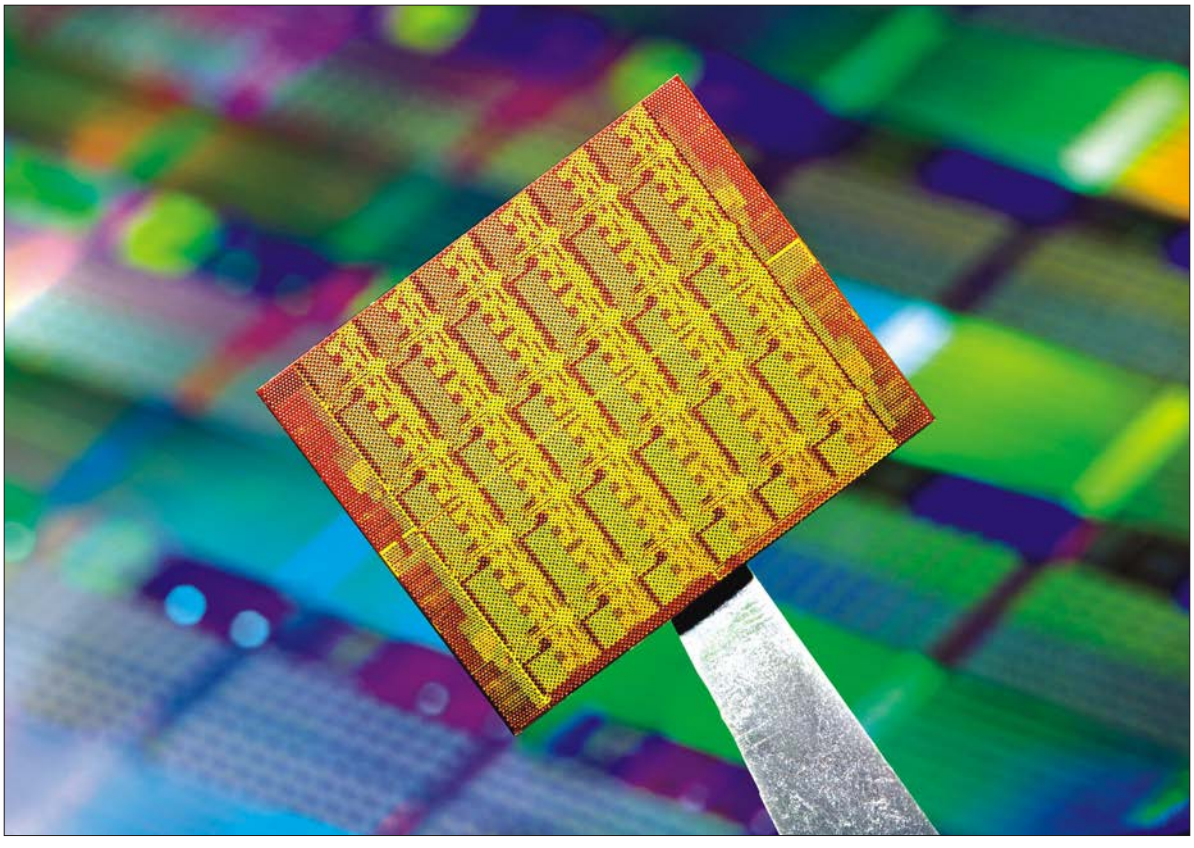

Australasian Journal of

Computer Science

ISSN: 2251-3221

science alert

http://scialert.net/ajcs

Australasian Journal of Computer Science 1 (1): 9-16, 2014
ISSN 2251-3221 / DOI: 10.3923/aujcs.2014.9.16

Software Architecture Design Using Service Oriented on Quality Metrics

C.K. Gomathy and S. Rajalakshmi
SCSVMV Univeristy, Enathur, Tamil Nadu, India

Corresponding Author: C.K. Gomathy, SCSUMU University, Enathur, Tamil Nadu, India

ABSTRACT

Software architecture has the possible to vastly improve organization's potency. The firm wants technology and method know-how: Especially, Service Oriented Architecture (SOA) implies completely different stress of project management. Trendy software industry more and more rely heavily on evidence keep and processed in circulated various information sources and services to create vital, high-value quality decisions. Service-oriented systems are dynamic in nature and are becoming ever a lot of advanced architecture of systems. In such systems, knowing however a knowledge set was derived is, of serious importance in essential its validity and dependability. Today's analytical data systems demand innovative design ideas. So, as to address necessities like flexibility and quicker time-to-market. This study presents service oriented-bound design as a pattern based and platform independent to solve the quality issues. The study conjointly mentions the present problems, technical realization sector which require being researched a lot of with this design.

Key words: Quality architect in SOA, quality management in architecture, service oriented computing, patterns of SOA, software architect management

INTRODUCTION

The software architecture of a program or computing system is the structure or structures of the system which comprise software elements, the externally visible properties of those elements and the relationships among them (Rasool and Asif, 2007). Software has been used in every walk of life, playing increasingly important role. The ever-increasing expansion of applications and users requirements make a steep rise in the scale and complexity of software which results in the decrease in the software quality. So, it is a great challenge in software engineering to understand, measure, manage, control and even to low the software complexity (Pan, 2011). Software Product Line engineering aims at improving productivity and decreasing realization times by gathering the analysis, design and implementation activities of a family of systems. Variabilities are characteristics that may vary from a product to another. The main challenge in the context of the Software Product Lines (PL) approach is to model and implement these variabilities (Abdelmoez *et al*., 2009). Software Architecture (SA) is considered of highest importance to the software development life-cycle. It is used to represent and communicate the system structure and behavior to all of its stakeholders with various concerns. Additionally, SA facilitates stakeholders in understanding design decisions and rationale, further promoting reuse and efficient evolution. One of the major issues in software systems development today is systematic SA restructuring to

accommodate new requirements due to the new market opportunities, technologies, platforms and frameworks (Dobrica *et al.*, 2011). The ultimate goal of software engineering is to be able to automatically produce software systems based on their requirements. For the time being, we pass the synthesis of executable programs and concentrate on the automated derivation of architectural designs of software systems. This is possible because architectural design largely means the application of known standard solutions in a combination that optimizes the quality properties of the software system (Raiha *et al.*, 2009).

The software architecture of a system is the set of structures needed to reason about the system which comprises software elements, relations among them and properties of both. The term also refers to documentation of a system's software architecture. Documenting software architecture facilitates communication between stakeholders, documents early decisions about high-level design and allows the reuse of design components and patterns between projects (Ludik *et al.*, 2011). Software programming is a hard design task, mainly due to the complexity involved in the process. Nowadays this complexity is increasing to levels in which reuse of previous software designs are very useful to short cut the development time (Sharma *et al.*, 2007).

The various benefits of the software architecting are as given below:

- Architecting helps manage complexity
- Architecting ensures architectural integrity
- Architecting reduces maintenance costs
- Architecting provides a basis for reuse (Eeles, 2009)

Popular goals of software engineering are to develop and use techniques and tools for creating high quality applications. Applications that have high quality and modularity are more stable and maintainable (Al Dallal, 2009). The major design task in building enterprise applications is to design good software architecture. During recent years, the notion of software architecture has emerged as the appropriate level for dealing with software quality. One of the major issues in software systems development today is quality. A quality attribute is a nonfunctional characteristic of a component or a system (Gumuskaya, 2005). The usability of software can have a considerable impact on the total cost of ownership through factors such as training, productivit and technical support. As a result, organizations planning to acquire new software (or upgrades) often evaluate software usability as part of their software acquisition decisions. However, evaluating software usability can be a challenging proposition from the perspective of striking a balance between costs (in time and effort), validity and objectivity (Sobiesiak and Diao, 2010).

Software must possess the qualities like safety, reliability, availability, cost, maintainability, performance or response, time, energy consumption (Meedeniya, 2011). Usability is important not only to increase the speed and accuracy of the range of tasks carried out by a range of users of a system but also to ensure the safety of the user. Productivity is also imperative where the software is used to control dangerous processes. Computer magazine software reviews now include usability as a ratings category (Seffah *et al.*, 2006). There are some recent attempts to establish software science as a foundation of software engineering. This may promote more analytical reasoning about software architecture, if it becomes popular. Software architectural design would benefit from analytical reasoning with scientific foundations. Importance of software architecture in the software design process is generally accepted among practitioners (Dey, 2011).

Review of literature: A handful of researches have been done in the field of software architecture since it has gained more importance with the development in computer technologies. Some of the recent researches are as mentioned.

Xu *et al.* (2006) have proposed a research question of transforming dependability requirements into corresponding software architecture constructs by proposing first, that dependability needs could be classified into three types of requirements and second, an architectural pattern that allows requirements engineers and architects to map the three types of dependability requirements into three corresponding types of architectural components. The proposed pattern was general enough to work with existing requirements techniques and existing software architectural styles, including enterprise and product-line architectures.

Kaur *et al.* (2009) have presented a survey of current component-based software technologies and the description of promotion and inhibition factors in CBSE. The features that software components inherit were also discussed. Quality Assurance issues in component based software were also catered to. The feat research on the quality model of component based system starts with the study of what the components are, CBSE, its development life cycle and the pro and cons of CBSE. Various attributes were studied and compared keeping in view the study of various existing models for general systems and CBS. When illustrating the quality of a software component an apt set of quality attributes for the description of the system should be selected.

Zimmermann *et al.* (2009) have proposed a formal definition of architectural decision models as directed acyclic graphs with several types of nodes and edges. In their model, architectural decision topic groups, issues, alternatives and outcomes form trees of nodes connected by edges expressing containment and refinement, decomposition and triggers dependencies, as well as logical relations such as incompatibility of alternatives. The formalization could be used to verify integrity constraints and to organize the decision making process; production rules and dependency patterns could be defined.

A key aspect of the design of any software system was its architecture. An architecture description provides a formal model of the architecture in terms of components and connectors and how they were composed together. COSA (Component-Object based Software Structures), was based on object-oriented modeling and component-based modeling. The model improves the reusability by increasing extensibility, evolvability and compositionality of the software systems. Smeda *et al.* (2009) have presented the COSA modelling tool which help architects the possibility to verify the structural coherence of a given system and to validate its semantics with COSA approach.

Kim *et al.* (2009) have presented a quality-driven approach to embodying Non-Functional Requirements (NFRs) into software architecture using architectural tactics. Architectural tactics are reusable architectural building blocks, providing general architectural solutions for common issues pertaining to quality attributes. The architectural tactics are represented as feature models and their semantics was defined using the Role-Based Metamodeling Language (RBML) which was a UML-based pattern specification notation. Given a set of NFRs, architectural tactics are selected and composed and the composed tactic was used to instantiate an initial architecture for the application. The proposed approach addresses both the structural and behavioral aspects of architecture.

Ampatzoglou *et al.* (2012) have proposed a methodology for comparing design patterns to alternative designswith an analytical method. Additionally, the methodology compares three design patterns with two alternative solutions, with respect to several quality attributes. A

theoretical/analytical methodology to compare sets of "canonical" solutions to design problems were defined. They proposed theoretical study in the sense that the solutions are disconnected from real systems, even though they stem from concrete problems also analytical in the sense that the solutions are compared based on their possible numbers of classes and on equations representing the values of the various structural quality attributes in function of these numbers of classes. The exploratory designs have been produced by studying the literature, by investigating open-source projects and by using design patterns.

Shatnawi and Li (2011) have proposed a hierarchal quality model where the effect of software refactoring on software quality was studied. They provided details of their findings as heuristics that can help software developers make more informed decisions about what refactoring to perform in regard to improve a particular quality factor. They validate the proposed heuristics in an empirical setting on two open-source systems. They found that the majority of refactoring heuristics do improve quality; however some heuristics do not have a positive impact on all software quality factors. In addition, they found that the impact analysis of refactoring divides software measures into two categories: high and low impacted measures. These categories help in the endeavor to know the best measures that could be used to identify refactoring candidates.

Problem definition: Software architecture is generally the structure of components in a program or system, their interrelationships and the principles and design guidelines that control the design and evolution in time (Gumuskaya, 2005). The various problems that remains in the existing researches are being identified from our review of recent researches. Some of the problems existing in the research field are:

- Application of process and framework to large and complex software systems during architecture are not possible (Rasool and Asif, 2007)
- The effectiveness of architectural design largely depends on the quality attributes (Dey, 2011)
- Architecture generally exhibit higher computational time
- Service oriented architecture designing must be completely a service oriented process with better quality aspects (Pahl and Barrett, 2010)
- The quality attributes related to execution qualities should be fully supported (Ovaska *et al.*, 2010)
- Explore ways to improve the accuracy of architectural knowledge sharing quality prediction (Liang *et al.*, 2011)

Architecture principles of SOA: There are many definitions of software architecture:

- Every software has architecture
- Architecture defines components and their interactions
- Interfaces (externally visible behavior) of each component are part of the architecture
- Interfaces allow components to interact with each other
- A system comprises many different kinds of components but none of these is the architecture

Software architecture is a metaphor that helps us to better cope with the challenges in software systems. These challenges are described by a number of so-called "Laws of Software Evolution".

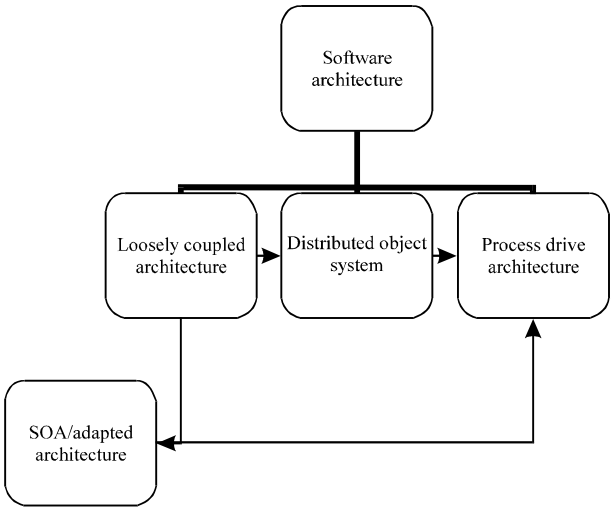

Fig. 1: Loosely coupled architecture and SOA

The two most prominent are:

- Law of continuing change
- Law of increasing complexity

But software architectures are not easy to document, create and maintain and description of the architecture using quality attributes.

Characteristics of an architect:

- The architect is a technical leader
- The architect understands the software development process
- The architect has knowledge of the business domain
- The architect is a good communicator
- The architect makes decisions
- The architect is aware of organizational politics
- The architect is a negotiator
- The architect has technology knowledge
- The architect has design skills
- The architect role may be fulfilled by a team
- The architect has programming skills

Architectural quality attributes:

- Quality of architecture essential attributes for the fulfillment of the requirements
- Factors those are important to make architecture good or bad

System quality attributes: Availability, reliability, maintainability, understandability, changeability, resolvability, testability, portability, efficiency, scalability, security, inerrability, reusability.

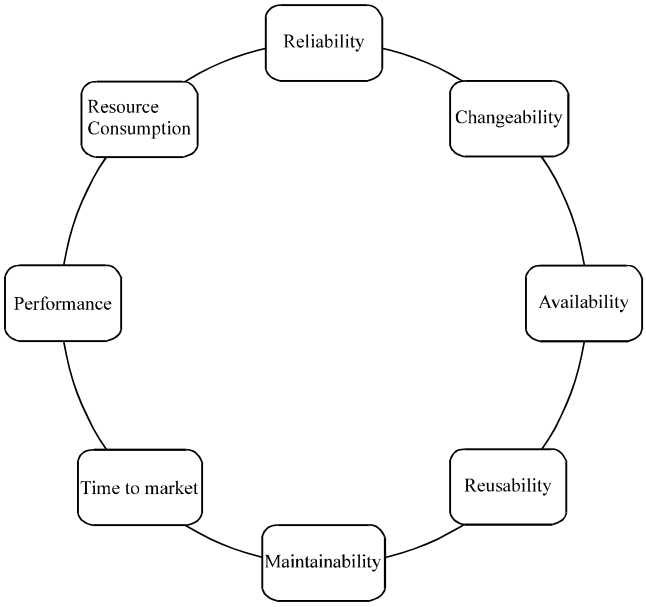

Fig. 2: Architectural quality attributes

Business quality attributes: Time to market, costs and projected lifetime, targeted market, legacy system integration, roll-out schedule.

Architecture quality attributes: Conceptual integrity, correctness, completeness, buildability (Fig. 2).

Contribution of SOA: The quality aware software architecture remains as one of the basic needs while designing software. Various researches have been done in the field of software engineering in order to overcome these drawbacks. But still software architecture remains to be a tedious job for the designers with the consideration of the quality metrics. Various quality attributes such as maintainability, reliability, readability, usability etc. have to be considered while designing software. In this study, proposed an efficient software architecture model based on service oriented architecture design pattern with major consideration being the quality metrics. The Service Oriented Architectural (SOA) pattern is used for our designing purpose. The service oriented architecture design generally the way of designing a software system to provide service to either end user or other service through published interfaces. The usage of service oriented architectural patterns in our design provides reusable and extensible technical solutions to common design problems in a standard Architectural format. The architecture design is also incorporated with the quality metrics like usability and portability to perform a better software architectural design. The usability of the software can be measured with the help of various usability metrics like task completion, time on task (usage time), error counts and satisfaction scores through a process called six sigma methods and the portability is measured using the matrix method. The proposed method proper maintainable and deliverable of better outcome in the form of design metrics (Fig. 1, 3).

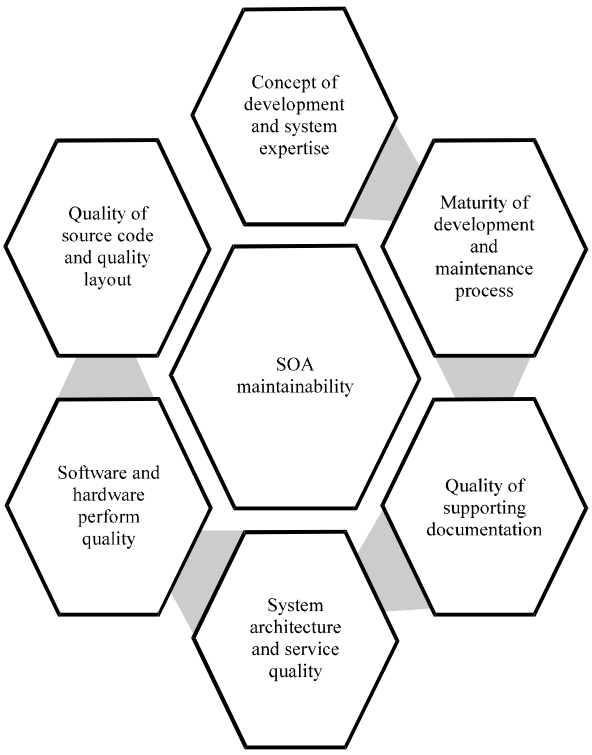

Fig. 3: Architect maintainability analysis

CONCLUSION

This study proposed a new quality architect paradigm to enable system quality to connect with software architectural models from which it is possible to extract precisely information. Our scheme has been proven quality in the standard model. A systematic complexity analysis and extensive experiments show that our proposal is also efficient in terms of computation and design. These features quality analysis framework scheme a talented solution to group-service oriented communication with access control in various types of design.

REFERENCES

Abdelmoez, W.M., A.H. Jalali, K. Shaik, T. Menzies and H.H. Ammar, 2009. Using software architecture risk assessment for product line architectures. Proceedings of the International Conference on Communication, Computer and Power, February 15-18, 2009, Muscat, Oman.

Al Dallal, J., 2009. Software similarity-based functional cohesion metric. IET Software, 3: 46-57.

Ampatzoglou, A., G. Frantzeskou and I. Stamelos, 2012. A methodology to assess the impact of design patterns on software quality. Inform. Software Technol., 54: 331-346.

Dey, P.P., 2011. Strongly adequate software architecture. World Acad. Sci. Eng. Technol., 60: 1559-1562.

Dobrica, L., A.D. Ioniþa, R. Pietraru and A. Olteanu, 2011. Automatic transformation of software architecture models. U.P.B. Sci. Bull. Series C, 73: 3-16.

Eeles, P., 2009. Software architecture masterclass. Proceedings of the IBM Rational Software Conference, May 31-June 4, 2009, Orlando, FL., USA.

Gumuskaya, H., 2005. Core issues affecting software architecture in enterprise projects. Proc. World Acad. Sci. Eng. Technol., 9: 32-37.

Kaur, I., P.S. Sandhu, H. Singh and V. Saini, 2009. Analytical study of component based software engineering. World Acad. Sci. Eng. Technol., 26: 370-375.

Kim, S., D.K. Kim, L. Lu and S. Park, 2009. Quality-driven architecture development using architectural tactics. J. Syst. Software, 82: 1211-1231.

Liang, P., A. Jansen, P. Avgeriou, A. Tang and L. Xu, 2011. Advanced quality prediction model for software architectural knowledge sharing. J. Syst. Software, 84: 786-802.

Ludik, T., J. Navratil and A. Langerova, 2011. Process oriented architecture for emergency scenarios in the Czech Republic. World Acad. Sci. Eng. Technol., 59: 2342-2351.

Meedeniya, I., 2011. Robust optimization of automotive software architecture. Proceedings of the Auto CRC Technical Conference, July 7, 2011, Melbourne, Australia.

Ovaska, E., A. Evesti, K. Henttonen, M. Palviainen and P. Aho, 2010. Knowledge based quality-driven architecture design and evaluation. Inform. Software Technol., 52: 577-601.

Pahl, C. and R. Barrett, 2010. Pattern-based software architecture for service-oriented software systems. e-Inform. Software Eng. J., 4: 31-45.

Pan, W., 2011. Applying complex network theory to software structure analysis. World Acad. Sci. Eng. Technol., 60: 1630-1636.

Raiha, O., E. Makinen and T. Poranen, 2009. Using simulated annealing for producing software architectures. Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference, July 8-12, 2009, Canada, pp: 2131-2136.

Rasool, G. and N. Asif, 2007. Software architecture recovery. World Acad. Sci. Eng. Technol., 34: 99-104.

Seffah, A., M. Donyaee, R.B. Kline and H.K. Padda, 2006. Usability measurement and metrics: A consolidated model. Software Qual. J., 14: 159-178.

Sharma, A., R. Kumar and P.S. Grover, 2007. A critical survey of reusability aspects for component-based systems. World Acad. Sci. Eng. Technol., 33: 35-39.

Shatnawi, R. and W. Li, 2011. An empirical assessment of refactoring impact on software quality using a hierarchical quality model. Int. J. Software Eng. Appl., 5: 127-149.

Smeda, A., A. Alti, M. Oussalah and A. Boukerram, 2009. Cosastudio: A software architecture modeling tool. World Acad. Sci. Eng. Technol., 49: 263-266.

Sobiesiak, R. and Y. Diao, 2010. Quantifying Software Usability Through complexity Analysis. IBM Press, New York.

Xu, L., H. Ziv, T.A. Alspaugh and D.J. Richardson, 2006. An architectural pattern for non-functional dependability requirements. J. Syst. Software, 79: 1370-1378.

Zimmermann, O., J. Koehler, F. Leymann, R. Polley and N. Schuster, 2009. Managing architectural decision models with dependency relations, integrity constraints and production rules. J. Syst. Software, 82: 1249-1267.