Testability Analysis of Aspect Oriented Software

Sushil Garg, Dr. K.S.Kahlon and Dr.P.K.Bansal

***Abstract*—Design for testability is an important issue in software engineering. Measuring and assessing the testability during the analysis or development phase of software engineering would help in planning testing activities. Testability becomes crucial in case of Aspect oriented software system (AOS), where control flows are generally distributed over the whole architecture because of the static structure and dynamic behavior of Aspect. This paper presents a generic and extensible measurement framework for Aspect oriented software testability. We identify a set of design attributes that are helpful to measure the testability of AOS. The goal of this framework is to provide structured guidance for facilitating empirical research on testability and point out parts of the design that need to be improved to reduce the final testing effort.**

***Index Terms*—Software testability, Software Testing, Aspect Oriented Programming, Software design.**

I. INTRODUCTION

As the size of software application increases it becomes more complex. Effective testing is therefore required to achieve adequate level of software quality and reliability. Any technique that improves a software design at an early stage can have highly beneficial impact on the final testing cost and quality. Still Aspect Oriented system is not enough matured field a lot of efforts are required for optimal testing. Testing is not only depends on human factors, test techniques and test tools but also on characteristics of the software development artifacts. To maximize the impact of testing, we need to design systems so that their testability is optimal. The degree to which a software artifact facilitates test tasks in a given test context is called testability [1]. ISO defines testability as a attribute of software that bear on the effort needed to validate the software product. To test any model or component one must be able to control its input and analysis to output. Software testability based on probability of faults in the software [2].

In this paper our goal is twofold:

1) To help measuring and assessing testability in a practical manner, with a focus on the analysis and design stages of Aspect Oriented development.
2) To define hypothesis to guide future empirical research on testability.

Software testability is an external software attribute which is used to evaluate the complexity and the effort required for software testing. Testability is an impotent quality characteristic of software. For improving the testing process we need to optimize the testability. Some quality metric can be used to locate part of software which contributes to a level of testability. We know effective testing of the software can improve the quality and reliability of the software [3]. But testing is a major cost driven factor during design, development & maintenance of the software.

In this paper we mainly focus on design phase of the software [4]. The testability analyses can maximum effective during the design phase. Testability analysis during the analysis and design phase helps us in taking design decisions to improve testability before implementation starts. Testability is a very important issue in software engineering. Testability is external software attribute the complexity and the effort required for software testing. In AOP aspects interact and change the behavior of the base system so it is very difficult to pin point the area of code design directly or indirectly affected by aspects in its dynamic behavior of overall system. Design for testability is an important issue in software engineering. Measuring and assessing the testability of early in the analysis development phase of software engineering would help in planning testing activities and improving the design. Testability becomes essential in case of Aspect Oriented software system [5]. Where control flows are generally distributed over the whole architecture and the aspect have on the static structure with dynamic behavior of the overall system.

This paper presents generic and extensible measurement framework for Aspect-Oriented software testability. We identify some design attributes that are helpful to measure testability of AOS. The goal of this framework is to provide structured guidance for facilitating empirical research on testability. And point out parts of the design or code that need to be improved, driving structured modification to reduce the final testing effort. A lack of testability contributes to a higher testing cost and testing effort. It becomes crucial in the case of Aspect Oriented Software system design (where control flows are generally distributed over the whole architecture) due to large impact that aspects have on the static structure and dynamic behavior of the overall system.

In aspect oriented programming aspect spreads through the program automatically and generates internal executions details at run time. We have defined a Probe based testing mechanism, Goel et. A [6] that observing internal execution detail at different level of abstraction unit, integration and system levels, during testing of aspect oriented programming. Probes are pre built in the software during the design phase.

Sushil Garg is Assistant Professor & Head, CSE Dept., RIMT-Institute of Engineering & Technology, Mandi Gobindgarh, Punjab,india. (email: sushilgarg70@gmail.com).

Dr. K.S. Kahlon is Professor & Head of CSE Dept., Guru Nanak Dev University, Amritsar, Punjab,india. (email: karanvkahlon@yahoo.com).

Dr. P.K. Bansal is ex-Principal, MIMIT, Punjab, india. (email: pkbmimt@yahoo.com).

IACSIT International Association of Computer Science and Information Technology WWW.IACSIT.ORG

During testing, probes are externally activated or deactivated, facilitating visual display of execution details. The internal execution details displayed during testing consist of the classes and aspect of the module invoked, value of parameters or messages at entry/exit, and hierarchy of execution of classes and aspects. In aspect oriented software, coverage of inheritance hierarchy as well as dynamic binding relationship is needed to ensure proper testing o these relationships. During testing of inheritance hierarchy, the modification o subclass requires testing of subclass and retesting of inherited method of super class. In same way during modification of aspect, require testing all the code in which aspect is directly or indirectly related.

II. Software testability attributes

We describe the attributes that are part of the framework and the hypotheses establishing their relationship to testability.

We first define the details of testing levels.

- Unit testing
- Integration testing
- System testing

Regression testing is applied at different levels of testing because regression testing ensue that when we modifying a software system, no side effects are introduced that would result in failure of unchanged, previously working functionality. Our aim at having an exhaustive coverage of all design attributes that have an impact on testability. Now we focus our work on the following effort-intensive testing sub-activities:-

1) *Chose test cases*

This activity list all tasks that aim at defining the specification of test cases based on software artifacts such as specification & design.

2) *Developing Drivers*

This activity consists of writing the required code to execute test cases.

3) *Developing Oracles*

This consists in writing the code required to assess whether a test case execution is successful.

4) *Metrics*

Now we define metrics that help to answer the questions described in the previous section:

First metric is that

Average Aspect Dependence (AAD)

$AAD= 1/n \times \sum_{i=1}^{n} ADi$

Where n is number of Aspect in software, ADi is Aspect dependency of Aspect i, the number of aspects the Aspect i Depends on directly and indirectly.

The extent to which we have to take into account dependence Aspects during test case design increases with the overall number of dependence Aspect. For every activity and sub-activity we identify design attributes that have an impact on testability. A hypotheses is describes the cause effect relationship between attributes and testability. Then identify attributes and deriving the hypotheses is based on a through and systematic review of the literature and our own experience in performing testing experiments. All hypotheses are described systematically by listing the impacted testing activities and sub-activities and the reasons for the impact, these hypotheses will be further tested and the theory refined over time. [8]

Hypotheses 1

By increasing the size of non-overridden inherited features, unit testing may be required to build and execute test. The cost of unit testing directly depends upon the size of inherited features as more effort and time is required to build and execute test cases.

Hypotheses 2

In client server model, the higher the size of the inheritance hierarchy rooted by the server class, test are more expensive due to the dynamic dependences between client class and server classes. Testing the interface is expensive then testing a inheritance.

Hypotheses 3

Mainly unit testing techniques are based on exercising different sequences of operations, like based on sequential constraints, data flow, encapsulating unrelated operation will increase number of test cases and a more complex driver. Unit testing of Aspect Oriented software is different from procedural or object oriented software. Unit testing is to test each unit of the software to verify that the detail design for the unit has been correctly implemented. This approach uses a combination of these two types of testing to test aspects and classes in aspect oriented software.

Hypotheses 4

More number of paths will lead to more test cases to specify and longer paths will lead to more complex drivers.

Hypotheses 5

Direct and indirect dependencies of a component under test on its dependence components have many effects on the time and effort needed for testing as well as on the complexity of the test tasks. Additional stubs have to be implemented if the cycles shall be broken. A layered design without dependency cycles is therefore better testable.

Hypotheses 6

In Aspect oriented software difficulty is there of maintaining programs in the face of crosscutting concerns. Khalid, Peter found that maintaining crosscutting concerns is difficult due to invisibility of the crosscutting logic, invisibility of the crosscutting will increase the maintenance cost. These types of invisible tight coupling crosscutting affect the testability [7,9].

Hypotheses 7

The polymorphism is a feature, which showed a new technical challenge to tester. We know the method calls with polymorphism, because of the dynamic binding characteristics, the program code of the actual execution cannot be predict [10]. It will be dynamic and decided at the run time. So it is very difficult to observe and track executed path in test processes from the test perspective, the aspect oriented programming with polymorphism is hard to produce an adequacy test case because we don't know which type of object will be executed in runtime and it is difficult to analyze the testability of source code in static test. Jin-ChrngLin and Yun-Liang suggest the metric of polymorphism for measure in design stage [11].

III. Testability Measures

This section defines a set of potential measure for different attributes. In this paper we consider only potentially useful measures. All the measures are listed below.

A. *Complexity Measures*

In testability many attributes depend on the complexity of the expression, which help in designing of Aspect Oriented Modeling (AOM) diagram. Following we discuss some important points that can be related to testability.

- Number of loops that lead to iterative operations.
- Iterative expressions are directly proportional to the number of nodes, i.e. IE $\propto$ n, for sending any message or a sequence of operations and to execute the sequence of operations.

B. *Interface Complexity & Use Case Model for AOM*

Sequence numbers require handling of sequence interleaving and loops, otherwise, the number of sequence is either very large or sometimes may touch infinity.

Heuristically, we consider one possible interleaving for one independent sequence pair taking one loop exactly once. In our example, we consider use case open acc for which there exist two message conditions:

Condition 1: actid -> exists

Condition 2: not actid -> exists

As seen from the activity diagram (Figure 1) & the above heuristics we obtain the following possible sequences:

- 8 1 6 4 5 3 7 10 9
- 8 1 6 4 2 5 3 7 10 9
- 8 1 6 4 2 5 4 3 7 10 9

In the sequence diagram (Figure 2), for use case open acc, there are 2 scenarios, 9 messages & 3 classifiers.

These measures like, Total number of declared attributes, Number of overloaded operations, Number of inherited operations, Number of inherited attribute, Number of inherited dependency relationships for Unit Size attribute. Number of non-complaint preconditions, ratio of non complaint preconditions, Number of non-complaint invariants, ratio of non complaint invariants, Pair of inherited and overridden operation interactions, Depth and width of inheritance hierarchy are for Inheritance design properties. For Unit cohesion we measure lack of cohesion measure and Ratio of cohesive interactions measures [12,13]. Unit coupling can be measure by coupling between objects, response for class/aspect, class method interaction, class-aspect interaction and dynamic aspect coupling.

TABLE 1 MEASURES FOR TESTABILITY

Attribute	Measure
Unit size	• Number of classes and aspects. • Dynamic and static behavior of the aspect on the system. • Mapping of point cut and advice. • Number of inherited operations.
Unit Cohesion	• Class interaction. • Interaction among the advice and the method. • Interaction among the aspects and classes. • Complexity of interaction between class and aspect. • Complexity of a path in a class interaction.
Unit Coupling	• Number of paths between classes. • Number of messages between aspects. • Inter-procedural aspect control flow. • Interaction among aspect and modules. • Inheritance complexity. • Complexity of a path going through an inheritance hierarchy.
Use case structure	• Number of messages in sequence diagrams. • Number of scenarios in sequence diagrams. • Number of classifiers in sequence diagrams.
Interface complexity	• Number of use case. • Number of factors. • Number of operations. • Number of messages per actor.

C. *Model for testability:*

1. Selected testing activity
2. Select relevant attribute
3. Select measure
4. Develop a list of final testability measure

IV. Improving Design Testability

Improving testability of a software design means avoiding interactions between objects and parallel calls to common objects. A solution may be devised to clarify the design to make the code according to the requirements of the designer. This can be easily achieved with the use of empty interface classes, which is not possible for all the cases, by arranging the classes in a step wise manner to avoid big-bang interaction and minimize the number of stubs.

V. Conclusion

The features like Dynamic binding of Aspect oriented software introduce new types of errors, resulting in some testing issues of AO software to be different from the conventional software testing issues. One reason is that there are many potential factors that can affect testability. In order to

IACSIT
International Association of Computer Science and Information Technology
WWW.IACSIT.ORG

handle the testing issues of AO software, the conventional software testing techniques require improvisation or new techniques be developed. Testability of software is defined as ease of performing testing. The purpose of this paper is to increase our understanding of what makes code hard to test. Our approach is based on an extensive survey of the literature on software testability. In this paper we provide a set of hypotheses that explain its expected relationship with testability. This is important as an hypothesis can help decide, in a design and testing strategies. From a research viewpoint, this paper presents a number of precise hypotheses that can be investigated through empirical means. Such type of framework should help focus research efforts and motivate precise research question.

REFERENCES

[1] IEEE press," IEEE standard Glossary of software Engineering Technology" ANSI/IEEE standard 610.12-1990,1990.

[2] J.M. Voas, K.W. Miller, Software Testability: The new verification, IEEE Software 12 (3) (1995) 17-28.

[3] "A measurement framework for object-oriented software testability" Samar Mouchawrab, Lionel C. Briand , Yvan Labiche Software Quality Engineering Laboratory, Carleton University, 1125 Colonel by drive, Ottawa, Canada K1S5B6 Available online 4 November 2005

[4] L.C. Briand and Y.Labiche,"A UML- Based Approach to system Testing ", Software and system Modeling, vol.1(1),pp 10-42,2002.

[5] Filman,R., Elrad, T., Clarke, S,Akisit ,M., Aspect oriented software development. Addison- Wesely,2004.

[6] Goel A, Gupta, S.C, & Wason, S.K,2003 Probe Mechanism for object-oriented software testing in proceedings of FASE, LNCS 2621,Springer,Warsaw,Poland

[7] Testing Aspect Oriented Programs; an Approach based on the coverage of the Interaction among advices and methods.

[8] PDL introduces a type a type system to detect meaningless pointcuts and modify the semantic to provide a more appropriate behaviour.

[9] Bruno Harbulot, Johan R. Gurd., " A Joinpoint for loops in Aspect J", AOSD 06, March 20-24, 2006, Bonn Germany.

[10] Client Morgan, Kris De Volder, Eric Wohlstadter " A static Aspect Language for checking Design Rules" AOSD' 07 March 12-16 Vancouver Canada.

[11] A new method for estimating the testability of polymorphism in class hierarchy by Jin-ChrngLin and Yun-Liang Hung in Int. Computer Symposium, Dec 15-17,2007, Taipei, Taiwan.

[12] Suitability of object and aspect oriented languages for software maintenance by Khalid Al-Jasser, Peter Schachte, Ed Kazmierczak, The university o Melourne. (ASWEC'07) 2007 IEEE.

[13] The selection of join point or join point Designation Diagram should be independent from aspect oriented programming languigage.

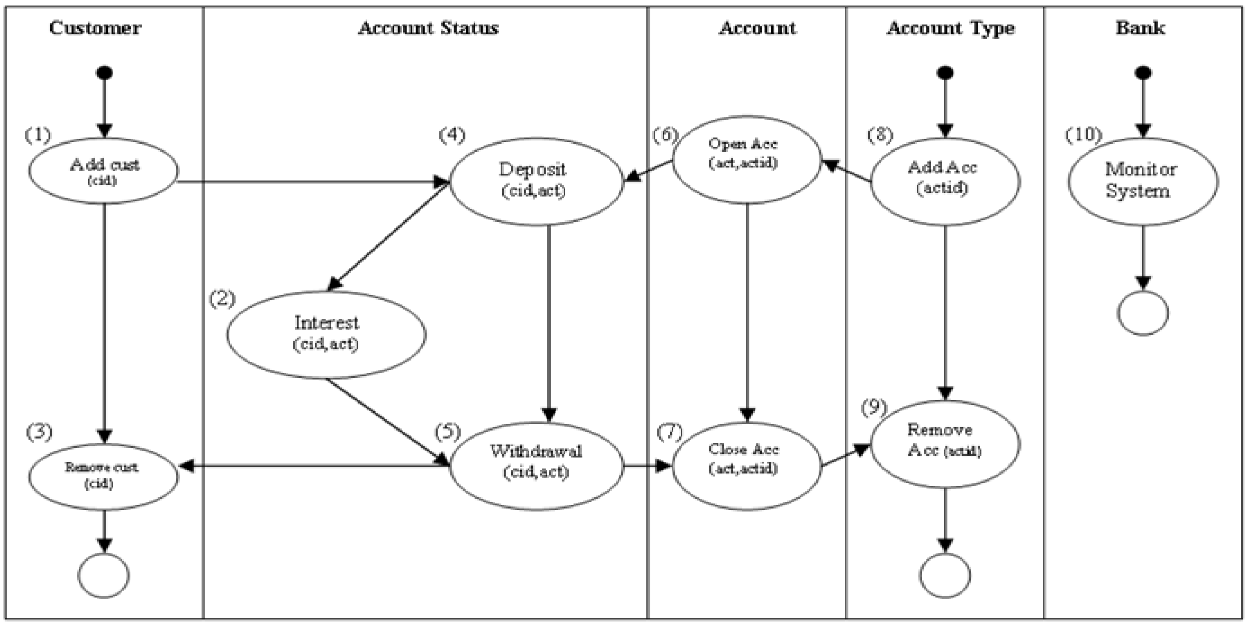

Fig.1 Activity Diagram for Open Account

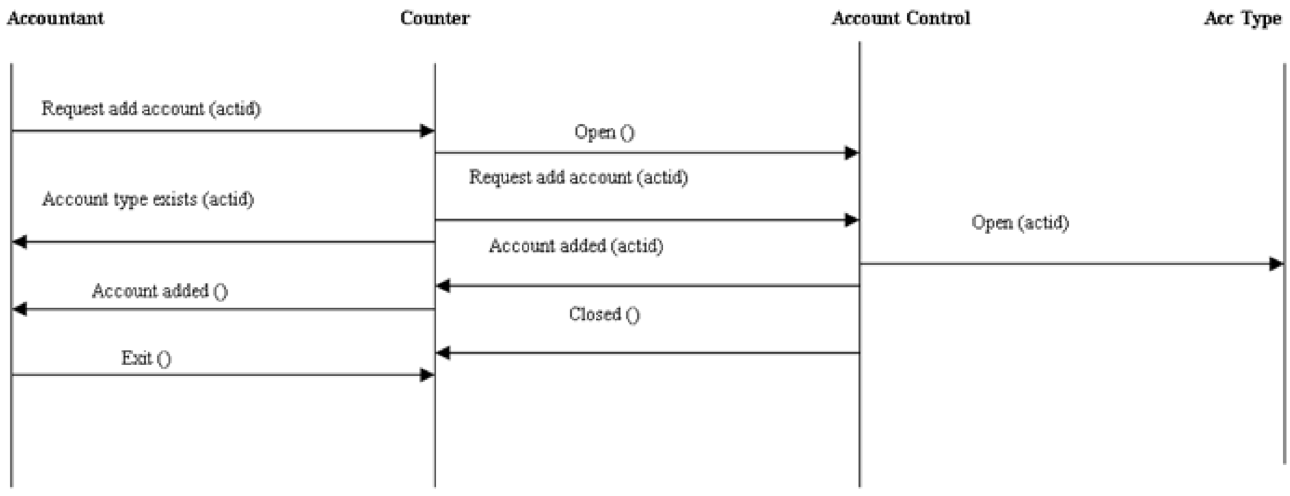

Fig.2 Sequence Diagram for Open Account.