Enhanced Intelligent Risk Divination Using Added Quality Attributes Injected ATAM and Design Patterns

N.Sankar Ram sankarramphd@yahoo.com
Professor/Computer Science and Engineering
Anna university
Chennai, 600 066, India

Dr.Paul Rodrigues
Professor/ Computer Science and Engineering
AK college of Engg
Chennai, India

Abstract

Architectural Tradeoff Analysis Method is a method for evaluation of architecture-level designs and identifies trade-off points between attributes, facilitates communication between stakeholders. ATAM has got the limitations like not a predictor of quality achievement, not deals more quality attributes, Efficiency always depends on the expertise and potential of stakeholders. In this paper we have proposed a system which uses ATAM to predict the risk analysis, with more possible quality attributes. We have used artificial intelligence to predict the risk of the SA based on the Knowledge base of the Stakeholder Experts.

Keywords: Architectural Tradeoff Analysis Method (ATAM), Stakeholders (SH), Stakeholder Experts (SHE), Design pattern (DP), Software Architecture(SA), Risk factor (RF), Radial Basis Function Neural Network (RBF-NN).

1. INTRODUCTION

Quality attributes of bulky software systems are primarily determined by the system's software architecture. The software architecture of a software-intensive system seriously decides system quality. The ability to appraise software architectures earlier than they are realized in finished systems can considerably reduce the risk that the delivered systems will not meet their quality goals. For architecture evaluation, the Carnegie Mellon® Software Engineering Institute (SEI) developed the Architectural Tradeoff Analysis Method (ATAM) [7]. There are more research going on ATAM limitations and improvements [1, 2, 3, 4, 5, 6].

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

In Our Previos Paper [29], we have discussed, we could find that ATAM is a risk identification mechanism not a predictor [8] 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). Here we have extended our previous work[29]. The same new method will predict the risk of software architecture with additional possible quality attributes and with the knowledgebase collected from historical data or SHE.

The rest of the paper discusses our approach and is organized as follows. Section 2 discusses the related work in the Architecture Tradeoff Analysis Method. Section 3 briefly explains the proposed system. Design Patterns and software architecture are explained in section 4. Section 5 describes the quality attributes. The survey on QA and DP are given in section 6.Section 7 presents the training and testing using RBF-NN and Section 8 concludes the paper.

2. RELATED WORK

Rick Kazman et al. [1] have proposed the Architecture Tradeoff Analysis Method (ATAM), a structured technique for understanding the tradeoffs inherent in the architectures of software intensive systems.

Robert L et al. [2] integrate the SEI ATAM and SEI CBAM. They build on their success in developing and piloting a collection of software architecture methods, they are focusing on integrating them, and building the bridges between them and the processes and architecture efforts outside the SEI, all the while continuing to refine existing methods and models. Paul Clements et al. [3] deal with the application of the SEI Architecture Tradeoff Analysis Method (ATAM) to the U.S. Army's Warfighter Information Network-Tactical (WIN-T) system and present the WIN-T program context, the definition of software architecture, and the background of the WIN-T organization and system being evaluated.

Siv Hilde et al. [4] have proposed the integrated security verification and security solution design trade-off analysis (SVDT) approach. SVDT is useful when there is a diverse set of requirements imposed upon a security critical system, such as a required security level, time-to-market and budget constraints and end users' expectations. Iain Bate et al. [5] have proposed the nine-step process of the Architecture Trade-Off Analysis Method (ATAM) and they address how the system's objectives should be decomposed and designed into components (i.e. the location and nature of interfaces); and what functionality the components should provide to achieve the system's objectives.

Mildred N et al. [6] have proposed the Architectural Tradeoff Analysis Method (ATAM) to evaluate two architectures and the goal of their evaluation is to determine which of the two architectures better provide the required services of the system. Their paper presents a detailed analysis of the different phases of ATAM on the two architectures. Erich Gamma et al. [9] have proposed to capture design experience in a form that people can use effectively. To that end they have documented some of the most important design patterns and present them as a catalog and the purpose of their book is to record experience in designing object-oriented software as design patterns. Each design pattern systematically names, explains, and evaluates an important and recurring design in object-oriented systems.

Atchara Mahaweerawat et al. [15] have proposed a new approach for predicting and classification of faults in object-oriented software systems. In particular, faults due to the use of inheritance and polymorphism are considered as they account for significant portion of faults in object-oriented systems. Their proposed fault prediction model is based on supervised learning using Multilayer Perception Neural Network. The results of fault prediction are analyzed in terms of classification correctness and some other standard criteria. Stefan Biffl et al. [16] have given the ideas for QATAM, a technique for the evaluation of QA strategies and their tradeoffs in the context of a software process and they illustrate the application of QATAM in an ongoing research project Lifecycle, which aim at improving evidence-based application of QA activities in SMEs.

Femi G et al. [17] have discussed the main tenets of the extended method, and illustrated its use through a small case study and they extend the popular ATAM (Architecture Tradeoff Analysis Method) method into a holistic approach that analyzes the quality attribute tradeoffs not only for the product line architecture, but for the individual product architectures as well. Ahmed BinSubaih et al. [18] have proposed ATAM to the test on their architecture and discuss the findings based on the outputs generated which include lists of risks, nonrisks, sensitivities, and tradeoffs made. They have developed to aid game portability between game engines and they present an Architecture Reactive View (ARV) to consolidate disparate outputs generated by ATAM into one which they consider as an improvement to ATAM. [19] [20] [21] are also the related work

3. PROPOSED SYSTEM

We underwent the detailed survey and finally arrived a more attributes which is more suitable for better performance of trade-off analysis .We have also introduced design pattern based software architecture analysis and it’s relationship with quality attributes.

Then we have chosen artificial intelligence to predict the risk of the SA. The impact of quality attributes on design pattern (IQADP) is analyzed. This input and the RF arrived using ATAM can either be achieved from the historical data or the result of a survey with the SHE. This IQADP is trained along with the RF. During testing only IQADP is given as input to get the RF.

4. DESIGN PATTERN (DP) AND SOFTWARE ARCHITECTURE (SA)

In software engineering, a design pattern [9, 10] is a common replicable solution to a frequently happening problem in software design. A design pattern is not a completed design that can be transformed straight away into code. It is a depiction or template for how to resolve a problem that can be used in many different situations. Object-oriented design patterns characteristically show relationships and communications between classes or objects, without specifying the ending application classes or objects that are involved. Algorithms are not thought of as design patterns, since they resolve computational problems rather than design problems.

Not every software patterns are design patterns. Other kinds of patterns, such as architectural patterns, illustrate problems and answers that have alternative scopes. Design patterns pact specifically with problems at the level of software design.

Design patterns can pace up the development procedure by supplying tested, proven development paradigms. Reusing design patterns assists to avoid subtle issues that can cause major problems, and it also improves code readability for developers and designers who are proverbial with the patterns. Efficient software design necessitates considering issues that may not become visible until later in the implementation [11].

Often, people only understand how to relate certain software design techniques to certain problems. These techniques are difficult to apply to a wider range of problems. Design patterns provide general solutions, documented in a format that doesn't require specifics tied to a particular problem.

Design patterns form a consistent language that can be used to depict classic solutions to common object oriented design problems. These patterns allow us to discuss systems of objects as quasi-encapsulated entities. By using design patterns to solve programming problems, the proper perspective on the design process can be maintained.

Design patterns are composed of more sections. Of particular notice are the Structure, Participants, and Collaboration sections. These sections explain a design motif: a prototypical micro-architecture that developers duplicate and adapt to their particular designs to solve the repeated problem described by the design pattern. A micro-architecture is a collection of program constituents (e.g., classes, methods...) and their relations. Developers use the design pattern by introducing in their designs this prototypical micro-architecture, which means that micro-architectures in their designs will have structure and association similar to the selected design pattern.

In addition, patterns allow developers to converse by means of well-known, well understood names for software interactions. Common design patterns can be improved over time, making them further robust than unplanned designs. Considering situations where patterns are used suitably in a program to solve their consequent design problems and assuming that the developers have a good knowledge of design patterns.

5. QUALITY ATTRIBUTES (QA)

We investigated the impact of design patterns on the overall quality of program in a SH point of view, thus attributes like Learnability and Understandability refer to the whole program and not to the pattern solely.

The quality attributes are defined as:

1. **Robustness:** The degree to which an executable program prolongs to function suitably under a typical circumstances or conditions.
2. **Scalability:** Scalability is the ease with which an application or component can be customized to expand its offered capabilities at runtime.
3. **Reusability:** Reusability here is the degree to which a piece of design be capable to be reused in another design.
4. **Trainability:** The degree to which the code source of a program is effortless to be trained by new developers.
5. **Realizability:** The degree to which the code source of a program is easy to understand.
6. **Modularity:** The degree to which the implementation of functions in a program are independent from one another.
7. **Modularity at runtime:** The degree to which functions of a program are independent from one another during execution.
8. **Generality:** The degree to which a software product can achieve a wide range of functions.
9. **Extendability:** The degree to which architectural, data or procedural design can be extended.
10. **Simplicity:** The degree to which a program can be understood without intricacy.
11. **Maintainability:** The maintainability of the code source of a program is its aptitude to undergo repair and evolution.
12. **Availability:** The availability of the code source of a program is a measure of its readiness for usage
13. **Reliability:** The reliability of a system is a measure of the ability of a system to keep operating over time.

6. THE SURVEY ON QA and DP

Considering situations where patterns are used appropriately in a program to solve SH's corresponding design problems and assuming that the SH have a good knowledge of design patterns, we underwent a survey. We prepared a questionnaire and asked SH to circle the letter corresponding to their answer to the questions. If they hesitate between options, we asked the most conservative choice. If they have a doubt, they were asked to choose option F. The options given were

1. Very affirmative
2. Affirmative
3. Insignificant
4. Negative
5. Very Negative
6. No Idea

For the survey we chose 23 DPs which are Abstract Factory, Builder, Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Façade, Flyweight, Proxy, Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, and Visitor.

The example of sample Questionnaire is given in Figure 2.

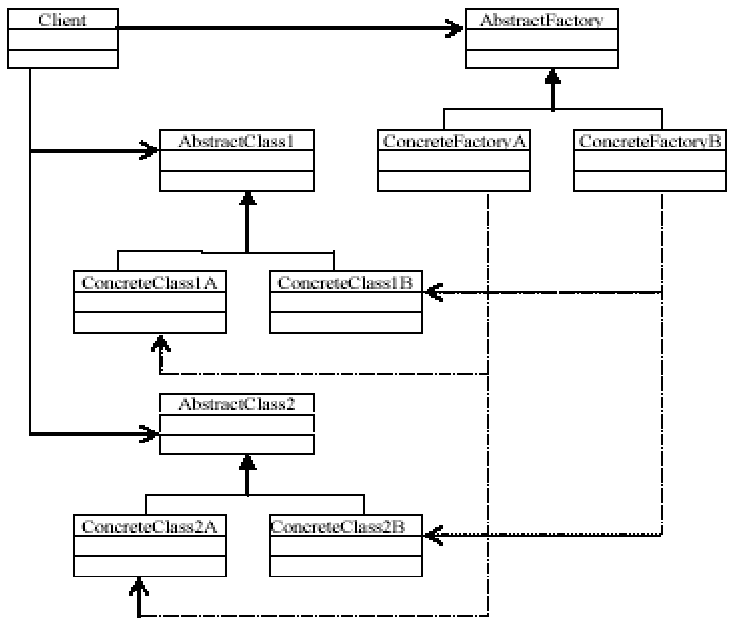

FIGURE1: Design pattern sample (Abstract Factory)

Abstract Factory							Comments
Robustness	1	2	3	4	5	6	
Scalability	1	2	3	4	5	6	
Reusability	1	2	3	4	5	6	
Trainability	1	2	3	4	5	6	
Realizability	1	2	3	4	5	6	
Modularity	1	2	3	4	5	6	
Modularity at runtime	1	2	3	4	5	6	
Generability	1	2	3	4	5	6	
Extendability	1	2	3	4	5	6	
Simbilicity	1	2	3	4	5	6	
Maintainability	1	2	3	4	5	6	
Availability	1	2	3	4	5	6	
Reliability	1	2	3	4	5	6	

FIGURE2: Sample Questionnaire used in the Survey

7. TRAINING AND TESTING

7.1 Radial Basis Function Neural Network (RBF-NN)

Radial Basis Functions emerged as a variant of artificial neural network in late 80's.Radial basis function (RBF) neural network is based on supervised learning. RBF networks were independently proposed by many researchers and are a popular alternative to the MLP. RBF networks are also good at modeling nonlinear data and can be trained in one stage rather than using an iterative process as in MLP and also learn the given application quickly. RBF's are embedded in a two layer neural network, where each hidden unit implements a radial activated function [22]. Due to their nonlinear approximation properties, RBF networks are able to model complex mappings, which perception by means of multiple intermediary layers [23].

Radial basis networks can require more neurons than standard feed forward back propagation networks, but often they can be designed in a fraction of the time it takes to train standard feed forward networks. They work best when many training vectors are available. RBF networks have been successfully applied to a large diversity of applications including interpolation [24], image restoration [25], shape-from-shading [26], 3-D object modeling [27], data fusion [28], etc.

RBF-NN is used by many authors for prediction [13, 14, 15] .During testing; the impact of quality attributes on design pattern (IQADP) is analyzed as discussed in the previous section. Then RF was analyzed based on ATAM. IQADP and RF were trained using RBF-NN.

While testing, the SA is analyzed to get the number and types of DPs used. These are given to RBF-NN to get the RF based on the trained data arrived using the previous procedure.

8. CONCLUSION

In this paper we have discussed some limitation of ATAM and proposed an Expert System. The proposed system can be used for Predicting risk factors of a SA based on SHE using RBF-NN. The Knowledge Base we have proposed is based on the expert SHE so the accuracy is improved. We have analyzed and added more possible quality attribute to improve the quality of RISK prediction. The performance of ATAM is also improved as it is automated.

9. REFERENCES

1. R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, "*The Architecture Tradeoff Analysis Method*", Proceedings of ICECCS'98, 8-1-1998.

2. R.L. Nord, M.R. Barbacci, P. Clements, R. Kazman, M. Klein, L. O'Brien, J.E. Tomayko, "*Integrating the Architecture Tradeoff Analysis Method (ATAM) with the cost benefit analysis method (CBAM)*", CMU SEI Technical Note CMU/SEI-2003-TN-038, Software Engineering Institute, Pittsburgh, PA, 2003.

3. Paul Clements, John Bergey and Dave Mason," *Using the SEI Architecture Tradeoff Analysis Method to Evaluate WIN-T: A Case Study*" CMU SEI Technical Note CMU/SEI-2005-TN-027. Software Engineering Institute, Pittsburgh, PA, 2005.

4. Houmb, Siv Hilde; Georg, Geri; Jürjens, Jan; France, Robert: "*An Integrated Security Verification and Security Solution Design Trade-Off Analysis Approach*". Integrating Security

and Software Engineering: Advances and Future Visions / Mouratidis, Haralambos; Giorgini, Paolo: Idea Group Inc, 2006, 190-219

5. Bate, I. and N. Audsley (2002): "*Architecture Trade-off Analysis and the Influence on Component Design*". Proceedings of Workshop on Component-Based Software Engineering: Composing Systems from Components

6. Mildred N. Ambe, Frederick Vizeacoumar " *Evaluation of two architectures Using the Architecture Tradeoff Analysis Method (ATAM)*", 2002.

7. "*Software Architecture for Software-Intensive Systems*" from www.sei.cmu.edu/architecture /ata_ method.html

8. Arnon Rotem-Gal-Oz, "*Architecture Tradeoff Analysis Method*" www.rgoarchitects.com/Files / ATAM.ppt

9. Gamma Erich, Richard Helm, Ralph Johnson, and John Vlissides (1995). "*Design Patterns: Elements of Reusable Object-Oriented Software*", hardcover, 395 pages, Addison-Wesley.

10. Gabriel Richard (1996). "*Patterns of Software: Tales from the Software Community*". Oxford University Press.

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

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

13. Panda, Sudhanshu S., Chakraborty, Debabrata, and Pal, Surjya K., "*Prediction of Drill Flank Wear Using Radial Basis Function Neural Network*", Proceedings of the National Conference on Soft Computing Techniques for Engineering Applications, NIT Rourkela, 2006, pp. 94-102.

14. Zhou, Guozhong, McCalley, James D. and Honavar, Vasant (1997) "*Power System Security Margin Prediction Using Radial Basis Function Networks*". Technical Report TR97-10, Department of Computer Science, Iowa State University.

15. Sunyoung Lee, Sungzoon Cho, and Patrick Wong "*Rainfall Prediction using Artificial Neural Networks*", Journal of Geographic Information and Decision Analysis, Vol. 2, No. 2, pp. 253-264, 1998.

16. Biffl S., Denger C., Elberzhager F., Winkler D.: "*Quality Assurance Tradeoff Analysis Method (QATAM) - An Empirical Quality Assurance Planning and Evaluation Framework*", Technische Universität Wien, Technical Report IFS-QSE-07/04, 2007

17. Femi G. Olumofin and Vojislav B. Miˇsi´c " *Extending the ATAM Architecture Evaluation to Product Line Architectures"* in Department of Computer Science, University of Manitoba Winnipeg, Manitoba, Canada R3T 2N2 June 2005.

18. A. BinSubaih, S.C. Maddock (2006), "*Using ATAM to Evaluate a Game-based Architecture*", Workshop on Architecture-Centric Evolution (ACE 2006), Hosted at the 20th European Conference on Object-Oriented Programming ECOOP 2006, July 3-7, 2006, Nantes, France.

19. Liming Zhu, Muhammad Ali Babar, Ross Jeffery" *Distilling Scenarios from Patterns for Software Architecture Evaluation – A Position Paper*" EWSA 2004: 225-229.

20. Ali Babar, M., Kitchenham, B., "*Assessment of a Framework for Comparing Software Architecture Analysis Methods*", in Proceedings of the 11th International Conference on Evaluation and Assessment in Software Engineering, 2007, Keele, England.

21. Dongyun Liu "*Mapping requirements to software architecture by feature-orientation*" Hong Mei in Institute of Software, School of Electronics Engineering and Computer Science Peking University, Beijing 100871, P.R.China.

22. Adrian G. Bors, "*Introduction of the Radial Basis Function (RBF) Networks*", Department of Computer Science, University of York, UK.

23. Haykin, S. (1994) "*Neural Networks: A comprehensive Foundation*". Upper Saddle River, NJ; Prentice Hall.

24. Broomhead, D.S., Lowe, D. (1988) "*Multivariable functional interpolation and adaptive networks*," Complex Systems, vol.2, pp.321-355.

25. Cha, I., Kassam, S.A., (1996) " *RBFN restoration of nonlinearly degraded images*," IEEE Trans. on Image Processing, vol. 5, no. 6, pp. 964-975.

26. Wei, G.-Q., Hirzinger, G., (1997) " *Parametric shape -from-shading by radial basis functions*, " IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 19, no. 4, pp. 353-365.

27. Bors, A.G., Pitas, I., (1999) " *Object classification in 3-D images using alpha-trimmed mean radial basis function network*," IEEE Trans. on Image Processing, vol. 8, no. 12, pp. 1744-1756.

28. Chatzis, V., Bors, A. G., Pitas, I., (1999) "*Multimodal decision-level fusion for person authentification*," IEEE Trans. on Systems, Man, and Cybernetics, part A: Systems and Humans, vol. 29, no.6, pp. 674-680.

29. 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