules. There is a change in the perspective which may lead to the reconfiguration of those modules in an unanticipated way – or even keep the modules separate in a manner that offers unexpected yield. The specialty of division is that each module preserves the characteristics of the whole.

Yahoo known for its wide range of usage all over the globe had all the utilities integrated into a single domain. Those utilities were yahoo messenger, search engine, mail, sports, and movies. Now they have been separated into individual modules. The main advantage of this method is that even if the "yahoo.com" is down, the user can still browse the various areas using their individual modules.

3.4 Task Unification Pattern

According to this method one can understand pattern innovation by assigning a new task to an existing product or its constituent environment, thereby fusing two tasks in a single component. The basic rationale for this bundling of tasks: if a single component is sufficient for

performing the task of the pattern and its environment, why not just see whether it can be made to do double duty.

A classic example for task unification would be Borland C which used to compile only 'C 'Programs. The newer version Turbo C contains the files required for both C and C++, so that both C and C++ codes can be compiled in the same environment. An even more specific example would be Microsoft VC++, which can execute all codes that run on a common platform. By creating patterns which provide double benefits, a huge customer base is created and an incredible level of innovation is achieved.

3.5 Attribute dependency change Pattern

This pattern mainly involves dependent relationship between attributes of a product and attributes of its immediate environment. Pattern can be made more adaptable to the given environment. One can also create dependencies that exist between two unrelated attributes of a single pattern. The attribute dependency pattern often generates what later seem like inevitable patterns.

Windows Media Player is mainly used for playing audio and video files. This media player identifies the file format and plays the file accordingly. That is if the extension is .mp3 then it identifies that it is a audio file and plays the file in the audio format. If the extension is .avi then it identifies that it is a video file and plays it in the video format. In this way it adapts to the given environment and satisfies the needs of the customer.

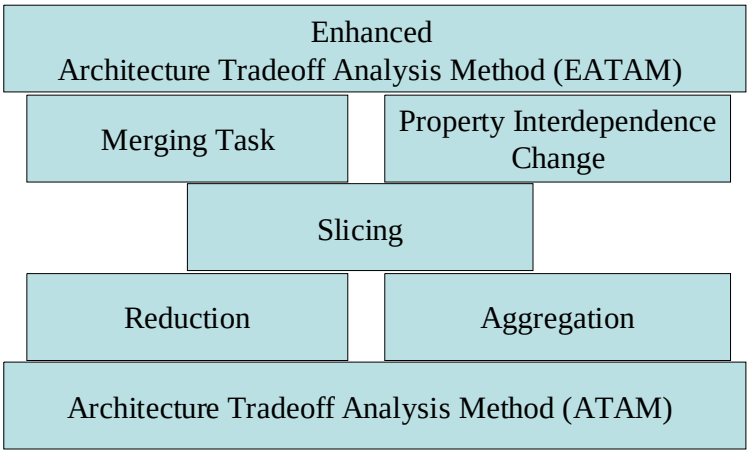


Figure 1 Functional diagram of EATAM

4. IMPLEMENTATION

The following steps describes the a simple search in a program without applying innovative pattern

1. The Program for search a number between 1 and 30. The numbers are actually coded in random
2. The user needs a key a number between 1 and 30. If the number found in first few search well and good or if the number founds in 29th search it will be worst case.

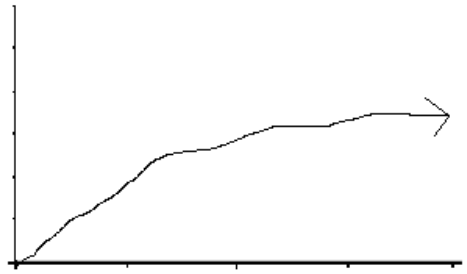
After applying Division Pattern

1. User needs to enter the value
2. Then program mood the number and finds whether the number is odd / even.
3. If the number is odd then program transfers the control to odd number series to check the number whether it presents in the given in list. The same procedure will be followed for even number
4. Now the worst case is the number will be found in 15th search.

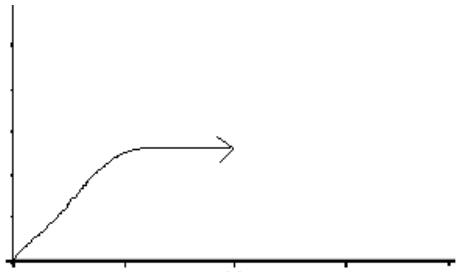
Best case / Worst case calculated using the following formula
(Number of Search actually encountered by program / Total number of search case in program) * 100

Example
Number of Search actually encountered by program = 15
Total number of search case in program = 30
MOOD = (15 / 30) * 100 = 50.00 %

The value of MOOD is less than 50 % means that good case else worst case



Worst Case Search
Without Applying Division Pattern



Best Case Search
With Applying Division Pattern

5. 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 innovative patterns for better evaluation. We have heard some pattern developers initially complain that imposing these patterns seems to take the fun out of their work. But the process, by forcing developers to follow a certain path, can actually make the creative challenges more interesting. We would like to emphasize that the process we have described isn't meant to replace all of the companies' pattern development methods. The method we have suggested focuses on the components- that are essential, that can be reshuffled, removed or replicated in new ways thereby enhancing the discipline of the pattern that is vital to guide the company to a sweet spot. EATAM really fits its customer needs.

REFERENCE

[1] N.Sankar Ram and Dr. Paul Rodrigues "Intelligent Risk Prophecy Using More Quality Attributes Injected ATAM and Design Patterns", 7th WSEAS Int, Conf. on Software Engineering, Parallel and Distributed Systems(SEPADS '08) University of Cambridge, UK, Feb 20-22, 2008

[2] L. Bass., P. Clements, and R. Kazman, *Software architecture in practice*. 2nd ed. SEI series in software engineering. 2003, Boston: Addison-Wesley.

[3] Clements, P., R. Kazman, and M. Klein, *Evaluating software architectures : methods and case studies*. SEI series in software engineering. 2002, Boston: Addison- Wesley.

[4] M. Shaw and D. Garlan, *Software Architecture.* Perspectives on an Emerging Discipline, Prentice – Hall, India.

[5] D. Colquitt and J. Leaney "*Expanding the view on Complexity within Architecture Trade-off Analysis Method*" Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems(ECBS'07)

[6] L. Dobrica and E. Niemela , "*A Survey On Software Architecture Analysis Methods*", IEEE *Trans. Software Eng*., vol. 28, no.7, July 2002.

[7] P. Kruchten, H. Obbink and J. Stafford, "*The Past, Present, and Future of Software Architecture*" IEEE Software March/April 2006.

[8] V. Cortellessa, P. Pierini and D. Rossi "*Integrating software models and platform models for Performance Analysis*" IEEE *Trans. Software Eng*., vol. 33, no.6, June 2007.

[9] M.A. Jalil and S.A. Mohamed Noah, "*The Difficulties of Using Design Patterns among Novices: An Exploratory Study*", Proceedings of the 5th IEEE International Conference on Computational Science and Applications

[10] Mary Shaw and Paul Clements, "*The Golden Age of Software Architecture*" IEEE Software March/April 2006

[11] Kazman, R., M. Klein, and P. Clements, *ATAM: Method for Architecture Evaluation.* 2000, Software Engineering Institute: Pittsburgh.

[12] Rick Kazman, Len Bass, Gregory Abowd and Mike Webb "*SAAM: A Method for Analyzing the Properties of Software Architectures"* IEEE 1994

[13] Gabriel, Richard (1996). Patterns of Software: Tales From The Software Community. Oxford University.

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

[15] Freeman, Eric; Elisabeth Freeman, Kathy Sierra, and Bert Bates (2004). "Head First Design Patterns". O'Reilly Media.

PROF. N. SANKAR RAM received the Master of Engineering in Computer Science from Madurai Kamaraj University, India, in 1997 and currently pursing his PhD degree in Anna University. He is an professor and head in the department of computer science and engineering at Velammal Engineering College, Chennai, India. His research interest includes software analysis, design and software architecture. He has published several journals and conference publications.

Dr. PAUL RODRIGUES received the Master of Engineering in Computer Science and PhD degree from Pondicherry University. He is an professor in the department of computer science and engineering at A.K College of Engineering, Krishnankoil India. He has more than 20 years of experience in both teaching, industries and guiding many research scholars. He has published several journals and conference publications.