Vol.2011-SE-174 No.9
2011/11/2

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

Kiatsoongsong Weerayut[†1] Camargo Cruz Ana Erika[†1]
Koichiro Ochimizu[†1]

Aspect-oriented software development (AOSD) with use cases proposed by Ivar Jacobson is said to be an approach which helps increase maintainability and reduce the effect of crosscutting concerns of the system implemented by Use Case Driven Software Development (UDSD). However, there still has no evidence to prove efficiency of AOSD over UDSD. This paper proposes 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.

1. Introduction

Nowadays, use case driven software development (UDSD) has broadly been used in software industry [1]. However, during the use case realization, there occur two problems; scattering and tangling. Scattering is a situation that the codes that realize a particular use case are spread across multiple components of the system. And tangling is a situation that each component in the system contains the implementation to satisfy different use cases. As a consequence, use cases cut across the system and then use cases are not kept separate from each other [2]. In software engineering, these problems are called crosscutting concerns and separation of concerns problem[4].

Over the past decades, aspect-oriented programming (AOP) has been used in order to modularize crosscutting concerns at the implementation phase [3]. Then, Ivar Jacobson proposed aspect-oriented software development (AOSD) with use cases to complement the concept of AOP. AOSD is a holistic approach to developing software systems with aspects and is said to be the approach that helps reduce the effect of crosscutting concerns of UDSD. As a result, the software systems built by AOSD is said to have more maintainability than those built by UDSD.

Although AOSD has a possibility to improve problems in UDSD, but there are still no evidence yet, so we have to define proper metrics to evaluate complexity of systems implemented by both approaches. Then, we can compare those two systems implemented in different approaches according to the measures of the defined metrics.

This paper reports the results of our research on the evaluation of AOSD complexity in comparison to UDSD complexity to see how much AOSD can help improve maintainability and reduce the effect of crosscutting concerns in UDSD. In order to evaluate these two approaches, we proposed one metric suite called the change impact metrics suite and we applied metrics suite proposed by Conejero J. et al. [6] to our research.

This paper is divided into the following sections: section 2: Background, we present a brief explanation of background knowledge needed to carry out our research. Section 3, 4, 5: Our Approach, we present our approach used in our research in order to achieve our objectives. First, we describe about how to compare these two approaches. Then, we explain about metrics suites used in our research; change impact metrics suite and crosscutting concerns metrics suite. Section 6: Empirical Study and Results, we present the case study and the results to validate our metrics. Section 7: Conclusion and Future Works, we draw out our conclusions and describe our plans for future works.

2. Background

2.1 use case driven software development(UDSD)

Use Case Driven Software Development (UDSD) is an approach to develop software system by using use cases. Use cases drive the whole development process since most

†1 Japan Advanced Institute of Science and Technology (JAIST)

activities are performed starting from use cases. This leads to the increasing of system's understandability and maintainability [1]. UDSD process can be divided into five phases and artifacts are created in each phase as shown in the table 1.

表 1 UDSD process and artifacts

Phase	Artifacts
Requirements	use case diagram and use case description
Analysis	class diagram and collaboration diagram
Design	class diagram and collaboration diagram
Implementation	source code
Test	test case and test results

After developers define the use cases in requirements phase, they define the components and their relationships according to each use case. This activity can be called use case realization. In this activity, use cases should have been separated from each other, but on the other hand, there are two problems occur. These problems are:

- Tangling - a situation that the codes that realize a particular use case are spread across multiple components of the system.
- Scattering - a situation that each component in the system contains the implementation to satisfy different use cases.

The example of these two problems is shown in Figure 1. In this figure, Hotel management system has three use cases; reserve room, check in customer, and check out customer. After use case realization, there are seven components in the system. For reserve room use case, there are four components spread across the system. Consequently, scattering occurs in this system. Moreover, for room component, there are three parts in order to fulfill the three use cases. This is called tangling [2].

2.2 Aspect-Oriented Software Development (AOSD) with Use Cases

Aspect-oriented programming (AOP) is a programming paradigm that gives developers the means to separate code that implements crosscutting concerns and modularize it into aspects [3]. However, we need a holistic approach to develop software systems with aspects from requirements phase. This is aspect-oriented software development (AOSD). Ivar Jacobson and Pan Wei NG applied use case concept to AOSD. They use

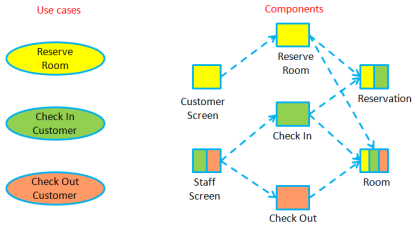

図 1 Tangling and Scattering Example [2]

the use cases as a representation of concerns. Moreover, they proposed the concept of use-case slice that is used to separate the use cases from each other.

With the concept of aspects, we can separate some features of classes into separate building blocks and separate extension features from the base features. Similar to this concept, Ivar Jacobson and Pan Wei NG proposed the concept of use-case slice to preserve the separation of concerns though the use case realization and implementation [3]. Use-case slice is a modularity unit that collates the specifics of a use case during use case realization. Each use-case slice collates parts of classes, operations and so forth, that are specific to a use case in a model. The task of composing these parts is left to some composition mechanisms provided by AOP.

The use-case slice of hotel management system is shown in Figure 2. The horizontal axis shows the element structure identifying the classes in the system. The vertical axis shows the use case structure. It identifies the use cases being realized with a different shade. Each horizontal row depicts a use-case slice containing the extensions of classes needed to realize the use case for that row. Thus, we have the *ReserveRoom* use-case slice, the *CheckInCustomer* use-case slice, and the *CheckOutCustomer* use-case slice. Each use-case slice contains partial classes specific to the use case realization. If we want complete class definitions, all we need to do is merge all the use-case slices.

AOSD process is the same as UDSD process but in analysis and design phase, we create use-case slice instead of class diagram.

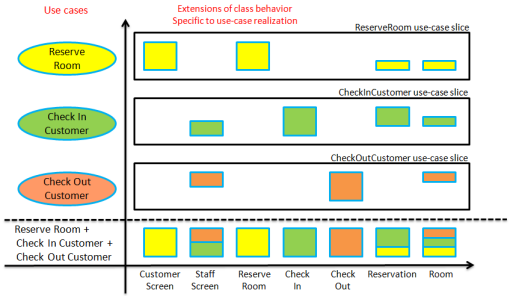

図 2 Use-case Slice Example [2]

3. How to Compare UDSD and AOSD

Because AOSD added new constructs to the systems; aspects, advices and intertype declarations. In order to compare systems implemented by different approaches, 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 [7] as shown in Figure 3. An aspect-oriented system consists of a set of components. A component has an interface, a set of attributes, a set of operations and a set of declaration. An operation consists of a return type, a set of parameters, a pointcut expression, and a set of statements. A declaration can also have a pointcut expression. On top of the structure, we can define concerns. A concern is not an abstraction of a modeling or programming language, such as components and operations. However, a concern can be considered as an abstraction which is addressed by those elements that have the purpose of realizing it.

This structure is abstract enough to be instantiated for different modeling and programming languages. In our research, we focus on the systems implemented by UDSD and systems implemented by AOSD. Therefore, we instantiate this meta-model structure for UDSD and AOSD as shown in Table 2. For example, the instantiation for system element is UDSD system and AOSD system for UDSD and AOSD respectively. For concern element, we refer to use case in both approaches. The component in UDSD

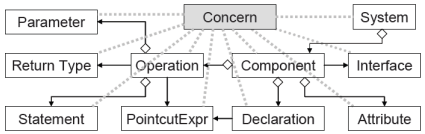

図 3 Abstract Meta-Model of Aspect-Oriented Systems [7]

is class or interface. But in AOSD, there are class, interface and aspect as components.

表 2 Meta-Model Instantiation for UDSD and AOSD Systems

Element	UDSD	AOSD
System	UDSD System	AOSD System
Concern	Use Case	Use Case
Component	Class and Interface	Class, Interface, and Aspect
Interface	Method Signature	Method Signature
Attribute	Class Variable and Field	Class Variable, Field, and Intertype Attribute
Operation	Method and Constructor	Method, Constructor, Intertype Method and Constructor, and Advice
Declaration	-	Pointcut and Declare Statement

4. Change Impact Metrics Suite

One of our objectives is to evaluate how AOSD approach improves maintainability of UDSD systems. Maintainability means the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment [8]. From this definition, the maintainability directly relates to the change to the system. Therefore, we propose the metrics suite to measure how system is affected by the change.

4.1 Components and Relationships

In UDSD and AOSD process, there are products created during each phase. These products are created in form of UML diagrams. In fact, there is the same characteristic amongst those diagrams. The diagram consists of components and relationships between components. We can conclude the components and relationships in each diagram as shown in Table 3.

表 3 Component and Relationship of System's Diagrams

Approach	Diagram	Component	Relationship
UDSD	Use Case Diagram	Use Cases	Use Case Associations
	Class Diagram	Classes	Class Associations
	Collaboration Diagram	Classes	Method Calls
AOSD	Use Case Diagram	Use Cases	Use Case Associations
	Use-Case Slice	Classes and Aspects	Associations of Class and Class, Class and Aspect, and Aspect and Aspect
	Collaboration Diagram	Classes and Aspects	Method Calls and Intertype Operation and Advice Calls

4.2 Change Impact Metric Definition

When the change occurs, software systems have to be modified. The system S after modifying can be divided into four parts;

- Added Part - the part that new components and new relationships are introduced to the system. It consists of components, *Add(c)* and the relationships, *Add(r)*.
- Modified and Derived Part - the part that existing components and relationships have to be modified or reorganized because of the change. It consists of components, *Mod(c)* and the relationships, *Mod(r)*.
- Removed Part - the part that components and relationships have been removed from the system after modifying the system to deal with change. It consists of components, *Rem(c)* and the relationships, *Rem(r)*.
- No Change Part - the part that is not modified or removed from the system by the change. It consists of components, *Noc(c)* and the relationships, *Noc(r)*.

The change impact metric is a metric that measures how much the system is affected by the change comparing to the whole system. The detail of change impact metrics suite is shown in Table 4.

For the change impact metric, we can apply this metric to measure the impact of change in the level of each diagram defined earlier in Table 3 or for the entire system by counting all the components and relationships from every diagram created from requirements, analysis and design phase.

表 4 Change Impact Metrics Suite

Metric	Equation	Description
$Imp(c)$	$= Add(c) + Mod(c) + Rem(c)$	impact of change on components
$Imp(r)$	$= Add(r) + Mod(r) + Rem(r)$	impact of change on relationships
$Sys(c)$	$= Add(c) + Mod(c) + Rem(c) + Noc(c)$	components in the entire system
$Sys(r)$	$= Add(r) + Mod(r) + Rem(r) + Noc(r)$	relationships in the entire system
I	$= \frac{Imp(c) + Imp(r)}{Sys(c) + Sys(r)}$	degree of change impact

5. Scattering, Tangling, and Crosscutting Metrics Suite

In our research, one of our objectives is to evaluate the reduction of crosscutting concerns in AOSD over UDSD. Conejero J. et al. proposed metrics suite for crosscutting concerns as predictors of software instability [6]. We applied this metrics suite in our research because it is well-defined and relevant to our objective.

5.1 Identification of Crosscutting

Conejero J. et al. defined the dependency matrix to trace the dependency between use case and module. For example, in a software system, there are 5 use cases and 6 modules. The dependency of this system is shown in Table 5. 1 in a cell means that class of the corresponding column contributes to addresses use case of the corresponding row. On the other hand, 0 means there is no dependency between class of the corresponding column and addresses use case of the corresponding row. If we consider the row of dependency matrix, we can trace the dependency from use case to module and can count the number of scattering for each use case. For example, use case 1 has dependency to module 1 and 4, so the number of scattering is equal 2. And if we consider the column of dependency matrix, we can trace the dependency from module to use case and can count the number of tangling for each module. For example, module 1 has dependency to use case 1, 2 and 3, so the number of tangling is equal 3.

The crosscutting product matrix is obtained through the multiplication of dependency matrix and transpose of dependency matrix. The crosscutting product matrix shows the quantity of crosscutting relations as shown in Table 6. In Table 6, we show the result of multiplication of dependency matrix in Table 5 and transpose of it. This

表 5 Example of Dependency Matrix

		Module					
		m[1]	m[2]	m[3]	m[4]	m[5]	m[6]
Use Case	uc[1]	1	0	0	1	0	0
	uc[2]	1	0	1	0	1	1
	uc[3]	1	0	0	0	0	0
	uc[4]	0	1	1	0	0	0
	uc[5]	0	0	0	1	1	0

matrix is used to derive the final crosscutting matrix as shown in Table 7. A cell in the final crosscutting matrix denotes the occurrence of crosscutting, but abstracts the quantity of crosscutting. In the crosscutting matrix, the diagonal cells are set to be zero because a use case cannot crosscut itself.

表 6 Example of Crosscutting Product Matrix

		Use Case				
		uc[1]	uc[2]	uc[3]	uc[4]	uc[5]
Use Case	uc[1]	2	1	1	0	1
	uc[2]	1	3	1	1	1
	uc[3]	0	0	0	0	0
	uc[4]	0	1	0	1	0
	uc[5]	1	1	0	0	2

表 7 Example of Crosscutting Matrix

		Use Case				
		uc[1]	uc[2]	uc[3]	uc[4]	uc[5]
Use Case	uc[1]	0	1	1	0	1
	uc[2]	1	0	1	1	1
	uc[3]	0	0	0	0	0
	uc[4]	0	1	0	1	0
	uc[5]	1	1	0	0	0

In our research, we use the union of all class diagrams from all use cases in UDSD and the union of all use-case slices from all use cases in AOSD as materials to create dependency matrix.

5.2 Metrics for Scattering, Tangling and Crosscutting

NScattering of a use case s_k is the number 1's in the corresponding row (k) of the dependency matrix:

$$NScattering(s_k) = \sum_{j=1}^{|T|} dm_{kj} \tag{1}$$

Where $|T|$ is the number of modules and dm_{kj} is the value of the cell [k,j] of the dependency matrix. This metric measures how scattered a use case is. This *NScattering* metric is normalized to obtain a value between 0 and 1. Then, *Degree of scattering* of the use case s_k is defined as:

$$Degree\ of\ scattering(s_k) = \begin{cases} \frac{\sum_{j=1}^{|T|} dm_{kj}}{|T|} & \text{if } \sum_{j=1}^{|T|} dm_{kj} > 1 \\ 0 & \text{if } \sum_{j=1}^{|T|} dm_{kj} = 1 \end{cases} \tag{2}$$

The closer to zero this metric is, the better encapsulated the use case is. In order to measure the scattering of the system, *Global scattering (GScattering)* is the average of the *Degree of scattering* values for each use case:

$$GScattering = \frac{\sum_{i=1}^{|S|} Degree\ of\ scattering(s_i)}{|S|} \tag{3}$$

Where $|S|$ is the number of use cases.

NTangling for the module t_k are defined, where $|S|$ is the number of use cases and dm_{ki} is the value of the cell [k,i] of the dependency matrix:

$$NTangling(t_k) = \sum_{i=1}^{|S|} dm_{ik} \tag{4}$$

This metric measures the number of use cases addressed by a particular module. Similar to the scattering metrics, we defined *Degree of tangling* and *GTangling* to represent normalized tangling for module t_k and global tangling, respectively:

$$Degree\ of\ tangling(t_k) = \begin{cases} \frac{\sum_{i=1}^{|S|} dm_{ik}}{|T|} & \text{if } \sum_{i=1}^{|S|} dm_{ik} > 1 \\ 0 & \text{if } \sum_{i=1}^{|S|} dm_{ik} = 1 \end{cases} \tag{5}$$

Vol.2011-SE-174 No.9
2011/11/2

$$GTangling = \frac{\sum_{j=1}^{|T|} Degree\ of\ tangling(t_j)}{|T|} \quad (6)$$

Metrics for crosscutting cosist of *Crosscutpoints*, *NCrosscut* and *Degree of crosscutting* extracted from the crosscutting product matrix and the crosscutting matrix. The *Crosscutpoints* metric for a use case s_k is the number of modules where s_k is crosscutting to other use cases. The *Crosscutpoints* for s_k is the value of the cell in the diagonal of the row k or cell [k,k] ($ccpm_{kk}$) in the crosscutting product matrix.

$$Crosscutpoints(s_k) = ccpm_{kk} \quad (7)$$

The *NCrosscut* metric for the use case s_k is the number of use cases crosscut by s_k calculated by the addition of all cells of the row k in the crosscutting matrix:

$$NCrosscut(s_k) = \sum_{i=1}^{|S|} ccm_{ki} \quad (8)$$

From *Crosscutpoints* and *NCrosscut*, the *Degree of crosscutting* of a use case s_k is the normalization between 0 and 1, so that the use case with lower value for this metric is better modularized.

$$Degree\ of\ crosscutting(s_k) = \frac{Crosscutpoints(s_k) + Concerns\ crosscut(s_k)}{|S| + |T|} \quad (9)$$

6. Empirical Study and Results

6.1 Case Study: ATM System

In our research, we apply our metrics to ATM system which is introduced in [9]. In the requirements phase, we create the use case model in the same way for both UDSD and AOSD as shown in Figure 4. Then, in analysis and design phase, in UDSD, we create class diagram and collaboration diagram. But in AOSD, we create use-case slice and collaboration diagram based on use-case slice. To measure change impact metric, we need to apply some changes to the system. In ATM system, we make new requirement by adding new use case to the system.

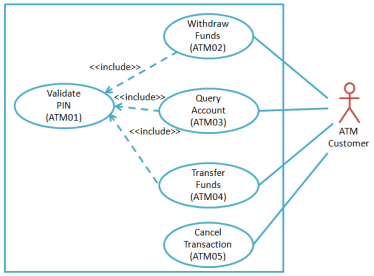

図 4 ATM System Use Case Diagram

6.2 Measures of Change Impact Metrics and Statistical Analysis

We measured the change impact metrics by collecting the number of components and relationships in each diagram and calculated the degree of change impact metric for each diagram. Moreover, for the overall impact of the change, we collected components and relationships in every diagram and calculated the metric. The results of the measurement of the degree of change impact I are shown in Table 8. According to the results, we can see that the measures of AOSD are lower than UDSD, so we can conclude that the maintainability of AOSD is higher than UDSD.

Since, only from raw data, it is not enough to lead us to the conclusion, so we analyzed these data statistically using T-test. The t-test is a statistical data analysis procedure to test whether or not the two independent populations have different mean values [10]. The final result of t-test is the p-value that is the probability value of a t-test. If the p-value is less than 0.05, we conclude that we found a statistically significant difference between the two groups.

According to results of t-test calculation, the p-value is equal 0.0109 which is less than 0.05, so we can conclude that our results for degree of change impact I metric can be used to identify the difference between the ATM systems implemented by UDSD and AOSD in term of maintainability. In this case, the AOSD can help increase maintainability on UDSD according to the raw data of measurements and the difference is significant according to the statistic analysis.

表 8 Measures of Change Impact Metric

Phase	Diagram	UDSD			AOSD		
		Imp(c)+ Imp(r)	*Sys(c)+ Sys(r)*	*I*	*Imp(c)+ Imp(r)*	*Sys(c)+ Sys(r)*	*I*
Requirements	Use Case Diagram	3	10	**0.300**	3	10	**0.300**
Analysis	Class Diagram/ Use-Case Slice	26	76	**0.342**	24	86	**0.279**
	Collaboration Diagram	33	116	**0.284**	30	121	**0.248**
Design	Class Diagram/ Use-Case Slice	32	98	**0.327**	30	117	**0.256**
	Collaboration Diagram	40	141	**0.284**	38	151	**0.252**
Overall		134	441	**0.304**	125	485	**0.258**

6.3 Measures of Scattering, Tangling, and Crosscutting Metrics and Statistical Analysis

For this metrics suite, we use the class diagrams in UDSD and the use-case slices in AOSD from analysis and design phase as materials to trace the dependency between use cases and modules.The measures of scattering, tangling, and crosscutting metrics for both UDSD and AOSD system at analysis and design phase are shown in Table 9.

表 9 Measures of Tangling Metric

Metric	UDSD		AOSD	
	Analysis	**Design**	**Analysis**	**Design**
GScattering	0.45	0.52	0.35	0.42
GTangling	0.34	0.32	0.21	0.20
Average of *Degree of crosscutting*	0.44	0.49	0.34	0.39

Moreover, we did t-test for *GScattering*, *GTangling* and average of *Degree of crosscutting* to prove whether the differences between the measures have statistical significance or not. The results of t-test show that p-values of all metrics are in between 0.08 - 0.13 which are higher than 0.05. Therefore, we can conclude that the differences between measures of UDSD system and AOSD system are not statistical significant. Therefore, we cannot conclude that AOSD has less effect of crosscutting concerns than UDSD.

6.4 Effect of AOSD Characteristic on Our Results

Our results of measurement show that the differences of AOSD and UDSD are quite small. After observations, we conclude that the efficiency of AOSD is lower than we expected because of the use of non-use-case-specific slice in AOSD. Ideally, in a tangled component or component that contains several parts fulfilling different use cases, these parts are not related as shown in Figure 1. AOSD puts the specific parts in aspects and use-case slice as shown in Figure 2. However, practically, in tangled component, these parts have some methods that are used in many use cases which we call “common parts”. The practical case for crosscutting concerns is shown in Figure 5. For example, in the *Room* class, there is a method called *retrieve()* to retrieve data of a room. This method is used in common for all use cases. Therefore, it is put in the common parts.

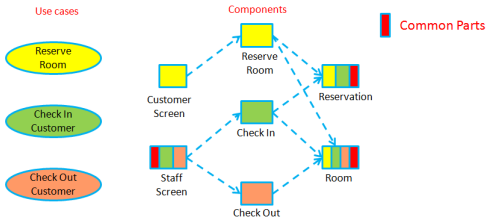

図 5 The Practical Case for Crosscutting Concerns

AOSD provides non-use-case-specific slice(NUCS) to manage the common parts. When we realize use cases, if there are methods that are used in many use cases, we put these methods in NUCS slice. As a result, for scattering, tangling, and crosscutting metrics, the use case still has the dependency to the base classes[⋆1] and also has a dependency to an aspect. Consequently, use cases are more scattered than the ideal

⋆1 The base class means a class that is used to fulfill a certain use case but it is also used by other use cases. After applying aspects to encapsulate the parts that are specific to a certain use case, there are still base classes for aspects to refer and merge the specific parts together.

case of crosscutting concerns. Some classes that have common parts are still tangled because use cases use the common parts, and then there still has dependency relationship between use case and class. Moreover, use cases cut across each other because they use the same methods in common parts. For the change impact metrics, some classes still have dependency to the base classes to access the common parts' methods and also have dependency to an aspect. Therefore, when change occurs, the change can affect the base classes, and then the number of components and relationships in modified part will increase from the ideal case of crosscutting concerns.

To sum up, the use of NUCS slice hinders the efficiency of AOSD on the improvement of maintainability and the reduction of crosscutting concerns of UDSD. If we ignore the use of NUCS slices, we have to duplicate the common methods and put the common parts in the aspect in each use-case slice. The results measured from the ATM system implemented by AOSD without NUCS slice are much lower than AOSD with NUCS. Moreover, the t-test results show the p-values of all metrics are a lot lower than 0.05. This means that the differences between measures of UDSD and AOSD without NUCS have definitely statistical significances. However, ignoring the use of NUCS and duplicating the common parts into each aspect lead to lower reusabiliity of the system.

7. Conclusion and Future Works

In this paper, in order to compare UDSD and AOSD, we proposed change impact metrics suite to evaluate the maintainability of the system and we applied scattering, tangling, and crosscutting metrics suite proposed by Conejero J. et al. to evaluate the effect of crosscutting concerns in the system.

The results of empirical study show that AOSD system is more maintainable and has less effect of crosscutting concerns than UDSD system but with small differences. This is because of the occurrence of common parts which AOSD provides non-use-case-specific(NUCS) slice to contain them. 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. Therefore, the efficiency of AOSD is hindered by the use of NUCS slices. However, if we remove the use of NUCS slices, it reduces the reusability of the system.

Our plans for future works, we will apply our metrics to more case studies because the case study that we used in our research was derived from textbook, so it is not the example from the real projects. Moreover, for the change impact metrics, the complexities of all components and relationships are not equal, so we should consider the complexity of them and refine our change impact metrics. Additionally, we will explore more about other approaches in AOSD proposed by Ivar Jacobson such as the approach for separating nonfunctional concerns from the functional concerns and the approach for separating platform-specific concerns from non-platform-specific concerns.

参 考 文 献

1) Ivar Jacobson, Grady Booch, and James Rumbaugh. "The Unified Software Development Process". Addison Wesley Longman Inc, 2000.
2) Ivar Jacobson, and Pan-Wei Ng. "Aspect-Oriented Software Development with Use Cases". Pearson Education Inc, 2004.
3) Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier and John Irwin. "Aspect-Oriented Programming". Computer Science Volume 1241/1997, pp. 220-242, 1997.
4) Tarr, P. et al. "N Derees of Separation: Multi-Dimensional Separation of Concerns". Proceedings of the 21st International Conference on Software Engineering, May 1999.
5) Everald E. Mills. "Software Metrics". SEI Curriculum Module SEI-CM-12-1.1, December 1998.
6) 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.
7) 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.
8) D.M. Coleman, D. Ash, B. Lowther, and P.W. Oman. "Using Metrics to Evaluate Software System Maintainability". Computer, vol. 27, no. 8, pp. 44-49, Aug. 1994.
9) Hassan Gomaa. "Designing Concurrent, Distributed, and Real-Time Applications with UML". Addison-Wesley, Object Technology Series, 2000.
10) Simon Hurst. "The Characteristic Function of the Student-t Distribution". Financial Mathematics Research Report No. FMRR006-95, Statistics Research Report No. SRR044-95, 1995.