SOFTWARE PATTERN QUALITY
COMPROMPT IN SERVICE-ORIENTED
ARCHITECTURES

C. K. Gomathy, Research Scholar
Dr. S. Rajalakshmi, Prof.
Department Of Computer Science And Engineering
Sri Chandrasekharendra Saraswathi Viswa Maha Vidyalaya (University)
Enathur, Kanchipuram

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 paper presents Service oriented-bound design as a pattern based and platform independent to solve the quality issues. The paper conjointly mentions the present problems, technical realization sector which require being researched a lot of with this design.

Keywords: Quality Management in SOA Service oriented Computing, Design 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 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. 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.

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. 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.

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. 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.

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

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. 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. The usability of software can have a considerable impact on the total cost of ownership through factors such as training, productivity, 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.

Software must possess the qualities like Safety, Reliability, Availability, Cost, Maintainability, Performance or Response, Time, Energy consumption. 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. 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.

**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. 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.
- The effectiveness of Architectural design largely depend on the quality attributes.
- Service oriented architecture designing must be completely a service oriented process with better quality aspects.
- The quality attributes related to execution qualities should be fully supported.
- Explore ways to improve the accuracy of architectural knowledge sharing quality prediction.
• For service-centric software systems specific properties arising from the often distributed and cross-organizational context are of central importance. The reliability of a system, the availability of services, and the individual service and overall system performance are often crucial.

**Review Of Recent Researches:**

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 below,

Lihua *et al.* 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.* 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 & 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.

Olaf *et al.* 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 (in)compatibility 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. 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.

Suntae et al. 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.

Apostolos et al. have proposed a methodology for comparing design patterns to alternative designs with 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.

Raed Shatnawi and Wei Li 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.
**Architecture Principles Of SOA:**

SOA is the aggregation of components that satisfy a design needs. It comprises components, services, and processes. Components are binaries that have a defined interface (usually only one), and a service is a grouping of components (executable programs) to get the job done. This higher level of application development provides a strategic advantage, facilitating more focus on the business requirement. SOA isn't a new approach to software design; some of the notions behind SOA have been around for years. A service is generally implemented as a coarse-grained, discoverable software entity that exists as a single instance and interacts with applications and other services through a loosely coupled (often asynchronous), message-based communication model. The most important aspect of SOA is that it separates the service's implementation from its interface. Service consumers view a service simply as a communication endpoint supporting a particular request format or contract. How service executes service requested by consumers is irrelevant; the only mandatory requirement is that the service sends the response back to the consumer in the agreed format, specified in contract.

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”

The two most prominent are:

- Law of continuing change
- Law of increasing complexity

**Messaging in SOA:**

SOA supports the concept of dynamic service discovery. The service consumer queries the "service registry for a service, and the service registry returns a list of all service providers that support the requested service. The consumer selects the cost-effective service provider from the list, and binds to the provider using a pointer from the service registry entry. The consumer formats a request message based on the contract specifications, and binds the
message to a communications channel that the service supports. The service provider executes the service and returns a message that conforms to the message definition in service contract. The only dependency between provider and consumer is the contract, which the third-party service registry provides. The dependency is a runtime dependency and not a compile-time dependency. All the information the consumer needs about the service is obtained and used at runtime. The service interfaces are discovered dynamically, and messages are constructed dynamically. The service consumer does not know the format of the request message or response message or the location of the service until the service is actually needed.

The ability to transform messages has the benefit of allowing applications to be much more decoupled from each other.

**Messaging Patterns Catalogue Within SOA Context:**

Messaging patterns exist at different levels of abstraction with the SOA. Some patterns are used to represent the message itself, or attributes of a messaging transport system. Others are used to represent creation of message content or change the information content of a message. Patterns are also used to discuss complex mechanisms to direct messages. SOA messaging patterns can be divided into the following categories:

- **Message type patterns.** Describe different varieties of messages that can be used in SOA.
- **Message channel patterns.** Describe the fundamental attributes of a messaging transport system.
- **Routing patterns.** Describe mechanisms to direct messages between Service Provider and Service Consumer.
- **Service consumer patterns.** Describe the behavior of messaging system clients.
- **Contract patterns.** Illustrates the behavioral specification to maintain a smooth communication between Service Provider and Consumer.
- **Message construction patterns.** Describes the creation of message content that travel across the messaging system.
- **Transformation patterns.** Change the information content of a message within the enterprise level messaging.

**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.
  - **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, build ability

![Architectural Quality Attributes Diagram](image)

**Fig-2: Architectural Quality Attributes**

**Contribution Of The Paper:**

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 work, we have proposed an efficient software architecture model based on Software design a pattern which is
based on Service Oriented Architecture (SOA) with major consideration being the quality metrics. Architectural and design patterns are recurring solutions to software design problems. The SOA design is 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. As the system operates based on the service oriented architectural design process, we have used application software as the input for which the architectural pattern is designed. The SOA based design strategy can be more efficient in designing such application software’s rather than other such methods.

![Fig-3: Architect Maintainability Analysis](image)

The resulting software must be completely service oriented that should benefit the user at a larger extend. 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 delivers better outcome in the form of design metrics when compared with the existing works. A web services oriented application will be used for this quality based software architectural design. Design patterns will play a efficient role in the design of Web services architectures.

**Conclusion**

This paper 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:
Peng Liang, Anton Jansen, Paris Avgeriou, Antony Tang and Lai Xu , "Advanced quality prediction model for software architectural knowledge sharing", The Journal