Testing of Aspect Oriented Programming with UML sequence flow diagrams

Amritpreet Kaur[#1], Janpreet Singh[*2]

[#]*Mtech Scholar, Computer Science Department, Lovel professional University*
Project.1198@gmail.com

[*]*Asst.Profssor, Computer Science Department, Lovely Professional University*
janpreet.s@gmail.com

***Abstract*— Aspect-Oriented Programming is a software engineering paradigm that offers new constructs, such as join points, pointcuts, advices, and a spects in order to improve separation of crosscutting concerns. The new constructs bring new types of programming faults with respect to crosscutting concerns, such as incorrect pointcuts, advice, or aspect precedence. In fact, existing object-oriented testing techniques are not adequate for testing aspect-oriented programs. As a result, new testing techniques must be developed. In this paper, an approach based upon UML activity diagram for testing aspect-oriented programs is presented. The proposed approach focuses on the testing of Aspect oriented programs with UML activity diagram including activity and sequence diagrams from UML process. Particularly for testing we will use AGRO UML tool. In our research we will focus on faults finding for Aspect Oriented Programs with help of flow diagram based on activity and sequence of UML.**

***Keywords*— Aspect oriented programming, Unified modeling language, Sequence diagram, Agro UML, Aspect j, Activity diagram.**

I. INTRODUCTION

Aspect-Oriented Programming is a software engineering paradigm that offers new constructs, such as join points, point cuts, advices, and aspects in order to improve separation of crosscutting concerns. The new constructs bring new types of programming faults with respect to crosscutting concerns, such as incorrect point cuts, advice, or aspect precedence. In fact, existing object-oriented testing techniques are not adequate for testing aspect-oriented programs. In Figure 1, we can see the structure of Aspect Oriented Language.

Aspect Oriented Programming (AOP) is an emerging discipline in Software Engineering. AOP is a programming paradigm which isolates secondary functions from the main program's logic. The definition given by Gregor Kiczales "Modular units that cross-cut the structure of other modular units." The central idea of AOP as an emerging discipline of post-object technology is to provide strong support to the separation of the repeated, scattered or entangled concerns at every stage of software development, introducing a new modular unit to encapsulate them to facilitate extensibility, changeability and reuse.ite. In the context of software engineering a concern is defined as a property or interest point of a system. Concerns, from the system point of view, are defined as those interests belonging to the system and its operation, or other aspects which are critical or important for the stakeholders. That is to say, a concern is a kind of requirement needed by the system. Some concerns can be easily encapsulated within classes or modules, according to the chosen implementation language; however, others whose functionality affects several modules, are called crosscutting concerns and they are not easy to separate. They cannot be easily encapsulated into new functional units as "implicit functionality" because they crosscut the whole system and are implemented in many classes or modules producing an entangled or scattered code, difficult to understand and maintain. The goal of AOP is to encapsulate them into a modular unit, called aspect, to handle these requirements at implementation level. AOP is an extension of Object oriented programming and it is a future programming technique

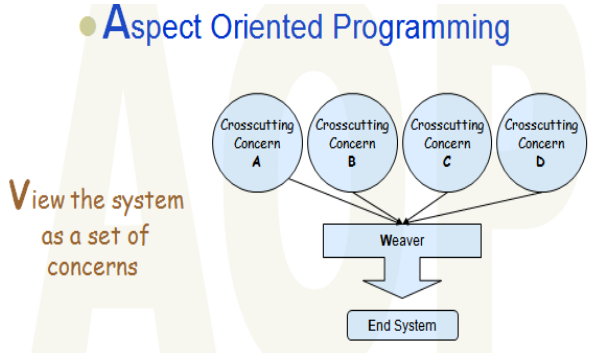

Fig. 1 Structure of Aspect Oriented programming

A use case illustrates a unit of functionality provided by the system. The main purpose of the use-case diagram is to help development teams visualize the functional requirements of a system, including the relationship of "actors" (human beings who will interact with the system) to essential processes, as well as the relationships among different use cases. Generally When we use use to develop Aspect programming the we face different type of complexity which could also leads to uneven behavior of programs which leads to the requirement of testing of the Aspect Oriented programming with unified modeling language with various operations of oriented langauge.

II. UNIFIED MODELING LANGUAGE DIAGRAMS

A use case illustrates a unit of functionality provided by the system. The main purpose of the use-case diagram is to help development teams visualize the functional requirements of a

system, including the relationship of "actors" (human beings who will interact with the system) to essential processes, as well as the relationships among different use cases. Use-case diagrams generally show groups of use cases -- either all use cases for the complete system, or a breakout of a particular group of use cases with related functionality (e.g., all security administration related use cases) [1] [2]. To show a use case on a use-case diagram, you draw an oval in the middle of the diagram and put the name of the use case in the center of, or below, the oval. To draw an actor (indicating a system user) on a use-case diagram, you draw a stick person to the left or right of your diagram (and just in case you're wondering, some people draw prettier stick people than others). Use simple lines to depict relationships between actors and use cases, as shown in Figure 2.

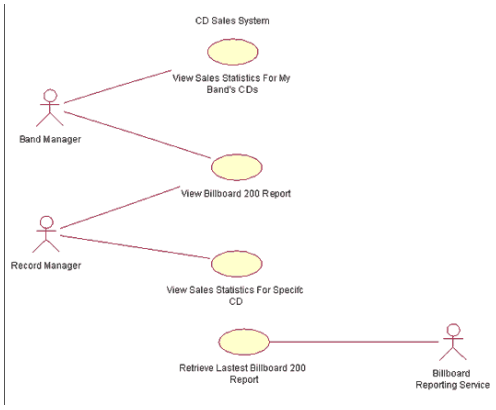

Fig 2 Relationship in UML diagram [1]

A use-case diagram is typically used to communicate the high-level functions of the system and the system's scope. By looking at our use case diagram in Figure 1, you can easily tell the functions that our example system provides. This system lets the band manager view a sales statistics report and the Billboard 200 report for the band's CDs. It also lets the record manager view a sales statistics report and the Billboard 200 report for a particular CD. The diagram also tells us that our system delivers Billboard reports from an external system called Billboard Reporting Service [1][2].

.

A. Class Diagram

The class diagram shows how the different entities (people, things, and data) relate to each other; in other words, it shows the static structures of the system. A class diagram can be used to display logical classes, which are typically the kinds of things the business people in an organization talk about -- rock bands, CDs, radio play; or loans, home mortgages, car loans, and interest rates [2]. Class diagrams can also be used to show implementation classes, which are the things that programmers typically deal with. An implementation class diagram will probably show some of the same classes as the logical classes' diagram. The implementation class diagram won't be drawn with the same attributes, however, because it will most likely have references to things like Vectors and Hash Maps [2]. A class is depicted on the class diagram as a rectangle with three horizontal sections, as shown in Figure 2. The upper section shows the class's name; the middle section contains the class's attributes; and the lower section contains the class's operations (or "methods").

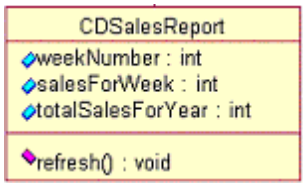

Fig 2 Sample class object in a class diagram [2]

B. Activity Diagram

Activity diagrams show the procedural flow of control between two or more class objects while processing an activity. Activity diagrams can be used to model higher-level business process at the business unit level, or to model low-level internal class actions. In my experience, activity diagrams are best used to model higher-level processes, such as how the company is currently doing business, or how it would like to do business [2]. This is because activity diagrams are "less technical" in appearance, compared to sequence diagrams, and business-minded people tend to understand them more quickly. An activity diagram's notation set is similar to that used in a state chart diagram. Like a state chart diagram, the activity diagram starts with a solid circle connected to the initial activity [3].

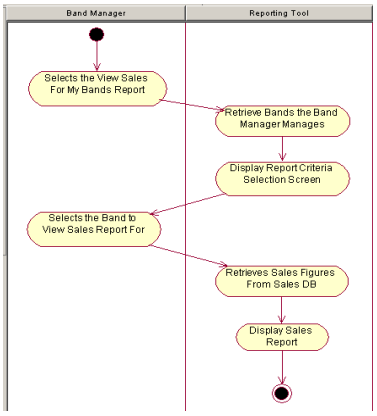

Figure 3: Activity diagram, with two swim lanes to indicate control of activity by two objects: the band manager, and the reporting tool [3].

The activity is modeled by drawing a rectangle with rounded edges, enclosing the activity's name. Activities can be connected to other activities through transition lines, or to decision points that connect to different activities guarded by conditions of the decision point. Activities that terminate the modeled process are connected to a termination point (just as in a state chart diagram) [3]. Optionally, the activities can be grouped into swim lanes, which are used to indicate the object that actually performs the activity, as shown in Figure 3.

III. Components of Aspect Programming

Testing of aspect oriented programs is a new programming paradigm. Many researchers had contributed their research in the field of testing AOP. Mutation testing is an emerging area of research in testing of aspect oriented programming. The effectiveness of mutation testing depends on finding fault types and designing of mutation operators on the basis of faults identified. Therefore the effectiveness of testing depends upon the quality of these mutation operators. We already have the mutation operators for procedural and object oriented languages, but for aspect oriented language only a few researchers have contributed.

C. Components

Crosscutting Concern: These are the aspects of a program which affect other concerns. For eg. If writing an application for handling medical records, The bookkeeping and indexing of such records is a core concern while logging a history of changes to the record, database or user database or an authentication system would be crosscutting concern.

Join Points: These are well defined points in the execution of a program. For eg. Execution of a method call is a join point.

Point Cuts: A point cut picks out join points. Join points are described by point cut declaration. Point cuts can be defined in classes or in aspects and can be named or be anonymous.

Advice: Advice is code that executes at each join point picked out by a point cut. Advice is a function.

Introduction: An introduction is a member of an aspect but it defines or modifies a number of another type or class. With introduction we can add method to an existing class, add fields to existing class and implement an interface in an existing class.

Aspect:- The combination of point cut and advice.

D. Metrics

WOM: Weighted operations in module counts number of operations or methods in a given module. The number of operations and the complexity of operations involved is a predictor of how much time and effort required to develop and maintain the module. Modules with large number of operations limit the possibility of reuse.

DIT: Depth of inheritance of a class is its depth in the inheritance tree, if multiple inheritance is involved. Maximum path from the node representing the class to the root. The deeper a module is in the hierarchy, the greater the number of operations it is likely to inherit, making it more complex to predict its behavior.

NOC: Number of children is a number of immediate subclasses or sub-aspects of a given module. Greater the number of children then greater the reuse due to inheritance, improper abstractions of parent module, misuse of sub classing and requires more testing.

CDA: Crosscutting degree of an aspect is a number of modules affected by the point cuts and by the introductions in a given aspect. This gives overall impact, an aspect has on the other modules.

CMC OR CBM: Coupling on method call is a number of modules or interfaces declaring methods that are possibly called by a given module.

RFM: Response for a module is number of methods and advices potentially executed in response to a message received by a given module.

LCO: Lack of cohesion in operations is number of pairs of operations working on different class fields minus pairs of operations working on common fields. When all methods don't access to any field, then LCO = 0. Low cohesion increases complexity, thereby increasing the likelihood of errors during the development process.

CAE: Coupling on advice execution is a number of aspects possibly triggered by the execution of operations in a given module. There is a dependence of the operation from the advice.

CFA: Coupling on field access is a number of modules or interfaces declaring fields that are accessed by given module. CFA measures the dependency of given module on other modules but in terms of accessed fields.

IV. Proposed Work

A. Problem Definition

Aspect-Oriented Programming is a software engineering paradigm that offers new constructs, such as join points, pointcuts, advices, and aspects in order to improve separation of crosscutting concerns. The new constructs bring new types of programming faults with respect to crosscutting concerns, such as incorrect pointcuts, advice, or aspect precedence. In fact, existing object-oriented testing techniques are not adequate for testing aspect-oriented programs. As a result, new testing techniques must be developed. In our research, an approach based upon UML activity diagram based on the flow of diagrams for testing aspect-oriented programs will be presented.

Our research will focus on the testing of aspect programs by generating test sequences based on activity diagrams which will validate the correct working and starting malfunctioning (errors). Further aspect model will be generated and integrate with basic aspect model with flow activity diagrams in UML activity diagrams.

In particular we have done some experimentation on Aspect Oriented Programming by developing aspect class with enhancement to object oriented programming. We have define the following aspect instructions in our experimentation.

```
aspect Check{
  pointcut check(Handel h): target(h) &&
(execution (public * intToHex(..)) || execution
(public*floatToHex(..)));
```


Above we have defined the point cut for aspect programming.

B. Objectives

- To find the faults that specific to aspectual structures.
- Provide solution for incorrect advice type, strong or weak pointcut expressions, and incorrect aspect precedence.

C. Research Methodology

This research will focus on providing solution for said problem by testing the aspect oriented programs with activity flow diagrams.

This research will be focused on the Flow of the activity diagrams to test the aspect oriented codes. This flow will be used in accordance with the aspect oriented programs.

Our research will focus on the testing of aspect programs by generating test sequences based on activity diagrams which will validate the correct working and starting malfunctioning (errors). Further aspect model will be generated and integrate with basic aspect model with flow activity diagrams in UML activity diagrams.

Particularly for testing we will use AGRO UML tool. In our research we will focus on faults finding for Aspect Oriented Programs with help of flow diagram based on activity and sequence of UML. Flow for activity diagram will be decided according to example set on which we will work and the according to generated flow, testing of aspect codes will be considered.

V. CONCLUSIONS

Our experimentation is in process for finding faults in Aspect Oriented Programming with different UML styles. Particularly we have developed some codes and we have started our testing experimentation. In near future we will test the different metrics of Aspect Oriented Programming with UML.

.

REFERENCES

[1] http://www.uml.org -- The official UML Web site.

[2] http://www.rational.com/uml/resources/documentation/index.jsp --Offer several different versions of the actual UML specification.

[3] http://argouml.tigris.org --Information on Argo UML, an .open source UML modeling tool built in Java.

[4] Mayank Singh, Shailendra Mishra" Mutant Generation for Aspect Oriented Programs", Indian Journal of Computer Science and Engineering, Vol 1, No 4, pp 409-415, 2011.

[5] Somayeh Madadpour, Seyed-Hassan Mirian-Hosseinabadi, Vahdat Abdelzad," Testing Aspect-Oriented Programs with UML Activity Diagrams", International Journal of Computer Applications, Volume 33, No-8, pp 23-29, November 2011.

[6] Reza Meimandi Parizi, Abdul Azim Abdul Ghani, Rusli Abdullah, and Rodziah Atan," On the Applicability of Random Testing for Aspect-Oriented Programs", International Journal of Software Engineering and its Applications, Vol. 3, No. 4, October, 2009.

[7] Swati Tahiliani, Pallavi Pandit," A Survey of UML-Based approaches to Testing", International Journal Of Computational Engineering Research, Volume. 2, Issue. 5, pp 1396, September 2012.

[8] Philippe Massicotte, Linda Badri, Mourad Badri," Towards a Tool Supporting Integration Testing of Aspect-Oriented Programs", Journal of Object Technology, Vol. 6, No. 1, January-February 2007.

[9] Deepali A. Bhanage, Sachin D. Babar," Analyzing effect of Aspect Oriented concepts in design and implementation of design patterns with case study of Observer Pattern", International Journal of Engineering Research & Technology (IJERT), Vol. 1 Issue 3, May - 2012.

[10] http://www.omg.org/gettingstarted/what_is_uml.htm#12DiagramTypes and the dictionary http://softdocwiz.com/UML.html, Summary of the UML Diagrams, Originated May 9, 2002; Use case and sequence diagram info added Feb 3, 2005.

[11] Suresh Chand Gupta, Prof Ashok Kumar," Smart Environment for Component Reuse", International Journal of Advanced Research in Computer Science and Software Engineering, Volume 2, Issue 2, February 2012.

[12] Dr.R.V.Krishnaiah, Banda Shiva Prasad," Analysis of object Oriented Metrics", International Journal of Computational Engineering Research (ijceronline.com) Vol. 2 Issue. 5, pp 1474, September 2012.

