Design Phase Analysis of Software Qualities Using Aspect-Oriented Programming

Daesung Park[1], Sungwon Kang[2] and Jihyun Lee[3]

[1] LG Electronics, Korea
grayger@icu.ac.kr
[2]Information and Communications University, Korea
kangsw@icu.ac.kr
[3]ETRI, Korea
jihyun@etri.re.kr

Abstract

If we can analyze software qualities during the design phase of development without waiting until the implementation is completed and tested, the total development cost and time will be significantly saved. Therefore in the past many design analysis methods have been proposed but either they are hard-to-learn and use or, in the case of simulation-based analysis, functionality concerns and quality concerns were intermingled in the design as well as in the implementation thereby making development and maintenance more complicated. In this paper, we propose a simulation-based design phase analysis method based on aspect-oriented programming. In our method, quality aspects remain separate from functionality aspect in the design model and the implementation code for simulation is automatically obtained by injecting quality requirements into the skeleton code generated from the design level functionality model. Our method has advantages over the conventional approach in reducing both the development cost and the maintenance costly.

Keywords

Aspect-oriented programming, Design stage software analysis, Performance analysis, Reliability Analysis

1. Introduction

Software architecture and design artifacts are produced early in the development process and reflect early solution decisions for the system. They should be carefully designed because later discovery and fixing of problems would result in a much higher cost of development. In the past, many early analysis methods were proposed. However, they have been rarely practiced in industry since they require additional tasks such as modeling, implementation and evaluation. It is desirable to have practical methods that can reduce analysis overhead and can fill the gap between modeling and implementation.

This paper proposes a new method for the design phase quality analysis based on Aspect-Oriented Programming (AOP). AOP is a programming paradigm that realizes the principle of separation of concerns and helps the programmer focus on various aspects one at a time. AOP helps modularization of software by allowing us to express various aspects independently. As will be shown later, AOP can be utilized for analysis of quality attributes for software systems. In this paper, we develop a simulation-based analysis method based on AOP that is general enough to apply for analysis of various quality attributes of software systems.

The rest of the paper is organized as follows: Section 2 introduces related analysis methods and Section 3 discusses aspect-oriented programming. Section 4 presents our analysis approach and compares it with the traditional approach. In Section 5, we show the efficacy of our method with performance and reliability analysis examples. Finally, Section 6 is the conclusion.

2. Related Works

Quality attributes such as performance, reliability, security, usability and etc. are often so critical to the business goals of the software systems that they should be considered from the early design phase. Therefore there have been some methods for early phase analysis of software qualities. Architecture Tradeoff Analysis Method (ATAM) is a method to evaluate software architecture considering multiple quality attributes at once [Kazman 98]. The architecture-based approach gives insight into the sensitivity of the total system structure to each component, but it does not address how to measure each quality attribute.

In the early stage of development, we can conveniently use models that capture quality characteristics of the system to predict software quality since in the phase usually there are no concrete implementations. Therefore most early phase analysis methods were model-based. For example, queuing networks, stochastic process algebras, stochastic timed Petri nets were used to model performance [Balsamo 04], and state-based models, path-based models, and additive models were

used for reliability analysis [Goseva 01]. They are called analytical models that describe a system using formal, mathematical notation [Marzolla 04]. On the other hand, simulation models predict qualities by designing a model similar to the real system, and then implementing and running it.

Although these methods seem to be promising, they are not widely practiced in industry for the following reasons. Analytical models have shortcomings such as the state explosion or the semantic gap from design. Moreover, analytical models may be difficult to learn and use. The drawback of simulation models is the high cost to build simulation programs. Therefore building simulation programs efficiently and maintaining program code are critical issues in simulation-based analysis. In this paper, we develop a simulation-based analysis method that reduces both the development cost and the maintenance cost.

3. Aspect-Oriented Programming

Aspect-Oriented Programming (AOP) is a method that uses aspectual decomposition and composition to separate and then recombine different aspects of a system. When designing and implementing a software system, we usually modularize the system into small units such as objects, modules, procedures, and so forth.

AOP [Kiczales 97] realizes modular implementation of crosscutting concerns. AOP enables us to decompose problems into not only functional components but also into aspects that crosscut functional components, and then implement them by composing these components and aspects. An aspect weaver, so-called aspect compiler, is a code generator that composes them.

In this paper, we use AspectJ to build an executable for a simulation. AspectJ [Kiczales 01] is a language that extends Java to support AOP. It has new concepts such as join point, pointcut, advice, and aspect. Join point is an identifiable point of program execution such as method/constructor calls, method/constructor execution, field get and set, exception handler execution, and static and dynamic initialization. Pointcut is a set of join points selected by a Boolean operation such as “and” or “or”. Advice is used to define additional code to be executed before, after, or around join points. Aspect is a modular unit of crosscutting implementation. The AspectJ compiler merges Java code with AspectJ code to achieve the weaving of crosscutting concerns.

4. Our approach

We explain our approach to design phase analysis of software qualities. We first present our analysis process in detail and then compare it with the traditional approach to show the benefits of our approach.

4.1 The Basic Idea

The basic idea of our approach is to model systems in the aspect-oriented way to implement an executable program for simulation. In order to develop simulation programs, we need to have quality analysis code in addition to code for system functionality. The system functionality concern is captured as a design model, and the quality analysis concern is captured as a quality analysis model. We regard quality analysis models or implementations as aspects because they crosscut multiple software units. In design phase, we decompose a system into core concerns for functionality and aspects for qualities. Each concern is implemented independently and then recomposed to a simulation program.

4.2 The Process

Our analysis process consists of the following steps: (1) Modeling functionality, (2) Modeling quality analysis aspects, (3) Weaving, (4) Simulation and (5) Feedback.

The steps (1) and (2) mean aspectual decomposition for separating aspects. The step (3) is the aspectual composition. This process is pictorially shown in Fig. 1.

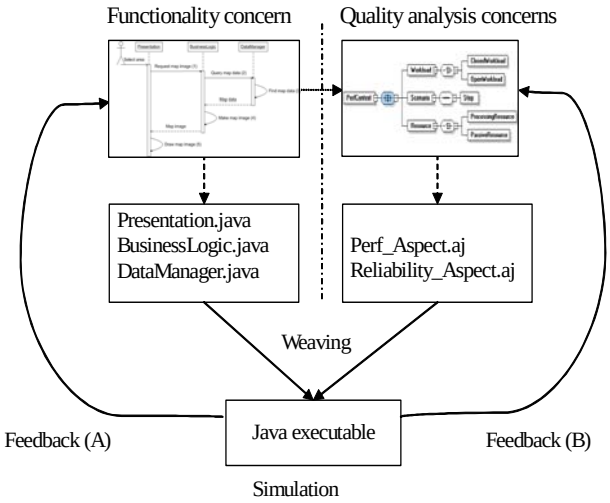

Fig. 1. The process of our analysis

(1) Modeling functionality Analysis starts from constructing a design model based on functionality of a system. A model is an abstract description of software that hides information about some aspects of the software to present a simpler description of others. In this step, we only consider functionality, which is the ability of the system to do the job for which it was intended. For example, we can model a design of software system with UML class diagrams and sequence diagrams. In these diagrams, quality analysis aspects are not included, and only functionality is considered.

In this step, it is very important that we extract the main objects and their operations which are likely to affect quality attribute being analyzed critically. It is easy to generate skeleton program code from class diagrams. Many UML tools can generate code for class, attribute, signature of operation, and relationship from class diagrams. Sequence diagrams can model object interactions arranged in time sequence and to distribute use case behavior to classes. The results of this step are core source code files with the extension ".java".

(2) Modeling quality analysis aspects The next step is to model quality analysis aspect of interest. The proposed approach is scenario-based analysis. A scenario is a sequence of component interactions triggered by a specific input stimulus. Operational profiles are used for description of scenarios with specified input variables (or parameters). With these parameters as input, metrics are used to produce output. Therefore our quality analysis models include quality profiles and metrics. Profiles or metrics differ from quality to quality. Details of them are addressed in Section 5. In case we would consider multiple attributes, we should model these attributes one by one. The results of this step are AspectJ files with the extension ".aj".

(3) Weaving Now we make a simulation program. With AOP, quality analysis aspects can be developed independently. Then AspectJ compiles code files from Step (1) and code files from Step (2) together. This compilation process is called weaving in the aspect-oriented world. The results of this step are executable files with the extension ".class".

(4) Simulation We execute the result of Step (3) and observe the result. Simulation model is based on a number of assumptions about the real system's behavior. The simulation model can be implemented to produce a simulation program. By running it, we can measure values of the parameters of interest. Like some early analysis methods in the past, it can predict software qualities before the actual system is completed [Marzolla 04].

(5) Feedback In analysis process, feedback can result in redesign of software or reorganization of resource demands when analysis result does not meet the requirements. It can be utilized in impact analysis to suggest where to allocate resource most effectively to improve quality of the software system.

Feedback (A) of Fig. 1 stands for modification of the software design. Feedback (B) of Fig. 1 means modification of quality parameters. Initially, parameters used in simulation input are usually predicted values or assumed values. Through the feedback, simulation results can be used as input, thereby making prediction results more precise.

As Fig. 1 shows, Feedback (A) and Feedback (B) are independent of each other because the model and the implementation for the quality analysis concerns were separate from those for the functionality concern. Modification of the design or code for the functionality concern will not influence that for the quality analysis concerns, and vice versa. Therefore, our approach enables us to minimize the coupling between functionality-related modules (i.e., *Presentation.java, BusinessLogic.java* and *DataManager.java*) and quality analysis modules (i.e., *Reliability_Aspect.aj, Perf_Aspect.aj)* in Fig. 1.

4.3 Benefits

The left hand side of Fig. 2 shows the conventional approach. In the approach, software design combines both a functional design and a quality analysis model and therefore code for the design model and code for the quality analysis model are forced to become inseparably mixed in implementation. It is assumed that we manually obtain code from the design model. Then the source code is compiled, executed and analyzed. If necessary, the system is redesigned and the whole process is repeated. Disadvantages of the conventional approach are that (1) it is difficult to develop the design model because the designer should consider both the design model and the quality analysis model at the same time, and (2) it is difficult to understand and maintain the program code because several concerns will be intermingled in the implementation code.

Our approach solves these problems. In the design phase, we clearly set the quality analysis model apart from the design model and maintain the separation to the implementation phase. In other words, the design model has its own Java implementation code, and the quality analysis model has its own AspectJ implementation code. Only at the last moment when the actual implementation is needed, these two are woven together by the AspectJ compiler.

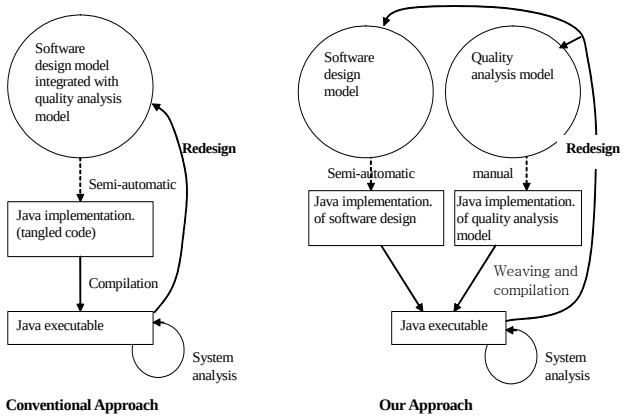

Fig. 2. Comparison with the conventional approach

Our approach has many advantages over the conven-

tional approach. Each of the design model and the quality analysis model is transformed into the corresponding code clearly. Even though feedback may require redesign of either the design model or the quality analysis model, direct code generation speeds up the feedback cycle. Moreover strict modularization in the implemented code enhances understandability and maintainability. We don't have to take the trouble to integrate the separated source code files for ourselves because it can be automatically done by the AspectJ compiler.

5. Application example

In this section, we demonstrate our approach using an example of the map viewer system. The map viewer system is a web-based application that allows users to view a detailed map of a location. When users select an area that they want to see, the map viewer system finds the information of the area and then shows the map image on the web. The system consists of three tiers: Presentation, Business logic, and Data manager. For example, the steps for "showing map image" scenario are as below: (1) The presentation tier gets input from users and requests a map image to Business logic, (2) The business logic tier queries map data to Data manager tier, (3) The data manager tier finds map data and returns it to Business logic tier, (4) The business logic tier makes a map image from map data and returns it to Presentation tier (5) The presentation tier draws the map image and shows it to users.

For the sake of simplicity, this scenario considers neither branching nor alternative flow.

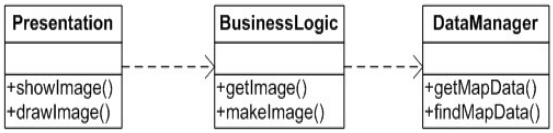

Fig. 3. Class diagram

As mentioned, class diagrams and sequence diagrams describe functionality. Fig. 3 shows object classes and their operations for this application, and Fig. 4 shows Java code mapped to this class diagrams.

Except for the implemented code (lines 4, 5, 14, 15, 23) of each operation in Fig. 4, lines of skeleton code can be generated from class diagrams automatically. The operations in classes will be the join points of AspectJ code for quality analysis aspects and will be used to describe behaviors of the system.

5.1 Performance Analysis

A model should represent behaviors of system and performance requirements in static and dynamic ways, and provide appropriate format that can be easily utilized for the analysis phase. We use XML to represent the performance model.

```
public class Presentation {
     BusinessLogic bl;
     public void showImage( ) {
            bl.getImage( );
            drawImage();
     }
     public void drawImage( ) {}
}
public class BusinessLogic {
     Presentation pre;
     DataManager dm;
     public void getImage() {
            dm.getMapData( );
            makeImage();
     }
     public void makeImage() {}
}
public class DataManager {
     BusinessLogic bl;
     public void getMapData() {
            findMapData();
     }
     public void findMapData() {}
}
```

Fig. 4. Java code for functionality

For simulation, we transform the performance model to the XML format. Each class in the model is mapped to an element in the XML schema and attributes of the class are mapped to the attributes of the element in XML. As performance context class in the model aggregates workload, scenario, and resource class, the performance context node in XML format has three child nodes: workload, scenario, and resource. The input of the simulation is a valid XML file containing actual performance parameters. Table 1 explains performance parameters that appeared as attributes of the XML nodes. The output of the simulation, response time or throughput, is also stored in the XML sharing the same schema.

Table 1. Performance parameters [Allen 97]

NODE	Attribute	Explanation
Workload	population	Size of the workload (number of system users).
Scenario	response time	Total time required to execute the scenario, including all resource waiting, synchronization delays and execution times.
Step	response time	Total delay to execute the step including all resource waiting and all execution times.
	interval	Time interval between successive repetitions of this step, when it is repeated within a scenario.
Resource	capacity	Number of permissible concurrent users.
	throughput	Rate at which the resource performs its function.

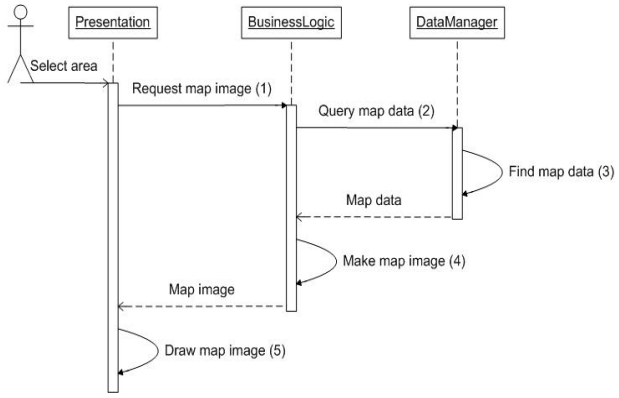

Fig. 5. A sequence diagram for a scenario

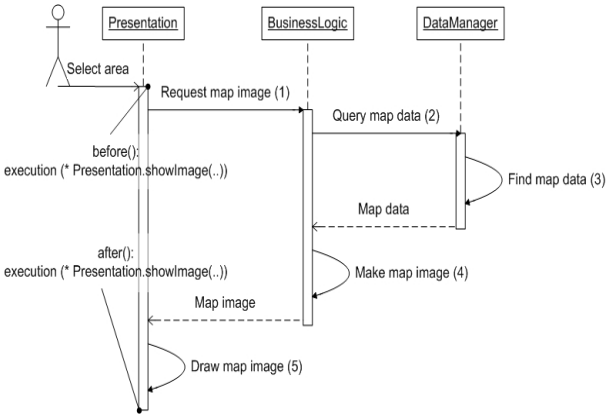

Fig. 6. Sequence diagram with pointcuts and advice

Except for the implemented code (lines 4, 5, 15, 16, 25) of each operation, lines of skeleton code can be generated from class diagrams automatically. The implementation of each operation is manually written from sequence diagrams straightforwardly. Later, the operations in classes will be the join points of AspectJ code reflecting performance characteristics and will be used to describe the behaviors of execution steps.

Performance concern As we mentioned in Section 3, the performance concern is represented in the XML format. As the functionality concern is implemented in Java language, the performance concern is also implemented in Java language. We simulate dynamic behavior of the system with the executable compiled from functionality-based skeleton code and the performance model implementation.

The workloads are generated with Java threads. The number of threads is the same as the population of workloads, or Workload.population in Table 1. Each thread tries to obtain resources to execute the given operation. However, each resource has limited capacity, or Resource.capacity. When a thread representing a workload fails to obtain the resource, it waits for an instance and retries. Step.interval represents the interval between trials. If the thread succeeds in obtaining the resource, it executes the operation for Step.responseTime. The time consumed for execution is emulated using the sleep() method of java.lang.Thread class.

Weaving In Fig. 5, objects in the sequence diagrams can represent resources required for workloads. The functionality concern and the performance concern are weaved by the AspectJ compiler. Fig. 6 shows sequence diagrams overlaid with AspectJ elements. Lines of Code for performance analysis are inserted before or after appropriate pointcuts.

Simulation In this section, we show how to calculate performance metrics using simulation and how to apply the AOP techniques for checking time taken for operations and counting the number of service completions.

```
<J-1> Presentation.showImage();
<J-2> BusinessLogic.getImage();
<J-3> DataManager.getMapData();
<P-1> pointcut pShowImage() :
        execution(*  Presenta-
ı.showImage(..));
<A-1> before() : pShowImage();
<A-2>after() returning : pShowImage();
```

Fig. 7. Join points, pointcut, and advices

The purpose of the simulation is to get the response time of the scenario and the throughput of resource from performance parameters of step, resource, and workload. We can get the response time of scenario, or Scenario.responseTime using join points, pointcuts, and advices of AOP.

In Fig. 7, the lines <J-1>, <J-2>, <J-3> are the operations that appeared in the functionality implementation. The line <P-1> is the pointcut appeared in the performance implementation. When <J-1> calls <J-2>, <J-2> calls <J-3>. The response time of scenario is the time taken to execute <J-1>. To get the response time, time checking should be done in before/after the pointcut <P-1> corresponding to <J-1>. The advices <A-1> and <A-2> mean the very time before <P-1> is executed and the very time after <P-1> is returned respectively. Therefore, we can get the response time by subtracting the moment of <A-1> from the moment of <A-2> because the difference is the elapsed time executing the operation step.

To get the throughput of the resource, or Resource.throughput, we use a simple formula. Let T be the length of time in the observation period, and let C be total number of service completions in the observation period. Then the throughput of the system is C divided by T. We can get T by observing simulation run time and can get C by setting counters before or after appropriate pointcuts.

5.2 Reliability Analysis

To specify reliability concern, we use profiles and metrics. There is no reliability profile universally accepted. Therefore we choose a reliability domain model proposed in [Cortellessa 04]. Parameters that we consider for reliability analysis are shown in Table 2.

Table 2. Reliability parameters [Cortellessa 04]

Node	Attribute	Explanation
Component	fail_prob	The atomic failure probability of a component
	usage_ratio	The usage ratio of a component
User	access_prob	The probability that a certain type of user accesses to the system
	service_prob	The probability that the type of user, once accessed, requires a certain service

The paper [Dolbec 95] proposed a component based reliability model using component reliabilities and component usage ratios. Software system reliability, R_s, is equal to:

$$R_s = 1 - \sum_{k=1}^{m} U_k D_k$$

where m represents the number of components used during system execution, U represents the usage ratio of component k in N_k tests, and D_k represents the probability of failure of component m. This model assumes that the component failures and execution path failures are independent. In our approach, we use this model for a reliability model due to its simplicity.

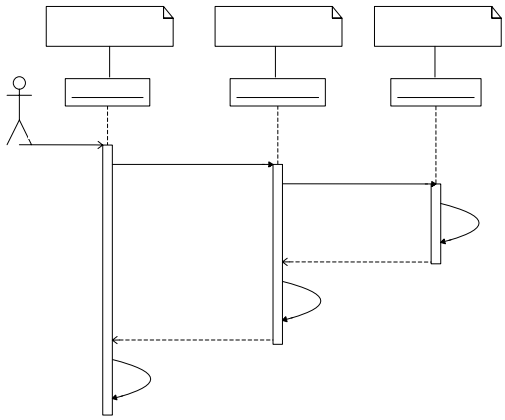

Fig. 8. Sequence diagrams with reliability information

Fig. 8 shows sequence diagrams with reliability parameters. Each component has a probability of failure and a usage ratio. Reliability information is implemented in AspectJ code and weaved into simulation program with functionality implementation.

7. Conclusion

In this paper, we showed a design phase quality analysis method based on AOP and demonstrated its efficacy with performance and reliability analyses for an example. The method is general and can be used for analysis of other quality attributes of software systems.

The benefits of our method are as follows: Firstly, we can check at the design stage whether the implementation will show the required qualities. With a functional model and quality analysis models of interest, we can predict software qualities before the actual system is implemented. Secondly, development becomes much easier than the case in which various concerns for simulation are interwoven with functionality. Thirdly, maintenance is simplified to modifying the functionality model and quality analysis models separately in the design phase. Lastly, separation of concerns enables improved traceability from design to code, and delivers much more understandable code.

8. References

[Allen 97] R. Allen and D. Garlan, "A Formal Basis For Architectural Connection,", A revised version of the paper that appeared in ACM Trans. on Software Engineering and Methodology, July 1997.

[Balsamo 04] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni, "Model-based Performance Prediction in Software Development: A Survey," IEEE Trans. SE, Vol.30, No.5, May 2004.

[Cortellessa 04] V. Cortellessa, A. Pompei, "Towards a UML profile for QoS: a contribution in the reliability domain", Proc. 4th Int'l workshop on Software and performance, 2004.

[Dolbec 95] J. Dolbec, T. Shepard, "A component based software reliability model," Proc. the conference of the Centre for Advanced Studies on Collaborative research 1995.

[Goseva 01] K. Goseva-Popstojanova, K. S. Trivedi, "Architecture-based approach to reliability assessment of software systems," Performance Evaluation, Vol.45/2-3, June 2001.

[Katara 03] M. Katara and S. Katz, "Architectural Views of Aspects," Proc. 2nd Int'l Conf. on Aspect-oriented software development, pp.1-10, March 17-21, 2003.

[Kazman 98] R. Kazman, M. Klein, M. Barbacci, "The Architecture Tradeoff Analysis Method," ICECCS, Aug 1998.

[Kiczales 97] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, et. al., "Aspect-Oriented Programming," Proc. ECOOP, Springer-Verlag, 1997.

[Kiczales 01] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold, "An Overview of AspectJ," Proc. 15th European Conf. on Object-Oriented Programming, pp.327-353, June 18-22, 2001.

[Marzolla 04] M. Marzolla, "Simulation-Based Performance Modeling of UML Software Architectures," PhD Thesis, Dipartimento di Informatica, Università Ca' Foscari di Venezia, Feb 2004.