Evaluating Complexity of Aspect-Oriented Software Development Comparing to Use Case Driven Software Development

KIATSOONGSONG Weerayut (0910204)

School of Information Science,
Japan Advanced Institute of Science and Technology

August 10, 2011

Keywords: Aspect-Oriented Software Development, Use Case Driven Software Development, Metric, Crosscutting Concerns.

1 Backgroud and Research Purpose

Use Case Driven Software Development (UDSD) is an approach to develop software system by using use cases to capture functional requirements and drive the whole development process. However, there are two problems occur after realizing all the use cases that make components of a use case cannot be kept separate from components of other use cases. First, scattering is a situation which codes that realize a particular use case are spread across multiple components. Second, tangling is a situation which some components contain the implementation to satisfy different use cases.

A concern is some part of the problem that we want to treat as a single conceptual unit. Sometimes, a concern affects more than one component and a component contains parts of multiple concerns. These situations are called scattering and tangling respectively. These kind of concerns are called crosscutting concern. As a result, it reduces comprehensibility, ease of evolution and reusability of software artifacts. Therefore, these concerns should be separated. In UDSD, use case is a kind of functional concern and some use cases are crosscutting concerns.

Aspect-oriented programming (AOP) was proposed to solve crosscutting concerns problem at code level by providing the means to separate code that implements crosscutting concerns and modularize it into aspects. With this concept, Aspect-oriented software development (AOSD) with use cases was proposed by Ivar Jacobson as a holistic approach to develop software systems with aspects for the whole development process. In addition to the use of aspects in AOP, use-case slice is used to encapsulate the classes and parts of classes that are specific to a specified use case.

Our research proposed the way to evaluate how much AOSD helps increase the maintainability and how much AOSD helps reduce the effect of crosscutting of the UDSD system. In order to evaluate UDSD and AOSD, software metrics are important materials to extract the characteristics of a software system. Our metrics are product metrics because we measured from artifacts of the system.

2 Research Approach

First, in order to compare UDSD system and AOSD system, we need mechanisms or rules to normalize them into the same level of abstraction. Figueiredo, E. et al. proposed a generic concern-oriented meta-model of the structural abstractions defined for aspect-oriented system [1]. They defined abstract structure of system in terms of system elements and how they relate to concern. This meta-model structure is abstract enough to be instantiated for different modeling and programming languages, so in our research, we instantiated this meta-model structure for UDSD and AOSD system.

Second, we defined the change impact metrics suite in order to evaluate how AOSD helps increase the maintainability of the system. Maintainability means the ease with which a software system or component can be modified, so it directly relates to change. In UDSD and AOSD development process, developers create some artifacts based on UML diagrams. The diagram consists of components and relationships between components. When change occurs, the system after change can be divided into four parts which consist of components and relationships;

- No change part — the part which is not related to the effect of the change

- Added part — the part which is added because of the change
- Modified part — the existing part which has been modified
- Removed part — the part which has been removed

The impact of change is the number of components and relationships in the last three parts and the entire system is the number of components and relationships in all parts. The degree of change impact I is the fraction of change impact divided by the entire system.

Third, we applied the scattering, tangling, and crosscutting metrics suite proposed by Conejero J. et al. [2] in order to evaluate the effect of crosscutting concerns in the system. First, they defined the meta-model of crosscutting pattern which explain that the system consists of sources and targets and sources and targets have the traceability relationships between them. In our research, we instantiated this meta-model by using use cases as sources and modules (classes and aspects) as targets. According to this meta-model, they created the dependency matrix which describes traceability dependency between use case and module. Then, they defined metrics to measure scattering, tangling and crosscutting of the system by calculating from this dependency matrix.

3 Empirical Study and Results

In order to validate our metrics suites, we applied them to ATM system which is introduced in [3]. We created two separate ATM systems implemented by UDSD and AOSD from requirements to design phase. Moreover, we added change to both systems and refined them according to the change. Then, we measured each of our metrics from both systems and statistically analyzed the results.

The results for change impact metrics suite show that there is statistically significant difference between AOSD system and UDSD system and can refer that AOSD is more maintainable than UDSD. However, the difference is quite small.

The results for scattering, tangling, and crosscutting metrics suite show that there is no statistically significant difference between AOSD system and AOSD system. Therefore, we cannot conclude that AOSD has less effect of crosscutting concerns than UDSD.

Although AOSD is said to be an approach to help increase maintainability and reduce effect of crosscutting concerns in UDSD, our results show that there is almost no difference between both approaches. This is because in the real world some classes contain not only the parts (methods and attributes) that fulfill different use cases and are not related to each other, but also the parts that are used by many use cases, which we called common parts. AOSD provides non-use-case-specific slice to contain these common parts. As a result, use case still has the dependency to the base class in common parts and some classes still has relationships to the base classes, so when the change occurs, it can also affect these common parts. Therefore, the efficiency of AOSD is hindered by the use of non-use-case-specific slices. However, if we remove the use of non-use-case-specific slices out of the system, it reduces the reusability of the system and increases the system size because of the common parts. Consequently, we have to consider the trade-off between separation of concerns and reusability.

4 Conclusion and Future Works

In this thesis, we proposed the way to evaluate UDSD and AOSD systems in terms of maintainability and effect of crosscutting concerns. First, we defined metrics suite called change impact metrics suite to evaluate the maintainability of the system. Second, we applied scattering, tangling and crosscutting metrics suite proposed by Conejero J. et al. to evaluate the effect of crosscutting concerns of the system. Then, we applied our metrics suites to our case study; ATM system.

The results show that the efficiency of AOSD is hindered by the use of non-use-case-specific slices to contain parts used in common by many use cases. If we remove the use of non-use-case-specific slice, we can see great efficiency of AOSD in increasing maintainability and reducing effect of crosscutting concerns in UDSD system. However, we have to consider the trade-off between the reduction of crosscutting concerns and reusability.

As future works, we will apply our metrics suites to more systems, especially the bigger and realistic systems to see more different viewpoints and validate our metrics suites. Moreover, we will explore more about the efficiency of AOSD over UDSD in the separation of nonfunctional concerns from functional concerns and the separation of platform-specific concerns.

5 Reference

1. Figueiredo, E. et al. "On the Maintainability of Aspect-Oriented Software: A Concern-Oriented Measurement Framework". Proc. of European Conf. on Soft. Maint. and Reeng. (CSMR). Athens, 2008.

2. Conejero, J., Figueiredo, E., Garcia, A., Hernandez, J., Jurado, E. "Early Crosscutting Metrics as Predictors of Software Instability". In 47th International Conference Objects, Models, Components, Patterns (TOOLS), 2009.

3. Hassan Gomaa. "Designing Concurrent, Distributed, and Real-Time Applications with UML". Addison-Wesley, Object Technology Series, 2000.