A Systematic Review of Testing Techniques for Aspect Oriented Programs

Shaukat Ali Khan, Abdulrahman shaheen, Lamia Shaheen
shaukatali74@hotmail.com, ayshaheen@hotmail.com, Lamia31@hotmail.com

Abstract— Software testing is the most important phase in software development life cycle. Different testing techniques have been developed for testing of software systems developed using different software methodologies. Aspect Oriented Programming (AOP) is a emerging software development methodology to model real world entities and separating crosscutting concerns into a separate module for reusability and ease of managing code . Aspect Oriented Programming complements objects oriented programming, not replacing it, by the addition of aspects in program for cross cutting concerns. As a result of new software development methodology testing techniques have been also developed to test Aspect Oriented Programs. This paper provides a comprehensive survey and analysis of testing technique developed for testing Aspect Oriented Software. We have performed analysis and evaluated testing methodologies based on our selected parameters.

***Keywords*— Software Testing, Aspect, Weaving, Point Cut, Join Point, Fault Model, Advice, AOP**

Introduction

Testing is the one of the important phase in the software development life cycle. The purpose of testing is to find errors in the software system. Testing is the process of executing a program with the intent of finding errors. The major purpose of software testing is to find errors.. A successful test case is one that finds an error. There are different levels of testing i.e. low level testing and high level testing. Low level testing includes unit and integration testing. Low level testing is mostly performed by the programmer. High level testing includes functional testing, system testing and acceptance testing etc. High level testing is mostly performed by quality assurance team. There are different types of testing strategies i.e. black box testing and white box testing. Black box testing validates functional requirements. Test cases are designed from the requirements specification of the system. . Black box testing is also called as functional testing. White box testing tests the program logic. Test cases are designed by using the source code of the program. White box testing is also called as structural testing [1][2].

AOP is a new emerging technology for software development by implementing cross cutting code as aspect. The aspect is like a class but contains code that affects multiple classes. Aspect is loosely coupled and can be integrated easily from one software program to other. AOP introduces new constructs like join point, point cut and advice. In OOP, basic principles are polymorphism, inheritance, modularity and encapsulation while in AOP the basic building blocks or language constructs are join points, advice, introduction, advice before and advice after. [15]

AOP introduces new constructs that are different from other programming paradigms. For testing AOP programs new techniques are required as existing techniques are unable to test the new constructs of AOP like join point, point cut, aspect etc. This paper is aimed to present the current research being done in the area of testing for AOP, their analysis based on selected parameters and considering the issues that arise in testing AOP. [14]

The rest of the paper is organized in the following order. Section 2 provides the background knowledge on AOP, Section 3 gives the overview of the some techniques developed for testing Aspect Oriented Software. Section 4 provides the comparisons of the testing techniques explain in section 3. Section 5 provides a conclusion and future work related to the testing of Aspect Oriented Software.

1. BACKGROUND

Aspect Oriented Programming (AOP) is a promising new technology that is based on separating crosscutting concerns into modules. Aspect oriented programming is a new technology for software development that support modularity, and reusability by separating cross cutting concerns into separate modules. In existing programming methodologies it is very hard to separate these cross cutting concerns into separate modules. AOP is basically an extension of OOP (object oriented programming) not a replacement of OOP. AOP works on the principle of OOP but only cross cutting concerns are separated into separate modules which are isolated from other code and classes and finally joins the other code at compile time in some AOP languages or runtime in some AOP (Aspect Oriented languages). [15]

In OOP basic principles are polymorphism, inheritance, modularity and encapsulation while in AOP the basic building blocks or language constructs are Join points, advice, introduction, advice before and advice after. Figure 1 shows

difference between weaving process in AOP and how it is different from OOP.[4][15]

In AOP crosscutting concerns are separately written in separate modules. These modules are called aspects. AOP provides modularity, reusability by modeling cross cutting concerns into separate aspects.

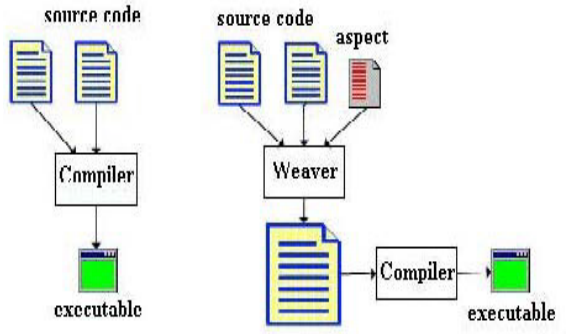

Figure 1. Weaving process in AOP

AOP code is easy to handle and more maintainable. If we have a function that is used for logging, we have to write logger everywhere in the software at the start and end of each module for this purpose we have to call our logger method for logging at the start and end of every function for proper logging. This is a cross cutting over code for logging at the start and end of every function in existing OOP while in AOP separate module for logging is written that is completely isolated from code and do logging before and end of every method call. This module is called Aspect and it joins the other code of the software at the time of weaving of aspect with remaining code that is mostly done at runtime or compile time through some weaver at some particular join point. [4][5]

AOP encourages more modularity then OOP so its code is easy to maintain, easy to debug and test. Basic construct of AOP are Pointcut, join point, advice and introduction. AOP is a technique which can be integrated with every language for modeling cross cutting concerns into separate modules. [3, 14, 15]

2.1 Aspect Oriented Fault Model

Aspect oriented fault model also identifies the requirement for aspect oriented language. There are six classes of aspect oriented faults [16]

- Incorrect strength of join points
- Incorrect aspect precedence
- Failure to establish expected post condition
- Failure to preserve state invariants
- Incorrect focus on control flow
- Incorrect changes in control dependencies

According to fault model design requirement for AOP language are as follows [16]

- Flexible join point language is needed
- Dynamic support at Aspect Activation level
- Reflection mechanism is extended for obtaining information about join points, point cuts and advice.
- Good control flow analysis
- Code weaver at load or runtime to avoid maintaining two versions of software.
- Join points for advices to monitor aspects.

Object Teams uses aspect as base for other aspects and explicit activation of aspect is controlled within the program and load-time weaver but it lack join point language . In AspectJ byte code weaving is used now for source code weaving. [15]

3. EVALUATION CRITERIA

This section presents the evaluation criteria for the analysis and evaluation of the covered techniques. Our analysis is based on parameters s whether a given techniques supports the selected parameter or not. Our selected parameters and their possible values are as follows:

3.1 Coverage

This parameter is used to identify the coverage of the technique whether the given technique covers all the constructs of AOP or it is capable to handle few. This parameter has a value of full or partial. If the technique covers all the constructs of AOP then it has a value of full otherwise it has a value of partial.

3.2 Supported AOP Construct

This parameter is used to identify the constructs of AOP that can be modeled by using given technique. This parameter contains the name of those constructs of AOP that can be modeled by using given technique as a value.

3.3 Complexity

This parameter is used to compute complexity of technique in terms of time, memory and usage. The technique is complex if it requires a lot of time and memory. If there are lot of steps in techniques, it is also considered as complex. This parameter has a value of high or low.

3.4 Case Study

This parameter is used to present whether a technique is evaluated through case study or yet it is not implemented on any case study. This parameter has a value of Yes or no.

3.5 Automatable

This parameter is used to present whether a technique is automatable or not. If all the steps in a technique are fully explained with inputs and outputs, then this parameter has a value of yes otherwise it has a value of no.

3.6 Modeling of weaving process

This parameter is used to identify the support of modeling of weaving process in the technique as dynamic or static.

3.7 Supporting language

This parameter is used to identify the current language support of the technique. This parameter contains the name of the supporting language as a value and if the technique is not supported by any language then it has a value of none.

3.8 OOP

This parameter is used to identify the reusability of OOP testing technique in AOP. If an approach is based on OOP testing approaches and it uses OOP testing approach as baseline, then this parameter has a value of Yes otherwise its value is no.

4. LITERATURE SURVEY

The following section contains survey of the existing testing techniques for AOP.

I. Slicing Aspect Oriented Software

Jianjun Zhao. [6] Proposed a technique for slicing aspect oriented programs using system dependence graph [4].

"A program slice is a set of program statements that contribute to or affect a value for a variable at some point in the programs".

A number of techniques have been developed for slicing Object Oriented software and procedural languages. As AOP is different from all other existing methodology for their specific features as joint point, advice and aspect so a new technique for slicing AOP system is developed which is based on aspect-oriented system dependence graph, which also extends the previous dependence graphs, to present aspect oriented software. The Aspect oriented system dependence graphs consist of three parts-[5, 6]. A system dependence graph for non-aspect code. A system dependence graph for aspect code. Some arcs to connect the graphs of aspect and non-aspect code.

On the basis of these graphs static slices of Aspect oriented programming can be computed efficiently.

Analysis:

This technique only computes static slice of a program one cannot compute the dynamic slice. This technique is complex and time consuming as slice is computed for aspect and non-aspect code and also for weaving point which requires a lot of hard work. No tool support for slicing AOP programs so slicing is time consuming. Currently this technique is only tested for AspectJ program so it is valid only for AspectJ program for other AOP languages we have to modify it according to language architecture.

II. JAOUT: Automated Generation of Aspect-Oriented Unit Test

This technique [7] was proposed by Guoqing Xu, Zongyuan Yang, Haitao Huang, and Qian Chen for unit testing of AOP (Aspect Oriented Programming). Aspect Oriented Programming addresses the problem of separation of concerns in programs which is well suited for unit test problems. AOP programmers build generic aspects to monitor certain cross cutting concerns. For testing purposes it is difficult to identify those aspects and use them as test oracles. Author uses the approach to collect the related aspect in one group and later on these aspects are used as oracles in some other application as at application level some application shares some common aspects e.g. aspects for logging etc .

This approach contributes in three ways towards the testing of AOP. Describes Aspect-Oriented Test Description Language (AOTDL). A tool JAOUT for translating from testing aspects to generic aspects in AOP. Double-phase testing to remove meaningless test cases.

Double phase testing is a two phase testing which is divided into three parts: making Operational Profile, pre test and final test. First test cases are dividing into several partitions. During first phase a small number of test cases from each partition are tested. Test suite run many times on each partition for counting actual number of meaningless test cases in each partition. In second phase a large number of test cases are taken for each partition according to the proportion of meaningless test case in first phase. First phase is the cost for second phase. This technique is suited for black box testing and also where test case space is very large. [3]

Analysis:

This technique is useful for removing meaningless test cases from the test space as double phase testing is used for reducing test cases space. Double phase testing itself uses two steps which is time consuming and costly for small test space but good for large test space. Double phase testing is good for

black box testing but for white box testing it is not applicable. This technique support reusability as related aspects are collected together and later on used as test oracles. It is not only for one AOP language we can apply it for all languages.

III. Testing Aspect Oriented Programming Pointcut Descriptors

The strategy [8] uses two approaches first to find the extra JPs selected by PCDs (point cut descriptors) and second to select intended JPs (Join points) that were not selected by PCDs. First approach is used to restrict the set and second approach is used to enlarge it. To verify a particular PCD1 is not selecting extra JPS for a particular advice a1 in specific program p1 structural integration criteria is used. This is done by gathering all JPs selected by PCD1 in particular program p1 and to check each one by integrating a1 with the method in which the JP is located. In this way PCD is restricted and JPS moves to smaller size as unintended JPS are removed.

In this approach statement coverage criterion is used to make sure that each and every statement of advice is executed at a particular JP. If JP is selected test passes else it fails. Stronger the coverage criteria more unintended JPs will be detected. This technique is also useful in finding bugs in advice which were not detected in unit testing [8].Detecting neglected JPs is more difficult task for this purpose structure testing technique is not useful as structured testing techniques are not able to detect missing JPs. For this purpose mutation testing is used, mutants are inserted into the program for simulation of programmers common mistakes and then bugs are find out. [8]

Mutants are inserted in the program so that a particular PCD select more JPs as commonly missed by programmers in this way neglected JPS are found. A common mistake is to neglect the JPS of a subclass for a particular super class. Tester realizes to pick the extra JP by the PCD at the time of creation of test cases for killing mutants from the programs. [8].

Analysis:

These two techniques are well suited for code based testing we cannot apply it to specification based testing as in specification we have only high level design. This technique is time consuming as we have to test every JPs and PCDs whether it is useful or not in that program. Mutation analysis is used which also takes time for proper mutation design and then removal of mutation from software. This technique useful as it optimizes the functionality and size of code.

IV. A Regression Tests Selection Technique for Aspect-Oriented Programs

During development of new components and maintenance of software, regression testing is used frequently to check the software functionality and behavior of new developed components. [9]

In this approach a framework RETESA (Regression Test Selection for Aspect Oriented program) is used for regression test selection.

There are five components of this Framework:

- Dynamic coverage recorder
- CFG comparator
- Safe Edge Identifier
- Candidate Test Selector
- Final Tests Selector

Dynamic coverage matrix takes the byte code of the old program and new aspect code and maps it into CFG and records a test along with it complete CFG path into a coverage matrix.

The comparators compare two graphs and identify the dangerous edges. Safe edge identifier takes two coverage matrixes and identifies the safe edges for each test. This components output two types of information, a set of test whose complete CFG path are effective equal and report for eliminated tests. Figure 2 explains the system architecture of the proposed approach with all major steps used in the technique and all components of the technique.

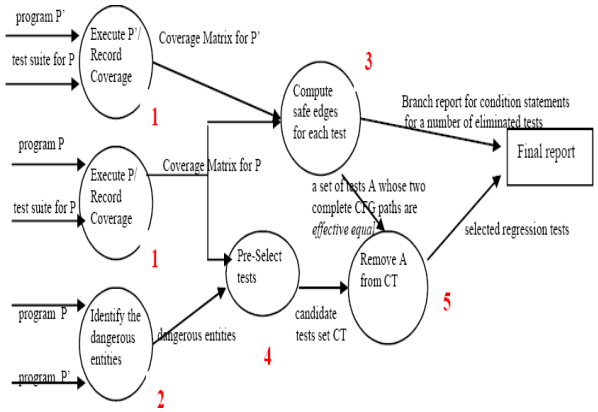

Figure 2. System architecture of RETESA framework

Candidate test selector takes input coverage matrix and dangerous edges and looks in coverage matrix for those tests which covers the dangerous edges. This component output the candidate tests for further inspection.

Final tests selector selects the required test and this set of test is represented to tester for testing.

The possible of effectiveness of this technique is based on existing classification systems.

Analysis:

In this paper, authors proposed a framework for regression selection of test cases of aspects oriented programs. The possible of effectiveness of this technique is based on existing classification systems. The technique is complex and required a lot steps for completion. The technique is fully automatable as each step is clearly defined with inputs and outputs. A case study is provided to check the validity of the approach.

V. State -Based Increment Testing of Aspect- Oriented Programs

Dianxiang Xu and Weifeng Xu proposed [10] an approach for state based incremental testing of aspect-oriented programs. In this approach authors addresses the following major issues, specification of the expected impact of aspects on object states for testing. To what extent can base class tests be reused for testing aspects? Detection of fault at aspect level rather than base classes. Aspect-orientation is incorporated into state models by following the concepts of AOP. In this approach, join points can be states, events, or variables in a state model, a point cut picks out a set of join points, advices are specified as a state model and an aspect is an encapsulated entity of pointcuts and advice model.

Aspect-oriented state model depends on the weaving mechanism that integrates aspect models into base models. As incremental modification to base models, an aspect may affect the base models in various ways such as adding new transitions, introducing new states, removing transitions, modifying guard conditions and introducing new events.

The general process of the approach to incremental testing is as follows: generate the state models of the base classes, generation of abstract test cases from the base model, instantiate the abstract test cases to form concrete test suites for the base classes, test the base classes, build aspect models and weave them into the base models, generate abstract test cases from the woven state models, generate test suites for the aspect-oriented program and test the aspect-oriented program. Test case generation is based on the modal class test design pattern for object-oriented programs.

Analysis:

Authors proposed an approach for state-based incremental testing of aspect oriented programs. The approach is based on the state model of object oriented programs and then aspects are added into the base model of object oriented programs. The approach detects the fault at aspect level incrementally. The approach is complex in sense that it requires a lot of steps for test case generation. A case study is provided for the proof of concept. The approach is well defined and can be easily automated. The approach provides full coverage to aspect oriented constructs for testing.

VI. Flow Graph for Testing Aspect-Oriented Programs

Weifeng Xu et al. Proposed [11] an approach for testing of aspect oriented programs using aspect flow graph. In this approach, a hybrid model known as aspect scope coverage model is produced by combining state models and flow graphs for producing test suites. The hybrid model contains the semantics of classes and aspects and it preserves completed information that is essential for revealing faults in aspect oriented models. The test-ready model contains enough information to produce test cases for its implementation automatically. This approach focus on understanding the interactions between classes and aspects and what interactions we are intending to test. Based on the Aspect Flow Graph (AFG) and the transition tree test suites can be derived using testing coverage testing criteria.

Analysis:

Authors proposed an approach for testing of aspect oriented programs using aspect flow graph. The approach is based on the hybrid model called aspect scope coverage graph. The approach detects the fault at aspect level using different interactions of aspect with other constructs of aspect oriented programs. The approach is complex in sense that it requires a lot of steps for test case generation. A case study is provided for the proof of concept. The approach is well defined and can be easily automated. The approach provides full coverage to aspect oriented constructs for testing.

VII. Data-flow-based unit testing of Aspect-Oriented Programs

This paper presents [12] an approach for unit testing of aspect oriented programs using dataflow analysis. This paper combines unit testing and data flow testing at aspect level for testing of aspect oriented programs. This paper presents an approach the test classes and aspects that may affect more than one aspect. For each class or aspect three level testing are performed. These levels are inter module level between different modules, intra module level within a module and Intra-aspect or intra-class testing is performed for modules that can be accessed outside the aspect or class and can be invoked in any order by users of the aspect or class. The selection of tests for aspects or classes is based on the computation of def-use pairs using control flow graph.

Analysis:

Authors proposed an approach for testing of aspect oriented programs using unit testing and data flow testing. The selection of tests for aspects or classes is based on the computation of def-use pairs using control flow graph. The approach detects the data flow fault at aspect level at unit level.. The approach is complex in sense that it requires a lot of steps for test case generation. A case study is provided for

the proof of concept. The approach is well defined and can be easily automated. The approach provides partial coverage to aspect oriented constructs for testing.

VIII. State-Based Testing of Integration Aspects

Weifeng Xu and Dianxiang Xu [13] have presented the state model based approach for testing integration aspects that group separate classes in a larger aggregate. A state-based model of integration aspect captures the expected interaction between base classes and integrated classes. Testing of an integration aspect is performed through the interface of its base classes. In this approach before testing integration aspect first testing of classes is done. The process starts with class modeling, which produces the state models of classes as well as specification of state predicates. State predicate defines the relationship between the abstract and concrete state. The process of testing aspect is similar to the process of testing classes and most of the concepts used in state based testing of classes are used in state based testing of aspects. Authors first describe the state based testing of classes and then state based testing of aspects.

Testing of integration aspects is done through testing of their base classes because aspect code is automatically weaved into base classes. Each test is a sequence of the construction of a base class object followed by method invocations to the object. This is similar to class testing. The timing case study is provided of the telecom domain for the validity of the proposed approach.

Analysis:

Authors proposed an approach for state-based integration testing of aspect oriented programs. The approach is based on the state model of object oriented programs and then aspects are added into the base model of object oriented programs. The approach detects the fault at aspect level. The approach is complex in sense that it requires a lot of steps for test case generation. A case study is provided for the proof of concept. The approach is well defined and can be easily automated. The approach provides full coverage to aspect oriented constructs, like join point, pointcut, advice, for testing. It is highly scalable technique for any type of requirements to accommodate AOP programs.

5. ANALYSIS MATRIX

Parameters	Coverage	AOP Construct	OOP	Case-Study	Complexity	Weaving mode	Supported language	Automatable
[6]	Partial	Joinpoint, Pointcut	Yes	Yes	No	Static	AspectJ	Yes
[7]	Partial	Joinpoint, Pointcut	Yes	Yes	No	Static	AspectJ	Yes
[8]	Partial	Joinpoint, Pointcut	No	No	Yes	Static	None	No
[9]	Full	Joinpoint, Pointcut, Advice	Yes	Yes	Yes	Static	AspectJ	Yes
[10]	Full	Joinpoint, Pointcut, Advice	Yes	Yes	Yes	Dynamic	AspectJ	Yes
[11]	Full	Joinpoint, Pointcut, Advice	No	Yes	Yes	Dynamic	No	Yes
[12]	Partial	Joinpoint, Pointcut	Yes	Yes	No	None	None	Yes
[13]	Full	Joinpoint, Pointcut, Advice	No	Yes	Yes	Dynamic	No	Yes

Table 1. ANALYSIS MATRIX

Table 1 presents the Analysis Matrix based on our evaluation criteria

6. CONCLUSION

In this paper, we focus a testing of emerging methodology for software development AOP. We first define the basic concept of AOP and how it handles the cross cutting concerns. We have discussed the various strategies for testing AOP software and we have also discussed fault model for AOP and its suggestion. Basic idea behind this paper is to provide overview of the various testing strategies with analysis.

Future work includes the extension object oriented testing techniques for AOP Software, so that these techniques are used for AOP testing with little modifications. Another approach is to develop common test techniques for AOP and OOP as these techniques are similar in many ways only differ at few points such as aspects, join points, point cuts and advice etc.

REFERENCES

[1] Robert V. Binder. Testing Object-Oriented Systems:Models, Patterns, and Tools. Addison-Wesley, 2000.Chapter 10.

[2] Jian-Hui. W, Chun-Zhang. B, *"Software Evolution with Feature-Oriented and Aspect-Oriented Programming,"* icicic, pp.460, 2008 3rd International Conference on Innovative Computing Information and Control, 2008,ISBN: 978-0-7695-3161-8.

[3] Grundy.J, *"Multi-perspective specification, design and implementation of software components using aspects",* International Journal of Software Engineering and Knowledge Engineering (2000) vol.10, No. 6.

[4] Colyer. A, Harrop. R, Johnson. R, Vasseur. A, Beuche. D, Beust. C, *" AOP Will See Widespread Adoption"* IEEE Software, vol. 23, no. 1, pp. 72-75, Jan./Feb. 2006, doi:10.1109/MS.2006.26.

[5] Mark Mahoney: "Modeling Crosscutting Concerns in Reactive Systems with Aspect-Orientation". Doctoral Symposium at MoDELS/UML 2005, Montego Bay Jamaica, October 2005.

[6] Jianjun Zhao; "*Slicing aspect-oriented software*" Program Comprehension, 2002. Proceedings. 10th International Workshop on , vol., no., pp. 251- 260, 2002 DOI: 10.1109/WPC.2002.1021346

[7] Guoqing Xu, Zongyuan Yang, Haitao Huang, Qian Chen. *"JAOUT: Automated Generation of Aspect-Oriented Unit Test* ".APSEC 2004 Proceeding, 11th Asia-Pacific software engineering conference November 30 - December 3, 2004, Busan, Korea.

[8] Otavio Augusto Lazzarini Lemos, *"Testing Aspect-Oriented Programming Pointcut Descriptors"*, Proceedings of the 2nd workshop on Testing aspect-oriented programs 2006, Portland, Maine July 20 - 20, 2006.

[9] Guoqing Xu , Atanas Rountev, "*Regression Test Selection for AspectJ Software*", Proceedings of the 29th International Conference on Software Engineering, p.65-74, May 20-26, 2007.

[10] Dianxiang Xu and Weifeng Xu, "*State-Based Incremental Testing of Aspect-Oriented Programs*", AOSD'06, March 20-24, 2006 Bonn, Germany.

[11] Cui. Z, Wang. L, Li.X, *" Aspect Flow Graph for Aspect Oriented Programs",* Proceedings of the 2009 ACM symposium on Applied Computing, Honolulu New York, NY, USA, 2009. Pages 430-437, ISBN:978-1-60558-166-8.

[12 J. Zhao, "*Data-Flow-Based Unit Testing of Aspect-Oriented Programs*," Proceedings of the 27th Annual IEEE International Computer Software and Applications Conference (COMPSAC'2003), pp.188-197, Dallas, Texas, USA, November 3-6, 2003.

[13] Dianxiang Xu and Weifeng Xu ,"*State-based testing of integration aspects*", WTAOP '06 Proceedings of the 2nd workshop on Testing aspect-oriented programs ACM New York, NY, USA ©2006 ISBN:1-59593-415-4 doi>10.1145/1146374.1146376

[14] Przybylek. A*," Separation of Crosscutting Concerns at the Design Level: an Extension to the UML Metamodel",* Proceedings of the International Multiconference on Computer Science and Information Technology, pp. 551–557 ISSN 1896-7094, ISBN 978-83-60810-14-9.

[15] AspectJ-Homepage. http://www.eclipse.org/aspectj/docs.php Access Date: 11-07-2011

[16] R. T. Alexander, J. M. Bieman and A. A. Andrews. "*Towards the systematic testing of aspect oriented programs*". Technical report CS-4-105, Colorado State University Fort Collius, Colorado, 2004.