

# Aspects in Architectural Description of Evolving Systems

Urjaswala Vora  
CDAC, Mumbai and IIT, Bombay  
Mumbai, India  
+91 22 26201606  
urja.vora@gmail.com

## ABSTRACT

Software architecture is the most important artifact in the software life cycle. The architectural description and the architectural design decide the quality and the longevity of the software. The design decisions made to arrive at a particular software architecture decide the value of quality attributes such as performance, reliability, security and modifiability. Also the percentage of design reuse is decided by the design heuristics like modularity and abstraction which are supported by metrics like cohesion and coupling. Aspects are cross-cutting concerns across components. Though concept of aspect-orientation had come up initially for the concerns cross-cutting the code, the concept is very much required to be addressed from the design stage itself. As the metrics for cohesion and coupling get affected due to these cross-cutting concerns, the scenario results in lower values for modifiability and reuse quality attributes. IEEE 1471-2000 is the standard for "Recommended practice for architectural description of software-intensive systems". The standard uses the conceptual framework and its components like views, viewpoints and models. It does talk about concerns such as performance, reliability, security, distribution, and evolvability, but fails to accommodate the description for cross-cutting concerns. In this paper we focus on the importance of aspects in architectural description of evolving systems.

## Categories and Subject Descriptors

D.2.11 [Software Engineering]: Software Architectures – Languages (e.g., description, interconnection, definition).

## General Terms

Documentation, Performance, Design, Standardization, Languages.

## Keywords

Aspects, Architectural Description, Evolving Systems.

## 1. INTRODUCTION

If we ask why software architecture is required then the answer is

1. to analyze, design, code and test at higher level of abstraction
2. to have standardization at the architectural level which can further lead to higher level reuse or design reuse.
3. to avoid architectural erosion and drift where erosion is defined as violations of the architecture and drift is insensitivity of the architecture.
4. To have long lived systems deployed.
5. to attack non-functional requirements or quality concerns like modifiability, portability, scalability, development efficiency, security as solutions to these concerns lie in architecture, not in data design or algorithms.
6. Need to have common language for designers, developers and users.

The properties of software architecture like abstraction, decomposition and modularity are interdependent and influence the software properties like performance and security which are discerned by observing the system at runtime, as well as software properties like modifiability, portability, scalability, development efficiency and reliability which are not discerned by observing the system at runtime. Hence architectural design and description are very important phases in software life cycle as they directly contribute to software quality in a significant way.

## 2. ASPECTS

Aspect-oriented programming (AOP) introduced by Kiczales et al. [7] is a technique that provides a means to cleanly encapsulate and implement aspects that crosscut other modules. Aspects are linguistic entities aimed at describing a crosscutting concern. For example it is difficult to implement an authentication policy and mechanism as a separate entity in a program because the management of authentication information and various checks have to be scattered through the code dealing with the protected functionality. The goal of AOP is to regroup all the elements of a specific concern in a single module. AOP is thus a technique that allows programmers to decompose a system by aspects, in addition to the base decomposition unit, such as class, function, or module.

### 3. ASPECTS AND QUALITY ATTRIBUTE

For each quality attribute, there exist indispensable software properties, which we have to analyze, when we take design decisions affecting the quality attributes and these properties can be called as aspects also as they are cross-cutting concerns across the components. As the quality attribute requirement of the software applies to all software components, here the component can be a class, function, module or even a database component. For example, to examine the performance, we may consider from an aspect related to data accessing or from an aspect related to communications as the performance bottlenecks are evident there. Likewise, if the aspect relates to data access, we may examine the quality attribute based on factors such as the number of data elements, size of the data, and access pattern. As stated by Noda and Kishi [10], some aspects influence multiple quality attributes, and some factors are used in multiple aspects. These aspects and factors are application or technology dependent, and it is very important to list up these aspects and factors as it is quite important to utilize them, because it is not easy to completely understand the very nature of quality attributes.

### 4. ASPECTS IN ARCHITECTURAL DESIGN

Applying the idea of aspects to the architectural design and descriptions is advantageous as recognized by Noda and Kishi [10]. According to them, in the Aspect Oriented Design (AOD), instead of designing software considering functions and required quality attributes simultaneously, the design of the architecture of each aspect is to be done independently, and then to be “woven” into the final architecture. Though, many technical issues have to be solved before realizing AOD, the basic idea of AOD is as follows: Firstly, based on the analysis of requirements and technologies used, redefine the service descriptions in terms of necessary aspects. Then design the software architecture from each aspect independently.

Even Pace and Campo [12] realize the architecture-oriented view is one of the three basic approaches to address the process of separation of concerns. They stress that good separation of concerns is ultimately enforced by architectural means.

As stated by Groher and Baumgarth [5], AOD is still lacking standardized concepts at the design phase that would foster the specification of crosscutting concerns at the high level architecture and low level design. The formalization of aspects in the architectural description will fill that gap. They state that the module construct for a concern should be higher-level than a functional module (which can be class or a function) since a concern itself can consist of several modules and all of these modules may be associated with the module the concern crosscuts. Otherwise associations modeled on module-level would supersede the logical grouping of the modules belonging to one concern. This would make the design models hard to read and lead to graphical tangling of crosscutting concerns further complicating the procedure of evolution of design model.

### 5. ARCHITECTURAL DESCRIPTION

As stated in ongoing version of IEEE 1471-2000 [4], architectural description of software intensive systems should include architectural descriptions for the following:

1. Expression of the system and its evolution
2. Communication among the system stakeholders
3. Evaluation and comparison of architectures in a consistent manner
4. Planning, managing, and executing the activities of system development
5. Expression of the persistent characteristics and supporting principles of a system to guide acceptable change.
6. Verification of a system implementation’s compliance with an architectural description.
7. Recording contributions to the body of knowledge of software-intensive systems architecture.

IEEE 1471 standard identifies the formidable risks and difficulties in the design, construction, deployment, and evolution of software-intensive systems despite of significant efforts to improve engineering practices and technologies.

In the conceptual framework of IEEE 1471, an architectural description is organized into one or more architectural views where each view addresses one or more of the concerns of the system stakeholders. A view conforms to a viewpoint. The viewpoint determines the specification of the conventions for constructing and using a view. An architectural description selects one or more viewpoints for use. The selection of viewpoints depends on the stakeholders to whom the Architectural Description is addressing. A library viewpoint that is defined elsewhere can also be referred in Architectural Description. A view may consist of one or more architectural models. Each such architectural model is developed as per the specifications given in the corresponding architectural viewpoint. An architectural model may participate in more than one view.

In this paper we want to focus on the architectural descriptions for evolving systems where evolvability is the major concern. Software Maintenance is a costly but unavoidable activity. As identified by Parikh and Zvegintzov [13], software maintenance consumes 50% of all computer resources and by Boehm [1] that maintenance costs can be up to ten times those of an initial development. In today’s fast moving world when we consider the systems for which business processes evolve at high frequency and also can not be taken easily offline due to high costs of their downtime (e.g. telephone and banking), evolvability of the system becomes the high priority quality attribute. Evolution is unanticipated concern and crosscuts several modules depending on the change requirement. Hence this aspect is very important in architectural description of such systems.

For evolutionary systems IEEE 1471 standards recognize the need of reassessment of stakeholder’s requirements with each iteration of the architecture design. Architectural descriptions for evolving systems will typically be developed in an iterative pattern, or rhythm, of version changes.

We are focusing on evolving systems hence we need to model for aspects, that is cross-cutting concerns of the problem domain. Identifying the aspects at the architectural design stage will result in a system with higher evolvability.

The principle of encapsulation in Object Oriented (OO) Architectures overcame the flaws present in structured methodology due to separate data and process components and their interdependencies which was an obstacle in evolvability of the system. However OO design has been proved inadequate in addressing design issues due to concerns which are crosscutting across the objects or components or packages. These crosscutting concerns are main obstacles in the evolution of OO systems. Aspect-oriented software development was proposed as a solution to this problem. The architectural description should recognize the aspects at this higher abstraction and the models are to be designed taking care of these aspects.

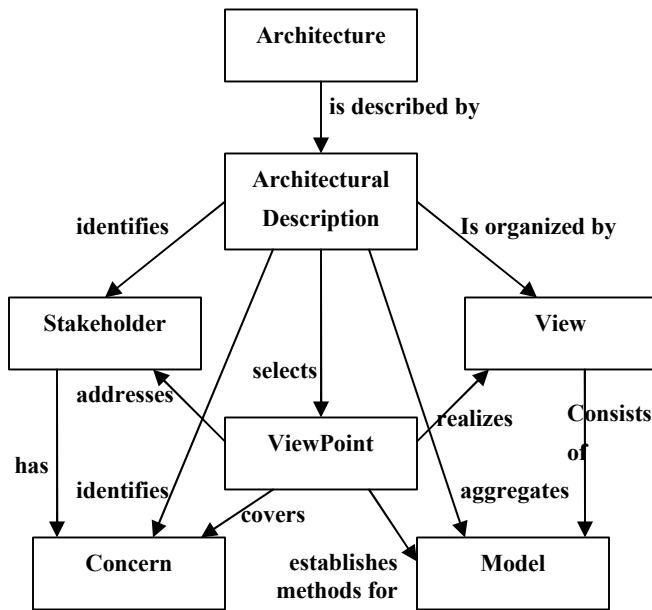


Figure 1. Conceptual Model for Architectural Description

May [9] in his survey of different architectural models compares these five models which focus on describing software architecture from multiple perspectives.

1. Kruchten’s “4+1” View Model [8].
2. Software Engineering Institute (SEI) set of views [3].
3. ISO Reference Model of Open Distributed Processing (RM-ODP) [6].
4. Siemens Four View model [14].
5. Rational Architecture Description Specification (ADS) [11].

The comparison framework is based on the IEEE 1471 standard [4]. We will consider these models and design for the application of aspects in their architecture descriptions. In this paper we will concentrate on Kruchten’s “4+1” View Model only.

Kruchten defined an iterative process for architecture design, with the phases:

1. Description of the critical scenarios.
2. Identification of the key abstractions from the problem domain and model the same in the Logical View.
3. Mapping of logical classes to modules and packages in the Development View,
4. Mapping of logical classes to tasks and processes the Process View.
5. Mapping of the processes and modules to the hardware in the Physical View.

In each subsequent iteration additional scenarios are modeled, following the sequence to make the architectural design final and stable. For this architectural model, along with the critical scenarios it is required to describe critical aspects in the system. Each scenario is to be classified under the aspects recognized for the problem domain. Each scenario can be under more than one aspect and one aspect can have more than one scenarios related to itself. The identifications of key abstractions have to understand the aspect classification and the program decomposition should follow the classification of the scenarios related to various aspects.

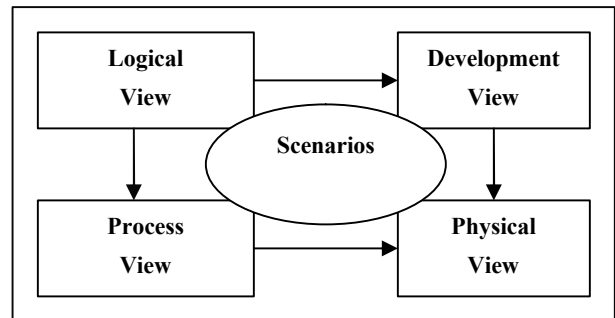


Figure 2. “4+1” View Model without Aspects

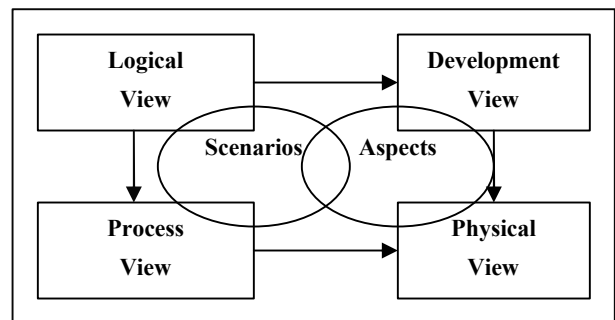


Figure 3. “4+1” View Model with Aspects

The conceptual Model for Architectural Description will have to accommodate aspects. In figure 4 we introduced aspects as an added layer of abstraction between concerns and viewpoints. Now the architectural description is organized into one or more architectural views where each view addresses one or more of the aspects of the system stakeholders. The cross-cutting concerns are identified as aspects. A view conforms to a viewpoint. The viewpoint determines the specification of the conventions for constructing and using a view. An architectural description selects one or more aspects for use. The selection of aspects depends on the stakeholders to whom the Architectural Description is addressing. A view may consist of one or more architectural models. Each such architectural model is developed as per the specifications given in the corresponding architectural viewpoint. An architectural model may participate in more than one view.

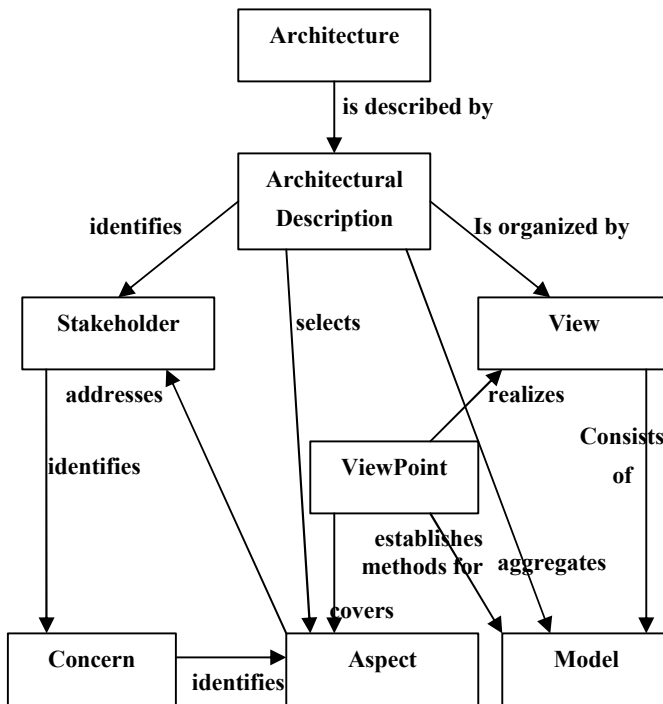


Figure 4. Conceptual Model for Architectural Description with Aspects

Using the comparison framework, May [9] states that there was considerable overlap between the viewpoint models. Hence we can safely assume that the conceptual model for architecture description can be adapted for other viewpoint models as well.

## 6. CONCLUSION

The focus on architectural design was due to the effect it has on non functional requirements, that is quality attributes of a software system. Similarly the concept of aspects contributes to our understanding of system construction, modularization, and the articulation of important functional and non-functional concerns. It is unavoidable to include the aspect-orientation of architectural description in the current standard of architectural description.

## 7. REFERENCES

- [1] Boehm B.W., "The High Cost of Software", in Horowitz E., Practical Strategies for Developing Large Software Systems, Addison Wesley, 1975.
- [2] P. Clements et al. "Documenting Software Architecture: Views and Beyond". Addison Wesley, Boston, MA, USA, 1st edition, 2002. ISBN 0-201-70372-6.
- [3] P. Clements et al. "A practical method for documenting software architectures". <http://www.cs.cmu.edu/~able/publications/icse03-dsa/> viewed on 24th January, 2007.
- [4] IEEE Std. 1471:2000 – Recommended practice for architectural description of software-intensive systems. Los Alamitos, CA: IEEE.
- [5] Iris Groher and Thomas Baumgarth. "Aspect-Oriented from Design to Code". Proceedings of the Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, AOSD conference 2004.
- [6] ISO. Reference Model of Open Distributed Processing (RM-ODP). International Organization for Standardization, 1994. Technical Report 10746.
- [7] Kiczales, G., Lamping J., Mendhekar J., Maeda C., Videira Lopes C., Loingtier J., and Irwin, J. "Aspect-oriented programming.", In Proceedings of the European Conference on Object-Oriented Programming (ECOOP), Springer-Verlag, 1997.
- [8] P. Kruchten. "Architectural Blueprints - The "4+1" View Model of Software Architecture". IEEE Software, 12(6):42–50, 1995.
- [9] Nicholas May. "A Survey of Software Architecture Viewpoint Models". [ISO/IEC 10746]
- [10] N. Noda and T. Kishi. "On Aspect-Oriented Design: An Approach to Designing Quality Attributes". Proceedings of 6th Asia Pacific Software Engineering Conferences (APSEC'99), pp. 230-237, 1999.
- [11] D. Norris. "Communicating Complex Architectures with UML and the Rational ADS". In Proceedings of the IBM Rational Software Development User Conference, 2004.
- [12] J. Andrés Díaz Pace and Marcelo R. Campo, "Analyzing the role of aspects in software design", Communications of the ACM, Volume 44, Issue 10 (October 2001), Pages: 66-73.
- [13] Parikh G., Zvegintzov N. "Tutorial on Software Maintenance". IEEE Computer Society Press, Silver Spring Maryland, 1993.
- [14] D. Soni et al. "Software architecture in industrial applications". In International Conference on Software Engineering, pages 196–207, 1995.