Systematic Review on Testing Aspect-oriented Programs
- Challenges, Techniques and Their Effectiveness

Muhammad Amar and Kashif Shabbir
This thesis is submitted to the School of Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:
Author(s):
Muhammad Amar
Address: Framgangen 252:91, SE-41280 Goteborg, Sweden
E-mail: muhammadamar@gmail.com

Kashif Shabbir
Address: Folkparksvagen 18:06, SE-37240 Ronneby, Sweden
E-mail: shabirkashif@gmail.com

University advisor(s):
Goran Fries
School of Engineering, BTH Ronneby

School of Engineering
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

Internet : www.bth.se/tek
Phone : +46 457 38 50 00
Fax : + 46 457 271 25
Abstract

Aspect-oriented programming is a relatively new programming paradigm and it builds on the basis of object oriented programming paradigm. It deals with those concerns that cross-cut the modularity of traditional programming mechanisms and it aims at reduction of code and to provide higher cohesion. As with any new technology aspect oriented programming provides some benefits and also there are some costs associated with it.

In this thesis we have done a systematic review on aspect oriented software testing in the context of testing challenges. Detailed analysis have been made to show that how effective are the structural test techniques to handle these challenges. We have given the analysis of Aspect-oriented test techniques effectiveness, based on research literature.

Keywords: Aspect-oriented software testing, Software test technique effectiveness, Testing challenges, systematic review.
ACKNOWLEDGMENTS

We are thankful to God who provided us ability and wisdom which enable us to deliberately focus on our Master Thesis.

We are very grateful to our Thesis Supervisor Dr. Göran Fries for his scholarly guidance and for giving us time and suggestions during comprehensive discussions. We really admire his analytic capabilities due to which we got able to direct our efforts towards a young and emerging research area. We appreciate Dr. Fries for enhancing our knowledge in the relevant research area.

We would like to thank especially our thesis faculty reviewer Dr. Robert Feldt for his critical reviewing and for giving us worthy recommendations that helped us in constituting our basic work related to thesis proposal. We are also thankful to all our teachers who make us proficient by giving us plenty of relevant knowledge in all the courses we have studied at Blekinge Tekniska Högskola (BTH).

We are also grateful to all our friends and class fellows at BTH who have always backed-up us during our research work. We appreciate the moral support of our friends and class-fellows.

The major contribution in our success of-course is of our family members. Our special thanks are for our parents, brothers and sisters for scarifying their wonderful time by sending us abroad in the quest of knowledge. They are the best.
# TABLE OF CONTENTS

SYSTEMATIC REVIEW ON TESTING ASPECT-ORIENTED PROGRAMS ............................................. I

- CHALLENGES, TECHNIQUES AND THEIR EFFECTIVENESS ..................................................................................... I

ABSTRACT .................................................................................................................................................................. I

ACKNOWLEDGMENTS ............................................................................................................................................. II

TABLE OF CONTENTS .................................................................................................................................................. III

TABLE OF FIGURES .................................................................................................................................................... V

TABLE OF TABLES ....................................................................................................................................................... VI

1 INTRODUCTION ........................................................................................................................................................... 1

1.1 AIMS AND OBJECTIVES ............................................................................................................................................ 2
1.2 RESEARCH QUESTIONS ............................................................................................................................................. 2
1.3 EXPECTED OUTCOMES .......................................................................................................................................... 3
1.4 RESEARCH METHODOLOGY .................................................................................................................................. 3
1.5 RELATED WORK ...................................................................................................................................................... 3
1.6 STRUCTURE OF THESIS ......................................................................................................................................... 4

2 SYSTEMATIC REVIEW ............................................................................................................................................... 6

2.1 PLANNING THE REVIEW ......................................................................................................................................... 6

2.1.1 Identification of the need for a Systematic Review ................................................................................................. 6
2.1.2 Development of a Review Protocol .......................................................................................................................... 6
2.1.3 Validation of a Review Protocol .............................................................................................................................. 11

2.2 CONDUCTING THE REVIEW .................................................................................................................................. 11

2.2.1 Identification of Research ...................................................................................................................................... 11
2.2.2 Selection of Primary Studies ................................................................................................................................ 12
2.2.3 Study Quality Assessment ...................................................................................................................................... 16
2.2.4 Data Extraction & Monitoring ............................................................................................................................... 16
2.2.5 Data Synthesis ...................................................................................................................................................... 16

2.3 REPORTING THE REVIEW ........................................................................................................................................ 17

3 TESTING CHALLENGES OF AO PROGRAM ............................................................................................................. 19

3.1 DEPENDENT ASPECT IDENTITY ............................................................................................................................ 19
3.2 COMPLEX EMERGENT BEHAVIOR ......................................................................................................................... 19
3.3 COMPLEX AND CHANGING CONTROL & DATA DEPENDENCIES ................................................................. 20
3.4 COGNITIVE NON-DETERMINISM ........................................................................................................................... 20
3.5 DIFFICULTY IN DECIDING TEST ADEQUACY ..................................................................................................... 20
3.6 UNNECESSARY JOINPOINTS DUE TO COMPLETE WEAVING .................................................................................. 21
3.7 COMPLEXITY IN DEFINING TEST AND COVERAGE CRITERIA FOR AO PROGRAMS ....................................... 21
3.8 INTERFERENCE OF ASPECTS AND RESULTING CONFLICTS ................................................................................. 22
3.9 DIFFICULT FOREIGN ASPECTS IMPORT ................................................................................................................ 23

4 SOFTWARE STRUCTURAL TESTING TECHNIQUES ..................................................................................................... 25

4.1 INTRODUCTION ...................................................................................................................................................... 25
4.2 UNIT TESTING TECHNIQUES FOR OOP .................................................................................................................. 25
4.3 OOP BASED TEST TECHNIQUES AND UNIT TESTING OF AO PROGRAMS ...................................................... 26
4.4 UNIT TEST TECHNIQUES FOR AO PROGRAMS ....................................................................................................... 26

4.4.1 Data Flow Based Unit Testing of AO Programs ..................................................................................................... 27
4.4.2 Regression Test Selection Technique for AO Program ............................................................................................ 28
4.4.3 State Based Testing of AO Programs .................................................................................................................... 31
**TABLE OF FIGURES**

Figure 1: Shows yearly distribution of research articles published on AO software testing during the period 1999 – 2008. .......................................................... 15
Figure 2: Different types of interferences [21].......................................................... 22
Figure 3: Working of RETESA Framework [29].......................................................... 29
Figure 4: Model Architecture for fault based testing [19]............................................ 34
Figure 5: Framework for mutation testing [5]............................................................. 35
Figure 6: Overall process of evaluating test design techniques [54]............................... 44
Figure 7: Relationship among Data-Flow strategies [59].............................................. 59
TABLE OF TABLES

Table 1: List of select journals and conferences.................................................................13
Table 2: Articles included for systematic review ...............................................................14
Table 3: Analysis of Regression testing technique from different Researchers ...............30
Table 4: Analysis of State Based Testing Techniques from different Researchers ..........33
Table 5: Analysis of Fault based and Mutation testing of AO programs .........................37
Table 6: Tools/Frameworks for AO testing.........................................................................42
Table 7: Relation of test technique and fault type.........................................................47
# Introduction

Separation of concerns (SoC) is the most important aspect in software engineering. It refers to the ability to identify, encapsulate, and manipulate only those parts of software that are relevant to a particular concept, goal, or purpose [47]. Concerns are classified as primary concerns and cross-cutting concerns. Primary concerns are related to core functionality and lie within problem domain and can be easily implemented in traditional programming paradigm like Object Oriented Programming (OOP). OOP implements primary/core concerns in classes. On the other hand cross-cutting concerns for example security, synchronization, error and exception handling, scheduling and optimization, the code implementing these cross-cutting concerns is scattered or woven with the other code in different classes which contradicts to fundamental principle of OOP i.e. SoC. OOP languages have serious limitations in modularizing adequately crosscutting concerns in a program. The code corresponding to these concerns is often scattered and duplicated in several basic classes [48].

Aspect Oriented Programming (AOP) is new paradigm having foundations on OOP. AOP is based on the idea that concerns crosscutting several modules of an application can be developed as single unit of modularity and weaved into application, through a process of composition, using join points (a construct of AOP) [21]. AOP offers a way of dealing with system behavior, which does not fit cleanly into the programming models currently being used in the industry [45]. The system behavior that cannot be encapsulated in classes because of its impact across the whole system is called crosscutting behavior [46]. AOP encapsulates this kind of behavior in aspects. For example, code regarding a particular security policy is commonly scattered across multiple class and methods that are defined to implement security policy. However, with AOP code that implements security policy is encapsulated typically into one aspect. Constructs of AOP improve modularity, cohesion of components and offer a better localization of modifications as stated in [42].

There are two scenarios in which aspects can be defined. In one scenario the aspect are the result of refactoring and aggregating the common code from may primary abstraction in one place i.e. aspect [56]. The benefit of using aspects in this way is smaller code units, and to some degree it allows crosscutting concerns to be treated as distinct entity [22]. The result of weaving the aspect back into related primary abstractions should show the behavior identical to the original non-factored implementation.

In second scenario which is inverse of first, instead of refactoring code from primary abstractions and aggregating it in the form of aspect implementation, aspect is defined independently with respect to crosscutting concern that is not present in primary abstraction(e.g. synchronization or security policy)[22].

Aspect oriented programming changes the development process[42]. Class and methods of primary concerns are developed and tested as before using object oriented programming paradigm however code regarding cross-cutting concerns is not embedded into bodies of methods instead it is contained in separately defined aspects [45][42]. Later aspects are woven with classes and it results in woven program composite of behavior of both core concerns and cross-cutting concerns [42].

Testing is an essential part of software development process that ensures software correctness [48]. There exist several testing techniques of different types such as unit testing, integration testing, system testing and others. These and other techniques have been developed, researched and applied on different programming paradigms. Even after so much extensive research for well known programming paradigms, testing is still a challenging
activity in software development process [48]. AOP is relatively new programming paradigm and aspect oriented (AO) programs provides different characteristics which differ from OO programs. There are new challenges regarding testing due to some characteristics of aspects, like dependency on the context of classes, tight coupling to class etc. [48][42]. The relation of aspect and its related classes is fundamentally different from the one existing between the classes in object oriented programs [42]. Alexander et al. [42] argues that challenges that AOP presents, cannot be addressed using traditional unit or integration test techniques, these techniques are applicable to class that implement core concerns but not applicable to aspects because aspect depend on woven context. Alexander et al. [42] mentions the four potential sources of faults and suggests that to find fault in AO program the code of the woven aspect must also be examined. AOP language features usage can not only cause new types of faults to appear in the program, but in addition aspect weaving can also affect the types of faults that are commonly found in object oriented programs and procedural programs [42].

In this thesis we want to explore the unique characteristics of testing techniques regarding AO programs and besides it will also be discovered that how testing these programs is different in terms of complexities. Moreover the evidence on effectiveness of testing techniques in research literature will be analyzed.

1.1 Aims and objectives

Our study aim is to identify the challenges and issues concerned with testing AO programs and gather evidence on structural test techniques and their effectiveness for AO programs. This aim will be achieved by performing a systematic review of the current state of the art research in the area of AO software testing. The following are the objectives to achieve the aim of this thesis.

1. To identify challenges/issues involved in testing AO programs.
2. To identify and analyze structural testing techniques specific to AO programs and object-oriented software structural test techniques that can be used to test AO programs.
3. To gather the evidences on, structural test techniques effectiveness for AO programs and methods that are used to evaluate effectiveness of structural test techniques.

1.2 Research questions

1. What are the testing challenges/issues/problems that are proposed for AOP based software in research literature?
2. What are the specific structural testing techniques that are proposed for AO programs, and whether the existing structural test techniques used for OOP based programs, can also be used to test AO programs?
3. What are the methods (if any) proposed in literature for evaluating the effectiveness of structural test technique for AO programs?
4. What are the evidences proposed in literature regarding the effectiveness of structural test techniques for AO programs?
1.3 Expected outcomes

The expected outcome of the thesis would be a comprehensive report which contains a systematic review of state of the art research on AO software testing. The systematic review focuses on the testing issues, techniques and their effectiveness for AO programs. The following would outcome for each research question.

1. List of testing challenges for AO programs.
2. Brief report on structural test techniques proposed in research literature for AO programs.
3. Brief report on methods to evaluate structural test technique effectiveness for AO programs, proposed in research literature.
4. List of structural test techniques most effective for testing AO programs.

1.4 Research Methodology

We used a mix of qualitative and quantitative research methodologies. Systematic review is performed according to the guidelines by Kitchenham [44]. Journals and proceedings of different conferences are explored by using proper search strings with the purpose of obtaining the most relevant study material to answer the given research questions. A clearly defined search strategy, selection and quality criterion has been defined for the primary studies of systematic review. Results from different studies are collected using data extraction forms and summarized in tables to give quantitative data synthesis of the results from different studies (e.g. See Table 7). Conclusions based on data of these summary tables are mentioned in the systematic review report.

Due to heterogeneous nature of the data of the primary studies authors have performed qualitative synthesis on extracted data from primary studies. Authors have used qualitative synthesis to infer the relevant information from the topic as a whole i.e. from a set of studies that focus on the issue. Line of argument qualitative synthesis has been used. It involves two steps, in the first step authors have analyzed individual studies and in second step a set of individual studies is analyzed as whole.

1.5 Related Work

The preliminary study of literature indicates that no systematic review has been conducted in the area of effectiveness evaluation of software test techniques for AO program. Two research works are present in the literature in which research from different authors have been documented who have proposed different test techniques for systematic testing of AO programs. A.A. Naqvi et al. [1] have evaluated three structural unit testing techniques for AO program. In their work they have evaluated the Data flow based unit testing presented by Zhao [3], State based testing approach for AO programs presented by Dianxiang Xu et al. [43], and Aspect Flow-Graph based test technique by Weifang Xu et al. [41], on the base of fault model presented by Alexander [42].

Maimandi and Ghani [6] have conducted a survey for testing approaches developed for AO programs in 2007. In their survey, first they have presented the fault model presented by Alexander [42], and then they have expressed five different testing approaches by different researchers for AO programs. In this review they have accounted Data flow based unit testing of AO programs by Zhao [3], State based approach for testing AO programs by Dianxiang Xu et al. [43], Aspect flow graph based testing for AO programs by Weifang Xu et al. [41], unit testing aspetual behavior by Lopes and Ngo in 2005, and A model based approach for test generation of AO programs by Weifang Xu and Dianxiang Xu.
Zhao [3] was the first researcher who proposed Data flow based unit testing technique for AO programs. In his research work, the main emphasis is on the use of def-use pairs. The testing criteria that he proposed covered three kinds of interactions between base code and AO code. He has declared a method of a class or an advice of an aspect as a module, so he has recommended 1) Intra module level testing 2) Inter module level testing and 3) intra class or intra aspect level structural testing criteria in his approach [3].

Guoqing Xu and Atanas Rountev [10] have proposed AspectJ Inter-module Graph (AJIG) for safe and precise test case selection for regression testing of AspectJ programs in 2007. In their research work they have used control flow graphs for notifying the behavioral consequences of interactions of aspects with base code [10]. Guoqing Xu [29] has proposed a framework named RETESA (REgression TEst Selection for AO programs). According to [29] RETESA helps in the safe selection of test cases for AO programs from the old test suite. This framework has five components which sequentially accepts the old test suite and at the end efficiently outputs the effective test cases for the testers. This framework comprises of dynamic decision making for the elimination of inappropriate test cases [29].

Tao Xie and Jianjun Zhao [20] has proposed a framework ‘Aspectra’ for the automated generation of test inputs of AspectJ programs in 2006. According to [20] Aspectra can select the effective test cases with the use of tool support from the old test suite and reduce the monotonous manual efforts of software testers of covering the behavioral effects of aspect interactions.

Tao Xie et al. [16] have proposed a framework ‘Raspect’ in 2006. According to [16] this framework can identify the redundant test cases for AspectJ programs, and it only selects the non-redundant test cases from the old test suite. This reduction in test cases provides ease to the developers and testers in order to save their time for the testing of AspectJ programs [16].

Guoqing Xu et al. [2] have proposed a framework for the automated unit test generation for AO programs in 2004. In [2] authors claimed to have developed a testing framework ‘JOUT’ for the reduction of insignificant test cases [2].

1.6 Structure of Thesis

This section provides the structure of the thesis and provides an abstract of each chapter.

Chapter 2 (Systematic Review) gives a detailed description of the basic elements of systematic review. It presents review protocol which give details of a comprehensive plan for conducting systematic review. The process adopted for systematic review on testing AO programs is presented in this chapter.

Chapter 3 (Testing Challenges of AO program) this chapter details the systematic review results for the research question proposed in the review.

Chapter 4 (Software Structural Testing Techniques) This chapter highlights the test techniques which are proposed in research literature for AO programs in context of research questions.

Chapter 5 (Effectiveness of Structural Test Techniques for AO programs) this chapter presents the results of studies on effectiveness evaluation of test techniques for AO programs, according to the research question in review.

Chapter 6 (Epilogue) this chapter presents the conclusion of this study; it also contains the future work and identifies the validity threats for this study.
2 SYSTEMATIC REVIEW

A systematic literature review is way of identifying, evaluating and interpreting all available research relevant to a particular research question, or topic area, or phenomenon of interest [44]. Individual studies that contribute to systematic review are called primary studies where as systematic review is called secondary study [44].

Systematic review consists of three phases namely planning the review, conducting the review and reporting the review. The planning phase of systematic review involves identification of the need for systematic review and development of a review protocol. Review protocol is a search guide line for whole systematic review process. Conducting the review phase involves identification of research, selection of primary studies, study quality assessment, Data extraction & monitoring and Data synthesis [44]. Reporting the review is single stage phase of systematic review process.

Some of features mentioned by Kitchenham for systematic review that distinguish it from literature review are the following [44]:

- Systematic reviews start with definition of review protocol. Review protocol species research questions, search strings, resources to be searched and methods that will be used to perform the review.
- Systematic reviews are based on a defined search strategy aiming to search all the relevant literature as much as possible.
- Systematic reviews involve explicitly documenting the search strategy so that reader can assess its rigor and completeness.
- Systematic reviews have explicit inclusion and exclusion criteria to asses each potential primary study.
- Systematic reviews specify quality criteria for evaluating the quality of each primary study and the information to be obtained from each primary study.
- Systematic review is a prerequisite for quantitative meta-analysis.

2.1 Planning the Review

2.1.1 Identification of the need for a Systematic Review

Aim of this systematic review is to collect and summarize the existing research on testing AO programs during the period of 1999 to 2008. In this systematic review authors will find the complexities involved in testing AO programs based on the research literature and will provide a brief summary of testing techniques and their effectiveness for AO programs. Authors will also provide the list of the areas where research is lacking.

2.1.2 Development of a Review Protocol

Review protocol is a detailed plan for conducting a systematic review and provides a method for selecting primary studies thereby reducing biasness [44].
2.1.2.1 Background

The major use of systematic review has been in the field of medicine. It is also being used now in Software engineering research to support Evidence based software engineering [44]. Every researcher starts his/her research work by doing some sort of thorough and fair literature review. A literature review that does not have thoroughness and fairness is of little importance and value. Systematic way of doing review makes it more likely to be fair and reduce the chance of biasness. So a systematic review summarizes the existing work in a fair way, because systematic review is undertaken using a predefined search strategy. Search strategy should be documented in such a way that its completeness can be assessed. The researcher undertaking systematic review should try to report research that supports his/her preferred research hypothesis as well as the research which does not [44].

AOP is a new paradigm having foundations based on Object Oriented Programming (OOP). The software developed using AOP paradigm offers some benefits in terms of modularity, cohesion and localization of code which otherwise may be duplicated across multiple modules. It also offers some challenges when it comes to testing. In this review authors want to explore and summarize all the research in AO software testing from 1999 to 2008 in accordance with the research question mentioned in the review protocol. This systematic review will present state of the art research in the area of AO software testing and will point out the research gaps in this area.

2.1.2.2 Research Questions

Following research questions will be answered in this systematic review.

RQ 1. What are the testing challenges/issues/problems that are proposed for AOP based software in research literature?

RQ 2. What are the specific structural testing techniques that are proposed for AO programs, and whether the existing structural test techniques used for OOP based programs, can also be used to test AO programs?

RQ 3. What are the methods (if any) proposed in literature for evaluating the effectiveness of structural test technique for AO programs?

RQ 4. What are the evidences proposed in literature regarding the effectiveness of structural test techniques for AO programs?
2.1.2.3 Search Strategy

The search process for the study is based on online searching. The search strings and resources to be searched are listed below.

a) Search Strings
Following search terms will be used to extract the primary studies.
1. Aspect Oriented Programming
2. Aspect Oriented Program AND Testing
3. AOP Testing
4. AOP AND Testing Technique
5. Aspect Oriented Programming AND Complexities
6. Aspect Oriented Programming AND Challenges
7. Aspect Oriented Programming AND problems
8. Aspect Oriented Programming AND Issues
9. Aspect Oriented Programming AND faults
10. AOP AND complexities
11. AOP AND challenges
12. AOP AND problems
13. AOP AND issues
14. AOP AND faults
15. AOP Testing AND challenges
16. AOP testing AND complexities
17. AOP Testing AND problems
18. AOP Testing AND issues
19. Aspect Oriented Programming AND OOP test techniques
20. AOP AND OOP test techniques
21. AOP Testing AND survey
22. Aspect Oriented Programming AND test technique AND evaluation
23. Aspect Oriented Programming AND test technique AND evaluation AND Method
24. AOP Test Technique AND evaluation AND effectiveness

b) Resources to be searched
Following online resources will be used during the systematic review.
- IEEE Xplorer
- ACM Digital Library
- Springer Link
- Electronic Library Information Navigator (ELIN@Blekinge)
- Compendex

The research journal and conferences will also be explored from 1999 to 2008 to make sure that we have all the relevant research reviewed and don’t miss the important research concerning our area of study.

2.1.2.4 Study Selection Criteria and Procedures
The research study selection will be based on the following inclusion and exclusion criteria:
a) Study Inclusion Criteria

The articles/research papers from Jan 01, 1999 to 30th May 2008 will be included in the review study and suitability of the articles will be assessed on the bases following criteria:

1) The research paper/article should be peer-reviewed by at least one reviewer.
2) Articles of these types will be considered only: case study, experiment, survey, experience report, comparative evaluation or action research.
3) The article will be considered if it proposes success/issues/failures or any type of experience on testing AO code/program/software.
4) The article’s full text should be available.
5) An article proposes new claims regarding AO testing and supports these claims with some sort of evidence or strengthens the already reported claim in research literature with new evidence.
6) The article is included if it identifies the problems / challenges in AO software/code/program testing.
7) The article is included if it provides some sort of technique / method / framework of AO software/code/program testing.
8) An article in included if it evaluates or compares two or more AO software testing techniques.
9) An article is included if it surveys two or more AO software testing techniques.

b) Study Exclusion Criteria

The articles that do not meet the specification mentioned in study inclusion criteria will not be included and in addition the articles published on websites of the companies and student thesis reports would not be included in the systematic review.

2.1.2.5 Study Selection procedure

The study selection procedure will involve studying the title, abstract and conclusion sections of the research paper and if it satisfies the inclusion criteria mentioned in review protocol then these articles will be considered and whole article will be studied.

2.1.2.6 Study Quality Assessment checklist and Procedures

The selected research papers will be evaluated based on research paper structure criteria. The potential primary study research paper will be evaluated on structure i.e. Introduction, Method, Results, Analysis, Discussion / Conclusion. Answers for the following questions in the checklist regarding structure of each section will be searched in each potential study.

**Introduction:** Does the research paper have introduction section that provides an overview of AOP?

**Method:** Does research paper clearly identify the research methodology used? Is the research methodology suitable for problem discussed in the paper?

**Results:** Does the research paper completely define the study results? Are results helpful to solve the AO software testing problems? What sorts of validity risk are defined in the research paper for its results?

**Analysis:** How the data in the research paper has been analyzed and evaluated? What sort of techniques have been used for analysis e.g. SWOT analyses, risk analysis etc.? If the research paper proposes some framework then is it validated in some industrial environment?

**Conclusion:** Does the research paper report negative findings properly? Does the paper also reports the limits or restriction imposed on the research conclusion claims?
2.1.2.7 Data Extraction Strategy

Data extraction forms will be used to extract the following information from each selected primary study paper. If the potential study does not explicitly show the required following information e.g. to decide for study environment, inference is made on the basis of the context and which is then validated by our supervisor for its correctness. Data extracted from each potential primary study involves some general information and some specific information as described following:

1. General Information about Research Article
The following general information will be documented for each research article.
1.1 Article Title
1.2 Author(s) Name
1.3 Name of Journal /Conference/Conference Proceedings article was published/presented in.
1.4 Search String(s) Used to Retrieve Research Article
1.5 Retrieval Database of Research Article
1.6 Publication Date

2. Specific Information about Research Article

2.1 Study Environment of research paper
2.1.1 Industrial study
2.1.2 Academia study

2.2 Research Methodology used in primary study
2.2.1 Experiment
2.2.2 Case Study
2.2.3 Survey

2.3 Participant involved in primary study
2.3.1 Professionals
2.3.2 Students
2.3.3 Number of participants

2.4 Relevant area of research study
2.4.1 Challenges in AO Software Testing
2.4.2 Problems with AOP
2.4.3 Solutions of AO Software Testing Problems
2.4.4 Model / Framework for Evaluating AO Software Testing Techniques
2.4.5 AO Software Testing Techniques
2.4.6 Number of AO Software Testing Techniques Used in Model / Framework
2.4.7 Name(s) of Test Techniques for AO software, Used in Model / Framework
2.4.8 Evidence Regarding Validation of Proposed Model / Framework in the article.

2.5 Method for measuring effectiveness of test Techniques
2.5.1 Evaluation of AO Software Test Techniques
2.5.2 Comparison of Test Techniques for AO Software

2.6 Validity threats involved
2.6.1 Conclusion Validity
2.6.2 Construct validity
2.6.3 Internal validity
2.6.4 External validity

2.1.2.8 Synthesis of the extracted data

Data synthesis involves collecting and summarizing the results of the selected primary studies. The studies for systematic review that are different from each other with respect to methodology and outcomes are called heterogeneous studies. Due to heterogeneous nature of the data of the primary studies authors propose qualitative synthesis to be performed on extracted data. In this qualitative synthesis authors will read and analyze the research articles. The results from primary studies will be documented in accordance to the research questions mentioned in review protocol. Data extraction forms will used for each primary study to obtain information from the primary study.

2.1.3 Validation of a Review Protocol

The review protocol is the most important and critical element of systematic review. The agreed validation process is necessary to make it transparent and fair enough. Kitchenham et al. [44] has proposed pilot searches for identifying the potential primary studies using search strings and resources defined in review protocol. The review protocol for this thesis has been validated and reviewed by the Thesis Supervisor who is an experienced researcher in Blekinge Institute of Technology Ronneby. Search strings have been checked and validated by the authors and also we have taken some help from librarian regarding search strings and resources.

2.2 Conducting the Review

The steps involved in conducting a systematic review are explained as follows:

2.2.1 Identification of Research

Systematic review aim is to find as many studies as possible, pertaining to the research questions using unbiased search strategy [44]. The search strategy is explicitly defined in review protocol and presented in replicable way. Search strings are defined according to research questions. Research question are divided into individual elements like population, intervention, outcomes and study designs, to extract and define the search strings [44]. The synonyms for the words of search strings are also identified. Search strings can be combined using AND to form a bigger and meaningful search string. Boolean OR can be used to include synonym for the search string words.

In software engineering experiment based studies, population may refer to any of the following [44]:

- A specific Software engineering role e.g. testers, software quality assurance managers.
- A type of software test engineer, e.g. a novice or experienced test engineer.
- An application area e.g. IT systems, command and control systems.

A question may be concerned with very specific population groups e.g. a novice software test engineer working on Real Time systems testing.

Intervention is specific software technology(s) that address the specific issue and are used to perform a specific task [44]. Software technology could be any software tool, method or procedure, e.g. Data Flow testing is an intervention for software testing.

Outcomes are the concerned with factors of interest to practitioners or researchers [44] e.g. Early and quick detection of faults in software testing is point of interest to practitioner i.e. Tester.

Authors have conducted searches using search strings defined in review protocol to search for studies relevant to research questions on different electronic search resources,
because no single resource gives all the relevant studies so authors have selected different 
electronic resources.

Systematic reviews may involve publication biasness that is positive results are more 
likely to be published than negative results [44]. So authors have to take care of it while 
reporting on the studies. Barbra [44] has mentioned three strategies to tackle the problem of 
publish biasness.

First strategy is scanning gray literature i.e. the research literature published by industry, 
academia, business or government that is not commercially biased. Authors have clearly 
defined inclusion criteria that study should be peer reviewed as most of research published 
by academia, industry and government does not fulfill this requirement and it is likely to be 
commercially biased and lack bibliographic control that is why this strategy is not used for 
the proposed systematic review study.

Second strategy mentioned by Barbra [44] is scanning the conferences preceding. 
Authors have searched not only the online resources, but the conference proceeding and 
journals were manually searched to reduce the chances of publication biasness. The 
conferences and journals (See table 1) are mentioned in review protocol that has been used to 
find relevant studies.

Third strategy mentioned by Barbra to reduce biasness is contacting experts working in 
the concerned area for knowing about any unpublished results. As unpublished results mean 
not satisfying the main peer review criteria, that is this strategy is not used for this study.

Endnote reference management software has been used to manage the large number of 
references for primary studies. Each article’s details were recorded in Endnote and then 
downloaded. Endnote made the double insertion of a particular article reference impossible, 
which saves the energy of checking for any duplication of reference.

2.2.2 Selection of Primary Studies

Primary studies selection is a two step process. In first step authors have studied the 
Title, abstract and conclusion to decide for the relevant studies. In the second step selected 
research papers from first step are reviewed on the basis of inclusion/exclusion criteria 
defined in review protocol and final list of primary studies is achieved. Table 1 lists the 
selected journals and conferences for the primary studies. During this process conflicts on 
the decision of inclusion or exclusion of the potential study are resolved by discussion and 
consulting the supervisor and/or by reviewing and editing the inclusion criteria.
<table>
<thead>
<tr>
<th>Journals</th>
<th>Conference/Symposium</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE Transactions on Software Engineering (TSE)</td>
<td>International Conference on Emerging Technologies</td>
</tr>
<tr>
<td>ACM, Aspect Oriented Software Development (AOSD)</td>
<td>IEEE International Conference on Program Comprehension (ICPC'07)</td>
</tr>
<tr>
<td>ACM, Transactions on Software Engineering and Methodology (TOSEM)</td>
<td>IEEE International Conference on Automated Software Engineering (ASE)</td>
</tr>
<tr>
<td>LNCS Transactions on AOSD, Springer</td>
<td>IEEE International Conference on Software Maintenance (ICSM)</td>
</tr>
<tr>
<td>Technical Reports</td>
<td>Conference on Computational Science and Applications</td>
</tr>
<tr>
<td></td>
<td>International Conference on Software Engineering (ICSE)</td>
</tr>
<tr>
<td></td>
<td>ACIS International Conference on Software Engineering Research, Management and Applications (SERA)</td>
</tr>
<tr>
<td></td>
<td>Asia-Pacific Software Engineering Conference (APSEC)</td>
</tr>
<tr>
<td></td>
<td>International Conference on Software Reuse (ICSR)</td>
</tr>
<tr>
<td></td>
<td>Euro-Par Conference</td>
</tr>
<tr>
<td></td>
<td>International Conference on Quality Software (QSIC)</td>
</tr>
<tr>
<td></td>
<td>Annual International Computer Software and Applications Conference (COMPSAC’03)</td>
</tr>
<tr>
<td></td>
<td>ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)</td>
</tr>
<tr>
<td></td>
<td>International Symposium on Software Composition (SC)</td>
</tr>
<tr>
<td></td>
<td>International Symposium on Software Reliability Engineering, ISSRE</td>
</tr>
<tr>
<td></td>
<td>Software Engineering Properties of Languages and Aspect Technologies (SPLAT) Workshop</td>
</tr>
<tr>
<td></td>
<td>Workshop on Software Quality (WoSQ)</td>
</tr>
<tr>
<td></td>
<td>Workshop on Testing Aspect Oriented Programs (WTAOP)</td>
</tr>
<tr>
<td></td>
<td>Workshop on Software Quality (SOQUA)</td>
</tr>
<tr>
<td></td>
<td>International Workshop on Aspect Oriented Soft. Development</td>
</tr>
<tr>
<td></td>
<td>Workshop on Assessment of Contemporary Modularization Techniques (ACoM’)</td>
</tr>
<tr>
<td></td>
<td>Workshop on Rapid Integration of Software Engineering Techniques (RISE)</td>
</tr>
<tr>
<td></td>
<td>Workshop on Mutation Analysis (Mutation 2006 - ISSRE Workshops)</td>
</tr>
<tr>
<td></td>
<td>Quality of Information and Communications Technology (QUATIC)</td>
</tr>
<tr>
<td></td>
<td>Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)'06</td>
</tr>
<tr>
<td></td>
<td>Informatica (Ljubljana),</td>
</tr>
<tr>
<td></td>
<td>Software Engineering Research Laboratory,</td>
</tr>
<tr>
<td></td>
<td>Japan Society for Software Science and Technology</td>
</tr>
<tr>
<td></td>
<td>Department of Computer Science, North Dakota State University</td>
</tr>
</tbody>
</table>
In this systematic review authors scanned 6996 research papers/articles and 54 were selected. Out of these 54, 43 satisfied inclusion/exclusion criteria and were finalized for systematic review. The listing of these articles is given in table 2.

<table>
<thead>
<tr>
<th>Ref#</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>[2]</td>
<td>JAOUT: Automated Generation Of Aspect-Oriented Unit Test</td>
</tr>
<tr>
<td>[7]</td>
<td>On Identifying Bug Patterns In Aspect-Oriented Programs</td>
</tr>
<tr>
<td>[8]</td>
<td>Diagnosis Of Harmful Aspects Using Regression Verification</td>
</tr>
<tr>
<td>[10]</td>
<td>Regression Test Selection For AspectJ Software</td>
</tr>
<tr>
<td>[15]</td>
<td>Using Program Slicing To Analyze Aspect Oriented Composition</td>
</tr>
<tr>
<td>[16]</td>
<td>Detecting Redundant Unit Tests For AspectJ Programs</td>
</tr>
<tr>
<td>[17]</td>
<td>Interference Analysis For AspectJ.</td>
</tr>
<tr>
<td>[18]</td>
<td>State-Based Testing Of Integration Aspects</td>
</tr>
<tr>
<td>[20]</td>
<td>A Framework And Tool Supports For Generating Test Inputs Of AspectJ Programs</td>
</tr>
<tr>
<td>[21]</td>
<td>Towards Detecting And Solving Aspect Conflicts And Interferences Using Unit Tests</td>
</tr>
<tr>
<td>[22]</td>
<td>Challenges Of Aspect-Oriented Technology</td>
</tr>
<tr>
<td>[23]</td>
<td>Towards Regression Test Selection For AspectJ Programs</td>
</tr>
<tr>
<td>[24]</td>
<td>Testing Aspect-Oriented Programs As Object-Oriented Programs</td>
</tr>
<tr>
<td>[26]</td>
<td>State-Based Incremental Testing Of Aspect-Oriented Programs</td>
</tr>
<tr>
<td>[27]</td>
<td>Towards A Fault Model For AspectJ Programs: Step 1 – Pointcut Faults</td>
</tr>
<tr>
<td>[28]</td>
<td>Distributing Classes With Woven Concerns – An Exploration Of Potential Fault Scenarios</td>
</tr>
<tr>
<td>[29]</td>
<td>A Regression Tests Selection Technique For Aspect-Oriented Programs</td>
</tr>
</tbody>
</table>
Figure 1: Shows yearly distribution of research articles published on AO software testing during the period 1999 – 2008.

Figure 1 clearly shows that the number of publications on AO software testing started to rise gradually after year 2003 and in the year 2006 it reached to its maximum level. Authors have observed that initially more focus was on AO software design because AO development paradigm is relatively new and more attention was given on development side of software using AO paradigm.
The rejected articles are included in appendix II. There are different reasons for not including these articles in the systematic review. During the search of different journals and databases, through pre-defined search strings special care has been taken in order to avoid the chance of missing any relevant article. But after the thorough study of all the selected articles some articles are found which did not depict the research according to the essence of research questions. For example some articles accentuate testing automation of AO programs which is not the concern of this systematic review therefore those articles instead of having relation with testing AO programs have been rejected on the grounds because they are out of scope, defined by the research questions.

2.2.3 Study Quality Assessment

Study quality assessment is performed on the selected articles/primary study in the review to assess the relevance of the primary study and to gain the confidence regarding the quality of these selected studies. This quality assessment is done by studying the research paper’s introduction, method, results, analysis and discussion sections and finding the answers for the questions defined to assess the quality of the study for each section of the paper. The questions details can be explored in section 2.1.2.6.

2.2.4 Data Extraction & Monitoring

The main objective of this step is to design data extraction forms with the purpose to document the data extracted from the primary studies [44]. Data extraction forms are defined and piloted when the review protocol is defined [44]. This helps to clear the reader what data would be extracted from the primary study and also it minimizes the chance of biasness.

To reduce the chance of biasness and missing out any important information one author extracted the data from primary studies and other author cross checked the extracted data during this phase.

2.2.5 Data Synthesis

Collation and summarization of the results of selected/included primary studies is called data synthesis [44]. The extracted data is synthesized in such a way that it answers the research questions defined in review protocol. The synthesis of data can be in descriptive form and it can be complemented with quantitative summary of data, others forms of synthesis are qualitative synthesis and quantitative synthesis [44].

Descriptive synthesis involves the extracted information about primary studies, presented in consistent way according to research questions defined in review protocol. Tables should be structured to explicitly highlight the differences or similarities between the outcomes of studies [44]. The results of outcomes of studies can be consistent (homogenous) or inconsistent (non-homogenous). Heterogeneity of studies can be presented in tabular form, in terms of potential sources of heterogeneity that are study type, study quality and sample size.

In quantitative synthesis the integrated results from different studies are presented in tabular form. The extracted data presented in tables include sample size of intervention, estimated effect size for each intervention with standard errors for each intervention, difference between the mean values for each intervention, and effect measuring unit used[44].

In primary studies researchers may have used different language terms and concepts with different meanings. Qualitative synthesis involves integrating such studies results and conclusion cautiously [44]. Kitchenham [44] proposes three approaches for qualitative data synthesis:
1. **Reciprocal translation**
   Reciprocal translation is used when studies involve similar things and researchers are attempting to give an additive summary [44].

2. **Refutational synthesis**
   Refutational synthesis is useful when studies are refutations of each other implicitly or explicitly [44]. In refutation qualitative synthesis individual studies and their refutations are translated first and then analysed.

3. **Line of argument synthesis**
   This approach for qualitative synthesis is useful when researchers can infer the information they are concerned about, from the topic as a whole, from set of studies that focus on the issue [44]. Line of argument synthesis involves two steps, in the first step individual studies are analyzed and in second step a set of individual studies is analyzed as whole.

   The systematic reviews nature for our case appears to be qualitative. Therefore authors selected line of argument synthesis approach for this systematic review.

2.3 **Reporting the Review**
   Reporting is single stage phase in the systematic review study. The results of the systematic review are reported in this phase according to the review protocol research questions. During the systematic review relevant data is extracted using data extraction forms and gathered. This gathered data is synthesized using appropriate data synthesis approaches and finally reported.
Systematic Review Results
3 Testing Challenges OF AO Program

AO programs development and testing involves development and testing of classes and methods as before, however aspects are defined to cover the cross-cutting concerns. Code regarding the cross-cutting concerns is not embedded into methods but rather it is contained in separately defined aspects. Later these aspects are weaved into classes of core concerns/core functionality to get the executable code.

There are two scenarios in which aspects can be defined. In first scenario aspects are the result of refactoring and aggregating of cross cutting concerns in the form of common code from many primary abstractions at one place i.e. aspect [56]. The benefit of using aspects in this way is smaller code units, and to some extent it allows crosscutting concerns to be treated as distinct entity [22]. The result of weaving the aspect back into related primary abstractions should show the behavior identical to the original non-factored implementation.

In second scenario which is inverse of first, instead of refactoring code from primary abstractions and aggregating it in the form of aspect implementation, the aspect is defined independently with respect to crosscutting concern that is not present in primary abstraction(e.g. synchronization or security policy)[22].

The following complexities are involved in testing weaved code.

3.1 Dependent Aspect Identity

Roger et al. [22] identifies challenges of AO technology to raise awareness of its use. Aspects do not have independent identity; they depend on the execution context of base classes which can lead to many new faults [42]. Aspects are not complete code units therefore they cannot be tested alone as a separate unit. In order to test aspects one has to take both aspect and its related classes and weave them to get executable code. Faults which came in to being through the aspect weaving with the primary abstraction can influence the types of faults that exist usually in object oriented programs and procedural programs [42]. Hence it also become difficult to identify where exactly faults relates to, in the code. This issue becomes even more severe when there are several aspects involving some common related classes.

3.2 Complex Emergent Behavior

Roger et al. [22] states that relationship that exists between aspect and its primary abstractions is many-to-many. The root cause of fault may lie in class, aspect or it can be side affect of particular weave order of multiple aspects [42]. The relationship of classes and aspects results in an emergent behavior after weaving process which is difficult to understand and can result in potential faults that are extremely difficult to diagnose [22] [42] [9]. Furthermore the methods used for detecting faults for non-AO programs are not enough for AO programs [42] [22] and there is also need to examine the code of woven aspect [22]. Roger et al. suggest the following four potential sources of faults after weaving process [22]:

- The faults reside in a portion of the primary abstraction not affected by the woven aspect. This kind of fault may occur in primary abstraction irrespective weaving of primary abstraction and aspect.

- The faults reside in code of aspect that is isolated from the woven context. Such fault may occur in any composition that included aspect however fault is located in the
aspect code which is independent of the data and control dependencies introduced by weaving process.

- The faults may emerge from interaction between the aspect and the primary abstraction. This kind of fault would occur when new data and control dependencies are introduced by weaving aspect in primary abstraction that was not present before.
- The fault may emerge from the particular combination of aspects woven into the primary abstraction. This kind of faults may occur with a particular permutation of aspects to primary abstractions. Data and control dependences from multiple aspects combined with those in primary abstraction can cause this kind of fault.

3.3 Complex and changing Control & Data dependencies

Depending on how aspects are used, aspects have the potential to change the syntactic structure and semantics of primary abstractions [22].

There are two scenarios in which aspects can be defined. Firstly, cross cutting concerns in a primary abstraction can be aggregated to form an aspect [56]. Secondly aspects can be defined independently as a separate unit, from the beginning of software development project [22]. No matter what scenario the aspects are used in, control and data dependencies of the composition that results from weaving process will be different from that of primary abstraction [22].

Roger et al. [22] [9] states that control and data dependencies of composition (weaved code) are different from primary abstraction. The control and data dependencies of aspect are incomplete in most of cases if these are dependent on primary abstractions. Control and data dependencies in weaved source code are difficult to comprehend for developers of classes or aspects, due to complex nature of weaving process. So relating failures to corresponding faults becomes difficult [42].

3.4 Cognitive non-determinism

Another facet of weaving process complexity mentioned by Roger et al. [22] [9] is that it changes the cognitive model of a primary abstraction as specified by its author, which leads to cognitive non-determinism. Thus author of aspect faces the challenge of ensuring that the behavior of woven artifact is no stronger than that of primary abstraction it is based on [22]. It is hard to ensure for the author of aspect that his aspect will not cause undesirable emergent properties after weaving; this is even more difficult if the aspect is to woven with other aspects with potentially many different primary abstractions [22]. For large systems the likely scenario that complicates matter is when there are collection of aspects written by different authors have to be woven [22]. The solution proposed is that each author must have knowledge of primary abstractions and the aspects that these are woven with [22].

3.5 Difficulty in Deciding Test Adequacy

Primary concerns and aspects are tested prior to weaving however if the aspect induce dependency changes in primary abstraction, the tests for primary concern or for unwoven aspect might not apply to woven artifact[22].Roger in [9] states that it is difficult to test aspect as a separate unit, because it depends on context of primary concerns. In addition weave order of multiple aspects can affect the resultant woven artifact’s behavior and decision of correct weave order can be made only by trial and error [9]. Given this scenario how to decide which weave order to test. Roger argues that assumption cannot be made that
testing only primary concerns before weaving aspect with them, is adequate and neither can we assume that faults are obvious and easily detected after weaving.

3.6 Unnecessary Joinpoints Due to Complete Weaving

The locations in the code that represent join points at runtime are called join point shadows. The shadows whose join points always lead to execution of aspect specific code are called unconditional join point shadows and the joint point shadows who lead to the execution of aspect specific code under certain conditions are called conditional join point shadows [31]. For conditional shadows the weaver adds runtime checks to determine whether aspect specific code needs to be executed or not. In article [31] authors refer these runtime checks as joint point checks. If this joint point check is successful then aspect specific code (i.e. advice code) is executed. Henenberg et al. [31] argue that complete weaving (pure static weaving) has the property that set of joint point shadows for woven aspects remains the same for the aspect’s life and does not change. Furthermore they describe that for complex applications complete weaving can lead to huge number of adapted join points shadows whose joint point checks fails and they produce not only runtime overhead for performance critical applications but also bring testing challenge in terms of effort and resources.

Article [31] has presented a technique to reduce the number of joint point shadows. The precondition for this technique is that systems have a large number of failing joint points checks during the program execution. Henenberg et al. [31] claim that this approach has reduced number of joint point shadows significantly for AO system Aspects. The approach give by [31] imposes a restriction on the application and that is underlying system must be weaved dynamically (weaving at runtime). More over morphing process takes additional time to determine and create joint point shadows, there by developers have to make trade off here between the runtime overhead due to unnecessarily introduced joint point checks caused by unnecessary adapted shadows and the over head of morphing process itself [31].

3.7 Complexity in Defining Test and Coverage Criteria for AO Programs

AO programs have unique characteristics which do not occur in object oriented, or even procedure-oriented programs. Each characteristic can manifest new types of faults that ultimately lead to program failures [42]. In [13] Bernardi et al. state that main difficulty in testing AOP based systems is associated with interactions among aspect and classes, and in particular among advices and methods they effect. In [13] authors refer to these interactions as ‘spaghetti bowl’ which has sources of failures that are difficult to find and remove. Roger et al. [42] has identified six types of faults:

- Incorrect strength in pointcut patterns
- Incorrect aspect precedence
- Failure to establish expected post conditions
- Failure to preserve state invariants
- Incorrect focus of control flow
- Incorrect changes in control dependencies

Roger et al. [42] has given test criteria to test AO programs for above mentioned six types of faults. These test criteria are focused on the behavior of core concerns, the behavior of each aspect and behavior of the weaver [42]. In [13] Bernardi et al. has proposed coverage criteria for AO programs based on interactions among advices and methods. These coverage criteria uses inter-procedural aspect control flow graph, a graph that represents the relationship among aspects’ advices and classes’ methods [13]. The coverage criteria
proposed in [13] discusses the test coverage criteria pertaining to dynamic crosscutting features only.

### 3.8 Interference of Aspects and Resulting Conflicts

In [4] Tessier has classified aspects by different types of interferences that these aspects can cause. The interference problems identified by Tessier include following:

- Wildcards characters use can lead to unintended join points.
- Conflicts of aspects and weaving order of these aspect into application.
- Circular dependency relationship between aspects.
- Conflicts between concerns i.e. one concerns changes the functionality needed by another concern.

![Figure 2: Different types of interferences](21)

Katz [8] classified aspects by the type of changes, aspect introduce in an application. Three types of aspect are identified by Katz:

- Spectative aspects: these aspects only collect information about the system to which they are weaved. This information is collected by adding fields and methods but it does not affect the possible underlying computation.
- Regulatory aspects: these aspects change the flow of control but don’t change the computation done to existing fields.
- Invasive aspects: these aspects change the values of existing fields but don’t invalidate the desirable properties.

In [21] author has used the work of Katz [8] and Tessier [4] to analyze how different types of changes introduced by aspects can create different types of interferences as shown in figure 2. Restivo et al. [21] has given an approach to resolve the problems of inferences and aspect conflicts by unit testing.

Katz [8] proposed use of regression testing as tool to help identify harmful aspects. Kienzle [14] states that aspects can be viewed as entities that require services from a system provide new services to that same system and remove other services. If we can describe the way what services are needed by each aspect, interferences of aspects can be identified and better weaving order can be chosen as a result to avoid those interferences. Balzarotti et al. [15] claim that the aspect interference problem can be solved using program slicing technique.
In [21] program slice is defined as “A slice of a program is the set of statements that affect a given point in an executable program”. According to Balzarotti et al. [15] this technique can detect not only all the possible interferences due to aspects, but it can also detect some ‘false-positives’. This approach uses point cuts defined on dynamic context and increases the number of false-positives.

Storzer et al. [17] developed a technique to detect interferences caused by two different and related properties of AO languages. Storzer et al. claim that it is possible that aspects can introduce new members in other classes and this can result in undesired behavior, it can result in changes in dynamic look up if the introduced method redefines the method of parent class [17]. Storzer et al. [17] call this type of behavior binding interference. Second problem highlighted by Storzer et al. is that aspects can change the inheritance hierarchy of a set of classes which can result in binding interferences and can lead to undesired behavior. To cater this type of interferences Storzer et al. [17] stress that the analysis should be based on look up changes introduced by aspects. In [32] Kessler analyzed the structural interferences due to aspects and presented an approach based on logic engine. In logic engine programmers can specify rules of ordering, visibility, dependencies, etc.

In [33] Lagaisse et al. have provided an extension to Design by Contract approach originally introduced by Meyer [49] as a defensive solution for dependency problems in object oriented programming. Lagaisse et al. [33] approach allows aspects to define what they expect of the system and how they will change the system. This way of designing aspect can allow the detection of interferences by aspects whether they are weaved already or going to be weaved later on. This approach forces programmer to specify all the requirement and modification for each aspect and also specify the permitted interferences thereby breaking obliviousness of the component.

3.9 Difficult Foreign Aspects Import

The introduction of foreign aspects into a software system can cause the behavioral change in the system in unexpected and unwanted ways. Multiple aspects advice are weaved into a concern in a particular order, affects the system behavior especially when aspects have mutual interactions through state variables in the core concern [42]. Roger et al. [42] refer to this problem as incorrect aspect precedence. The foreign aspects can interfere with the local aspects and affect the correctness, comprehension and maintenance [28]. McEachen et al. [28] highlight the problems caused due unanticipated composition of foreign aspects with the base classes which caused maintenance issues later on. McEachen et al. [28] discuss that unbounded pointcuts definition reliance on wild cards can potentially capture join points in the classes that were not intended to apply. Due to this problem foreign aspect can unexpectedly woven into classes during reweave process can cause maintenance issues in the software.

Unbound pointcuts are necessary in many cases as they enable foreign aspects to importing code during aspect reweaving [28]. McEachen et al. [28] states that fault can occur in both cases if unbounded pointcut does not capture all intended join points and if unbounded pointcuts capture exactly the intended join points. Foreign aspects may turn out to be fragile aspects or faulty due to unknown assumption upon which the pointcuts are based [28]. These assumptions are troublesome for foreign aspect authors and for the authors who are importing foreign aspects. These assumptions ultimately lead to same result and that is foreign aspects turn out to be fragile aspects [28]. When the foreign aspects from different authors are imported into an application, authors of foreign aspect do not know of any additional aspect presence in our code. Therefore the precedence of foreign aspects cannot be decided explicitly by the foreign aspect authors [28]. The default aspect weave precedence for aspects is arbitrary, and it varies for different version of AspectJ softwares and it is different even for the same implementation with different input programs[28]. To
solve some of the problems McEachen et al. [28] suggests three basic awareness criteria about foreign aspects and these are:

- Be aware with the presence of foreign aspect while importing code that contains foreign aspects.
- Knowing the pointcut definition for each advice of foreign aspects.
- Knowing the semantic impact of such advice on our code.

Additionally, developers need a way to explicitly control the weaving of foreign aspects [28].
4 SOFTWARE STRUCTURAL TESTING TECHNIQUES

4.1 Introduction

This chapter enlightens the variation in structural and unit testing techniques and strategies defined by different researchers for AO programs. Testing techniques have been discussed in the context of uncovering dissimilar type of faults as well as having capabilities to provide code coverage criteria by test case generation. Several faults are classified in literature according to their type and origin. During the testing of woven AOP software some faults could originate from aspect oriented and some from object oriented programming paradigms. Besides, there are some faults which could happen due to the weaving of both programming paradigms. The weaving process makes it complicated to diagnose the origin of fault in an AO program. In order to approach towards a solution for above described issue, structural and unit testing strategies for AO programs have been analyzed in this chapter.

Software testing is essential but most expensive activity for the whole Software Development Life Cycle [6]. It covers the efforts needed to correct some identified faults as well as the efforts involve in the identification and rectification of hidden faults in order to avoid drastic failure [50]. It is preferred by many researchers that testing activities should carry on in parallel to the software development work [6]. Apparently it is difficult to manage both activities simultaneously, but we cannot ignore the benefits that we can get through this practice e.g. faults refinement at the earlier software development phase. In a typical software development fashion, after completion of a software unit it is preferred to validate the developed unit according to requirement specification. A unit level testing is performed on all the developed units. Successful validation of these units guarantees that the programmers have codified correct requirements. Most important aspect of a software testing technique is to minimize the number of test cases, and to give maximum coverage of source code to be tested. Although the increase in test cases could result in revealing more number of faults but conversely in this case cost of overall project goes high. Therefore even that specific testing technique is more effective but it is not an efficient technique. Consequently the most suitable testing technique is that which is both effective and efficient [51]. A test case is basically a method to validate that either the code written has accomplished its assigned obligation or not. Well versed test cases increase effectiveness of a testing technique by providing complete coverage of a software code for its syntax and semantics area.

4.2 Unit Testing Techniques for OOP

There are many testing techniques which are used for the unit testing of OO programs, these techniques are based on different test selection methodologies [53]. According to Natalia et al. [53] following are some basic white box testing families and their relevant strategies:

- Control flow based unit testing
  - Sentence coverage
  - Decision coverage
  - Condition coverage
  - Path coverage

- Data flow based unit testing (See Appendix I)
  - All definitions
  - All-c-uses/some-p-uses
  - All-p-uses/some-c-use
- All-c-uses
- All-p-uses
- All-uses
- All-du-paths

- Mutation testing
  - Strong mutation
  - Selective mutation
  - Week mutation

4.3 OOP based test techniques and unit testing of AO programs

AOP is a software development paradigm which builds on OOP concepts. AOP uses some special language keywords and concepts which are commonly pronounced as constructs like ‘Aspects’, ‘advice’, ‘introduction’, ‘joinpoints’, ‘pointcuts’ etc [1]. These constructs are developed for effective working of AO programs. These constructs are enormously helpful for programmers to write the code efficiently, but in contrast when these constructs interact with base code (OO code), during the weaving process then various problems can occur [22]. After weaving process one of the major challenge is, that the base code did not retain its expected post behavior and that is due to the influence of special AOP constructs. Many behavioral issues arise after the interaction of base class and aspect code so it is necessary to focus on the semantics of weaved code [22] [26] [29]. We examine the testing related part for AO program in this thesis report. There are so many challenges in the way of effective and efficient testing of AO programs reported in the research literature (also discussed in the third chapter of this report). It is necessary to develop software testing techniques specifically for AO programs in order to deal with those challenges or problems [1]. Those software testing techniques which are used for OOP based software are not feasible to be used for AO programs without any modifications, due to the complexities which arise through the integration of AOP constructs with base code [1].

There are few testing techniques proposed for AO programs in literature. Even the structural unit testing techniques that are currently in use for AO programs are to some extent the modified adaptation of those testing techniques which are designed for OO programs. There is no technique reported so far which is called to be designed especially for AO programs without taking any idea from previous testing techniques [1].

Although according to research literature there are two researchers who have given an idea of testing AO programs through the OO based testing techniques but this idea still did not get matured. Chuan Zhao and Roger T. Alexander [24] have proposed the testing of AO programs using OO based test techniques. According to [24] after weaving of AO and OO code the resultant artifact is in byte code format and the code obtained by decompiling that integrated byte code is same as the base code with some minor modifications. Authors claimed that for implementation of this technique there is no need to take care of the behavioral complexity of AOP constructs, instead the same OOP based testing techniques can be used to test the decompiled weaved code [24]. This is a new idea and still need a lot of further research in order to be able to generalize for larger applications with confidence.

4.4 Unit test techniques for AO programs

There has been less attention given on the testing side of AO program development [3] [23] [34] [40]. Although there are several well known researchers who have put their efforts to find out some dependable solutions of testing AO programs, but still there is no such trustworthy solution exists for this purpose [7]. Most of the researchers have applied several
structural (white box) testing techniques for testing AO programs. Structural test techniques can better deal with the complexities that occur due to integration of AOP constructs and due to the behavioral dependencies of aspect on other aspect and on base code [42]. The rationale behind preferring white box testing techniques is to thoroughly understand the complete manipulation of data in an iterative way by giving maximum code coverage through test cases [20].

Most of the research work related to AOP testing is of little value because researchers have applied testing techniques on the toy code samples [50]. The results obtained by these experiments cannot be generalized and there is a lot of effort still required to develop some standard road map which could make the software testing job easy, reliable and valuable for AOP.

A big question for software testers which is still hard to answer is what will be the criterion for knowing the adequate level of testing [7]. The selection criteria of test cases, has been nurtured usually in the minds of software testers or programmers. Whereas, there should be a proper frame work or model that can be followed for the development of efficient and effective test suites. Hridesh et al. presented an approach to provide solution for the above mentioned problem in AO programs [30]. In this report authors have presented a solution which they called ‘concern coverage’ and according to them through this technique it is now possible to have accurate and machine readable representation of tester’s intention about the selection of test cases [30].

Following, different testing techniques have been discussed which were presented by different researchers for testing AO programs.

4.4.1 Data Flow Based Unit Testing of AO Programs

4.4.1.1 Introduction

This technique is previously in use for testing OO programs and it is first proposed by Zhao [3] for AO program in 2003. This is the first testing technique introduced in literature specifically for AO program. Data flow based unit testing lie down in the category of white box testing technique. By understanding several varying positions of variables in a code and dependency of those variables on each other this technique helps to develop effective test cases. This technique figures out the def-use pairs by using the control flow graphs for efficient selection of test cases [3]. AO program comprises of both the base code and aspects, and after the weaving process of both codes, different kinds of errors can arise in that code. It is very necessary to understand the origin of those errors. In this technique the piece of advice of an aspect or the method of a class is considered as a module, and three kinds of testing approaches has been proposed, i.e. Inter module level testing, Intra module level testing and Intra aspect or Intra class level testing [3].

Zhao explained in detail the behavior of advices and methods after the weaving of aspects and classes in the context of testing those modules. Author has highlighted the post conditions of aspects and classes by giving a comprehensive example and proves that due to the weaving of advices and methods by the complier, it is illogical to test aspects and classes in isolation [3]. It is recommended by the author that a base code should be tested along with those aspects which altered the behavior of that base code by use of special AOP constructs i.e. ‘advice’ and ‘introduction’. Secondly any aspect should be tested in combination of those methods whose behavior is being affected by those aspects [3].

Control flow graphs are main source for generation of test cases in many unit testing techniques. For maximum code coverage and extracting complete date flow information three kinds of control flow graphs are being created by the author [3] i.e. 1) Control Flow
Graphs (CFG) for Individual modules (for Intra module level testing) 2) Inter-procedural Control Flow Graphs (ICFG) for Interactive modules (for Inter module level testing) 3) Framed Control Flow Graphs (FCFG) for Aspects and Classes (for Intra module level and Intra aspect or Intra class level testing) [3].

Tool support for the implementation of Data Flow Based Unit testing is proposed by [3] which has three parts 1) Driver generator for creating test driver 2) The complier, which analyze the AspectJ program and as an output it constructs the CFG, ICFG and FCFG 3) Test case generator which takes input from the complier and produce test cases for each and every module. Following are some limitations which were highlighted by Zhao for Data Flow Based Unit Testing of AO programs:

- Behavior of classes and aspects which are inherited from parents are not discussed in this paper[3]
- Automation of test case generation is not supported in this technique while authors showed their aim to develop a tool in the future to automate the testing activities presented in this paper [3].

4.4.1.2 Analysis of Data Flow Based Testing Technique

The author has presented structural testing criteria of AO programs with the help of different control flow graphs. Zhao is the first researcher who has presented structural testing criteria for AO programs. In his work no fault based model has been discussed but the main emphasis of his work is to understand the testing coverage criteria which are beneficial for the development of effective and efficient test cases.

4.4.2 Regression Test Selection Technique for AO Program

4.4.2.1 Introduction

Whenever any software is modified and some additional functionality is introduced in order to improve and maintain that software, regression testing is performed to diagnose new faults or bugs [10]. Regression test selection technique for AO programs was introduced by [10] [23] [29] and it comprises of dynamic analysis and static analysis of weaved AO programs. Dynamic analysis is for the safe test selection with maximum code coverage and Static analysis is for the correctness and safety of selected test cases. Regression test selection technique can be applied not only to those modules like methods, advices and intertype declarations, which are modified due to the weaving process, but also to the complete AO program which uses those modified modules [23]. The reusability of those test cases which are already developed for the relevant base code portion of restructured AO programs can reduce the overall testing efforts [10][23][29].

4.4.2.2 Regression Test Selection for AspectJ Programs

Jianjun Zhao et al. [23] have established their technique in 2006 based on the work already done by Harrold et al. on safe regression test selection technique for Java programs [52]. Authors have introduced a concept in their work which is based on [52] called ‘dangerous arcs’ which is the variation between the original and the modified arcs in Control Flow Graph (CFG). These original and modified arcs represent the base code and weaved code respectively. Authors have used CFG’s for modeling of different components of AspectJ programs. They give solution for the modeling of Individual modules i.e. advices, methods or other inter-type methods. Initially they have modeled the individual aspects, then Aspect-Class interactions and finally they have modeled the complete programs of AspectJ to represent the static and dynamic behavioral relationships of Aspects and Classes. For modeling these relationships authors have exploited the CFG’s and developed Aspect Control Flow Graphs (ACFG) for modeling individual aspects and System Control Flow...
Graphs (SCFG) for modeling complete AspectJ programs [23]. Modeling of different versions of AspectJ programs helps in the detection of dangerous arcs, which ultimately helps in selection of effective test cases.

Another research work has been conducted in which Guoqing Xu and Atanas Rountev [10] have proposed a solution for regression testing of AO programs; they have introduced AspectJ Inter-module Graph (AJIG) [10]. AJIG is also based on already introduced regression testing technique for java programs by Harrold et al. [52]. AJIG is comprises of ‘CFGs’ and ‘Interaction Graphs’, former for the modeling of control flow of different classes and aspects and latter for the modeling of methods and advices at some specific joinpoints. This is a three-phased regression test selection technique for AO programs, and authors also introduced a framework for the implementation of their regression test selection technique as a part of their ongoing project [10]. According to authors their technique can reduce the number of test cases and so it is a low cost regression testing technique. At the same time precision of test case selection is also the key feature of this technique [10].

4.4.2.3 Regression Tests Selection Technique for AO program

Guoqing Xu [29] introduces a Safe test selection technique for regression testing in 2006. This technique is applicable in two expected scenarios, first, suppose if there is an Object Oriented Software P and then there is the AO version of that software i.e. P'. The second scenario is, suppose there are two AO software, and where first is the original one i.e. A, and the second is the modified version of the actual software i.e. A' [29].

A new framework for testing AO programs called RETESA (REgression TEst Selection for Aspect-Oriented programs) is introduced by [29]. As discussed above that author has considered any of the two kind of programs for experiment, i.e. either an AO program which is an enhancement of an OO Program Or an enhanced AO version of an AO program. In both cases it is obvious that the control flow graphs will be different for the both programs, so this technique first selects the test cases from the old test suite and then note the differences in the control flow graphs of both program versions [29]. After that this technique analyzed those differences dynamically and re-selects the new test cases for the enhanced version of software. A term ‘dangerous arc’ is used in this technique for those additional edges in CFG which are generated due to the weaving process. A graph traversal algorithm is used to identify the additional edges of a CFG [23] [29].

![Figure 3: Working of RETESA Framework [29]](image-url)
In figure 3 the working of RETESA [29] has been showed that P and P’ are executed and then the dangerous edges are identified by analyzing the differences in the graphs of both programs. The framework analyzed the safe test cases and in parallel those test cases were also analyzed which are designed for P and P’. After that all results are sent as the final report and that report contains precise and selective test cases having maximum code coverage [29].

This framework is comprised of five basic components which are 1) A Dynamic coverage recorder which by taking the byte code of weaved program run old test suite and then by mapping dynamic execution path to CFG, records complete experiment to the ‘coverage matrix’. 2) A CFG comparator which make comparisons between old and new CFG and then identify dangerous edges. 3) A Safe Edge Identifier which identifies the safe edges for every test case and then it produce two things a) effective test sets for P ad P’ b) A set of eliminated tests which have some conditional node for the P’ safe edges. 4) A Candidate Test Selector which produces the candidate tests for the final selection 5) A Final Test Selector which produced the final set of test cases and removed all the non relevant tests. This last component is the final report for the testers from this framework ‘RETESA’ [29].

Table 3: Analysis of Regression testing technique from different Researchers

<table>
<thead>
<tr>
<th>Analysis of Researched work for Regression Testing Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Article Details</strong></td>
</tr>
<tr>
<td><strong>Author/s</strong></td>
</tr>
<tr>
<td><strong>Year</strong></td>
</tr>
<tr>
<td><strong>Tool/Model/Framework</strong></td>
</tr>
<tr>
<td><strong>If based on OOP Testing technique; Which?</strong></td>
</tr>
<tr>
<td><strong>Major Emphasis</strong></td>
</tr>
<tr>
<td><strong>Significance</strong></td>
</tr>
<tr>
<td><strong>Limitations</strong></td>
</tr>
<tr>
<td><strong>Future Work</strong></td>
</tr>
</tbody>
</table>

4.4.2.4 Analysis

Due to the uniformity of working of regression test selection technique, all the researchers have followed almost the same way out to implement this software testing technique for AO programs as shown in Table 3. All the researchers have based their work on the research of Harrold et al., who have presented ‘Java Interclass Graph’ (JIG) for the safe regression test selection for Java programs [52]. Regression test selection technique for AO programs focuses on the safe test selection after the integration of the base code and the
AO code. So the emphasis is on code coverage and not on the identification and nature of faults with the use of fault model.

Guoqing Xu et al. [29] have first identified three problem areas in the regression testing of AO programs and then provide solution for those problems in the form of a three phase testing technique; along with they have also introduced a framework ‘RETESA’ to introduce their technique [29].

4.4.3 State Based Testing of AO Programs

4.4.3.1 Introduction

When an aspect is added to the base class then it changes the states of different variables in weaved code. With the usage of state models the transforming states can be identified, so the extension or new generation of states can be used to develop test cases which can better serve the purpose for testing of newly weaved code. In state based testing every attribute of weaved program has been judged in comparison to its initial and final state to understand the change.

4.4.3.2 State-Based Incremental Testing of AO programs

Dianxiang Xu and Weifeng Xu [26] in 2006 highlighted the reusability of those test cases with some modifications which were basically developed before for testing the base classes [26]. Authors have emphasized on modeling of cross cutting concerns from the very beginning of SDLC. Aspect Oriented State Model (AOSM) has been used to understand variant states of objects in an AO program. Due to dominant nature of some special AOP constructs like ‘around advice’, ‘join points’ and ‘introduction’ usually behavior of base class changes and base classes did not retain their post conditions as per expectations. In result of weaving not only some of the states became vanishes but also some new states have been introduced in base class code [26]. According to authors there are three things happen in the code after the weaving process: 1) Removal of some states 2) Introduction of some new states 3) Introduction of some new events [26]. In [26] authors have presented their research in response to following three research questions,

1. “How to specify the expected impact of aspects on object states for test generation purposes?
2. To what extent can base class tests be reused for testing aspects? Base class tests are not necessarily valid for testing aspect-oriented programs as aspects likely change transitions of object states.
3. How to determine that a programming fault actually has to do with aspects rather than base classes?” [26]

According to authors in [26] state based incremental testing technique has similarity with the regression test technique with a major difference that in state based technique the systematic reuse and changes in the existing test case suite can be better handled [26]. Like other testing techniques this technique also emphasized on minimization and reuse of test cases. According to Dianxiang Xu and Weifeng Xu [26] two kinds of test cases can be written for assuring the required working of any software, first is positive test case, which can check that does software do what it actually supposed to do. Second type is called negative test case, which verifies the software did not do what it is not supposed to do [26]. According to author positive test cases related to base code become negative test cases for aspect code and vice versa [26].

State based incremental testing technique for AO programs have successfully achieved two major goals, 1) extension of the state models which in result facilitate abstract test case generation for AO programs. 2) Exploration of how to reuse base class test cases for AO programs [26]. Authors have also mentioned that four type of faults from the fault model
presented by Alexander et al. [42] can be revealed by use of state based incremental testing, which are as following:

- Pointcut expressions picking out extra joinpoints
- Pointcut expressions missing joinpoints
- Incorrect advice types
- Incorrect advice implementation

Authors have claimed that by the help of state based incremental testing technique above mentioned faults can be revealed [26].

4.4.3.3 State-Based Testing of Integration Aspects

Weifeng Xu and Dianxiang Xu have presented state based testing for integrated aspects in 2006 [18]. In their research authors have first taken as an example two classes ‘Foo’ and ‘Bar’ and then they generate an aspect ‘FooBar’. Authors have exploited AOSM and create state models for aspect code and base code after that, test cases have been produced from those created state models [18].Basically a state model contains states, events and transitions under some specified conditions. Suppose there is an initial state ‘S1’ and then an event ‘E’ happened on ‘S1’ and that state changes and became ‘S2’. In this case the new states can produce the effective test cases which are different from the originals.

There are different factors involve in the structure of a state model, i.e. 1) There are some initial states of variables or objects 2) There are several events happened on some objects 3) There are some defined conditions under which those events happened 4) After the occurrence of those events the states of the objects became changed and they moved from the initial state to a new state. According to [18] there is an immense importance of modeling the overall program for a state based testing technique. A general way for the development of AO modeling can be followed by the identification, analysis, management and representation of the cross cutting concerns in the object oriented programs [18].

According to [18] there can be client class, base class and integrated class in a program. There are various classes which can be the client classes for integration with the aspect, if only one class is integrated with aspect then it is called ‘integrated class. Although the behavior of integrated class with the aspect is tested but it is also necessary to test the interactions of those other classes which are present in that program i.e. client classes.
Table 4: Analysis of State Based Testing Techniques from different Researchers

<table>
<thead>
<tr>
<th>Analysis of Research of State Based Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Article Details</strong></td>
</tr>
<tr>
<td>Author/s</td>
</tr>
<tr>
<td>Year</td>
</tr>
<tr>
<td>Tool/Model/Framework</td>
</tr>
<tr>
<td>If based on OOP Testing technique; Which?</td>
</tr>
<tr>
<td>Major Emphasis</td>
</tr>
<tr>
<td>Significance</td>
</tr>
<tr>
<td>Limitations</td>
</tr>
<tr>
<td>Future Work</td>
</tr>
</tbody>
</table>

4.4.3.4 Analysis

State Based incremental testing technique is developed for capturing those states which become change after the weaving process of base and AO code. As shown in Table 4 the main emphasis of this work is on identifying impact of aspects on the base classes after weaving process, reusability of the test cases generated for the base classes and to categorize aspect based faults in a weaved program.

Whereas, the state based testing for integrated aspects mainly emphasize on identifying the behavioral results of multiple classes and aspects involved in the process of integration. Authors have uncovered different fault types by taking in to account the fault model presented by [42].
4.4.4 Fault Based Testing of AspectJ Programs

4.4.4.1 Introduction

Fault based testing for AO programs based on the classification of fault types and then injection of those faults in the AO programs.

4.4.4.2 Fault Based Testing

This testing technique was introduced by Zhao and Alexander [19] in 2007. Authors have described a process for the efficient testing of AO program on the basis of different faults. Before the introduction of this technique Roger T. Alexander [42] has a major contribution in the research of faults classification for AO programs. He is the first researcher who presented the list of different faults specifically for AO programs and then on the basis of those faults he constitutes a fault model for AO program [42].

There are three elements in a complete AspectJ fault model presented by [19], 1) fault model for pointcuts, advice, intertype declaration and aspects 2) fault lists for pointcuts, advice, intertype declaration and aspects 3) java fault model which contains the java related faults. All three parts are necessary for the working of fault based testing technique for AO programs [19]. For effective and efficient use of Fault Based testing authors have explained dependence model and interaction model for weaved base and AO code. As a whole three models are introduced which are basic ingredients of this testing technique i.e. 1) Dependency model 2) Interaction model 3) fault based model. Dependence relationships among the AO programs are identified. It includes, firstly the ‘control dependence’ which is the control related feature of overall AO program structure, and secondly the ‘data-flow dependence’ which is the definition and usage of the different features of AO program. Authors also have analyzed the semantic and syntactic characteristics of the program. In figure 4, a complete structure of the fault model presented by [19] has been shown, which can be used for fault based testing of AO programs.

![Figure 4: Model Architecture for fault based testing [19]](image)

Roger T. Alexander et al. presented a fault model in 2004 [42]; they have highlighted six kinds of faults specific to AO programs which are 1) Incorrect strength in Pointcut patterns 2) Incorrect aspect precedence 3) Failure to establish expected post conditions 4) Failure to preserve state invariants 5) Incorrect focus of control flow 6) Incorrect changes in control dependencies [42].
4.4.4.3 Analysis

Fault based testing technique for AO programs are at its initial stage [19]. As shown in Table 5 this can be an effective and efficient technique if further explored because the basic purpose of testing activities is revelation of present and absent faults in software. For successful working of this technique it is necessary to have knowledge of behavioral and syntactical interactive relationships of software programming paradigms. This will facilitate a researcher to recognize different categories of faults which is ultimately beneficial for the systematic testing of software and getting effective and efficient results.

4.4.4.4 Mutation Testing of Pointcuts in AO Programs

Mutation testing is a kind of fault based testing. Mutants are basically those versions of a code in which some faults have been introduced intentionally and then effective and efficient test cases uncover those faults [5]. Prasanth et al. have introduced the mutation testing technique for pointcuts in AO programs [5]. Lemos et al. [25] have used mutation testing to identify a fault type related to pointcut descriptors (PCD) introduced by Alexander [42]. In [25] authors have first explained that while integration it is common that unintended and intended joinpoints are selected which causes problems in further execution of program and for testing of programs. Authors have used mutation testing to solve the problem of unselecting intended joinpoints [25]. Whereas according to [5] during AOP software development sometime it happens that more than required or less than needed pointcuts are selected by software developers. Due to the use of wildcards usually a large number of mutants of pointcuts are selected for the testing purposes and it is difficult to select the most relevant mutants among a large number. In this situation it is problematic for software testers to efficiently design the test suite for that particular software. According to authors with the use of their technique it is possible to select the closely similar mutants for the mutation testing suite [5].

Figure 5: Framework for mutation testing [5]

In figure 5 it is clear that the AspectJ source code became an input for the pointcut parser, which after receiving the complete aspect code split that code into pointcut expressions and that expression will became an input for the distance measurer. In parallel to this input Java class code also serves as the input source for the candidate generator which generates the joinpoint candidates and ultimately these candidates are converted into the joinpoint objects. Mutation engine filtered out only those mutants which are closely resemble to the original pointcut expression [5].
Relevant mutants are those versions which are closely resemble to the original programs, whereas equivalent mutants are those versions which are semantically equivalent to the original program. Because of having no semantic difference in equivalent mutants it is very hard to detect them from the huge amount of generated mutants. According to Prasanth et al. [5] the framework they have introduced for mutation testing can efficiently perform two basic tasks 1) generation of relevant mutants 2) detection of equivalent mutants. Basically this technique is based on the pre acquired knowledge and understanding of expected faults in a completely weaved AO program. It assumes that faults can be generated in four different positions of an AO code. 1) Faults can be in the base code 2) Faults can be in the Aspects 3) Faults can be produced due to the weaving of base code with an Aspect 4) Faults can be produced due to the weaving of base code with multiple Aspects. In mutation testing adequacy of testing has been verified by the detection of all inserted faults [5].

Fault based testing and mutation testing described by different researchers is highlighted in an analytical way in Table 5.
Table 5: Analysis of Fault based and Mutation testing of AO programs

| Analysis of Research of Fault Based and Mutation Testing Technique |
|---|---|---|
| **Article Details** | **Article [19]** | **Article [5]** |
| Author/s | Zhao and Alexander | Prasanth et al. |
| Year | 2007 | 2006 |
| Tool/Model/Framework | Fault Model | Framework for efficient mutant generation |
| If based on OOP Testing technique; Which? | Previously it is used for OOP e.g. Java Program Fault Model | Mutation testing was used before for OOP |
| Major Emphasis | Syntactic & Semantic knowledge of data and control flow in a program. Understand the complete interactions among base code and aspects. Consider the fault models for aspects and base code. | Generation of relevant mutants and Detection of equivalent mutants for effective and efficient mutation testing. |
| Significance | AO programs can be tested by using previous OOP based techniques | This technique can effectively generate mutants which can be then tested by using the already designed test data. The framework reduces the time for selection of mutants. |
| Limitations | Testing methodology not described | Not Specified |
| Future Work | To complete design of three described models To design relevant test method for each fault To obtain testing coverage criteria and testing strategy from all three models | Need further research | Not Reported |
4.4.5 Testing AO programs as OO programs

4.4.5.1 Introduction

This approach was presented by Chuan Zhao and Roger T. Alexander [24] in 2007, in which they gave an idea of first decompiling the woven code and then testing that code same like an OO code. They emphasize on the testing of AO programs as a woven code instead of first testing base code, then Aspects and finally weaved code [24]. Authors have exercised their approach on a small code example and they found that if AO woven bytecode is decompiled then the resultant product is an OO code, and that code is similar to the original code and need minor modifications. Now this code can be tested with the help of testing techniques used for OO code, so there is no need to develop a new testing technique specifically for newly introduced constructs of AO programs. Although this is a good and more convenient approach but there are some limitations of this technique which needs further research, like, it is hard to find the origin of faults in case of any failure in the conduction of testing process. Secondly this technique has been implemented on a toy code by the researchers so it is hard to generalize the results like other testing techniques[24].

4.4.5.2 Analysis

Although authors have tried to presented an easy and convenient way for the testing of AO programs, but on the other hand they have reported the difficulty of understanding the origin of faults in case of any failure in testing process. They have claimed for their technique there is no need to take care of the complex AOP constructs in order to find a way of AOP testing [24]. If the behavior and working of those constructs could not be understood then how can the origin of faults can be identified? Secondly the toy code sample can not be generalized on a large scale software application, so it is very necessary to exercise this technique on bigger and more complex software in order to observe the real time faults detection through this technique.

4.4.6 Control and Data Flow Structural Testing Criteria

4.4.6.1 Introduction

Control flow and Data flow testing is the base of structural testing criteria, because with every iteration in the lines of code the behavior of a program became change. Compiler of a software development language works on the basis of some pre-defined principles. Each and every iteration which happened in a program is validated by compiler and in case of any discrepancy an already stored relevant exception has been thrown by compiler. Iteration in a program deals with two situations i.e. change in the position of line of code and change in the control of that program. First situation is basically the flow of data and second situation is the flow of control. Both these controls compose the structure of the program and test cases are then generated on the base of program structure [11]. By understanding the data flow and control flow the syntax and semantic errors in a program can be handled. Jianjun Zhao [11] in 2006 has presented some techniques to construct effective Control Flow Graph (CFG) for partial as well as complete AO programs. Author has represented the flow of control for methods and advices using CFG, then flow of control for aspects using Aspect Control Flow Graph (ACFG) and then flow of control for the complete system using System Control Flow Graph (SCFG). Above explained CFG’s helps in development of test cases by providing adequate coverage criteria of the AO program [11]. According to Zhao [11] his research work helps in effective regression testing for AO programs and it also support for data-flow analysis and control flow dependency analysis in an AO program.

Lemos et al. [40] have proposed a method for test sequence generation for the dynamic interaction of aspects and classes in 2006. This AOP testing criterion is based on the work already done for the java code. Authors have used a model named Aspect-Oriented Def-Use
Graph (AODU) and they have proposed Control and Dataflow Structural Testing Criteria for Aspect-Oriented Programs (CDSTC-AOP) [40]. In this technique authors have used two kind of testing criteria in combination for the testing of AO programs i.e. Control flow and Data flow structural testing criteria. In control flow testing criteria there are different strategies to be followed in order to provide the coverage of the code under test. In their technique authors have selected the ‘all-nodes’ and ‘all-edges’ code coverage strategies. On the other hand Data flow testing criteria can also be implemented by adapting different code coverage strategies and authors have selected ‘all-uses’ criterion for the code coverage of AO programs. In all-uses criterion there are two kind of coverage criteria all-c-uses and all-p-uses [40]. In their research the authors have claimed that they have answered those question which are raised by Alexander et al. [42] that aspects can be tested without the interference of base class code.

4.4.7 Integration Unit Testing of AO program through IACFG

4.4.7.1 Introduction

A unit testing criteria is proposed by Bernardi and Lucca [13], in 2007 in which authors have showed that AO programs can be tested by estimating the inter-procedural interactions among advices and methods. Authors have based their work on the Inter-Procedural Aspect Control Flow Graph (IACFG) [13]. Due to the un-ordered interactions among the different constructs of AOP with OOP constructs, multiple type of faults may occue and it is hard to understand their origin. Even if those constructs are used in a systematic way again then it is very difficult to understand the origin of faults. It is very necessary to have a complete step by step information of the interaction process of aspects and the base code [13]. According to the authors there were many approaches presented by different researchers in the past for highlighting complexities which could be bring into being by the integration of AOP constructs but those approaches only show the partial behavior of AOP. They claimed that features like ‘structural modification’, ‘field introduction’, ‘around advice’ and ‘aspect precedence’ which were not previously considered now these features are exploited through the testing model, IACFG [13].

In their research authors especially highlight the behavioral change of base class after interacting with aspects. Two kinds of changes have been mentioned first is the ‘inter-procedural control flow based alterations’, which is the change of flow of control of base class code due to the integration at some joinpoints. Second is the ‘System’s structure transformation’, which is basically the modified structure of a class or an interface after the integration of special constructs of AOP [13].

Authors have emphasized in their testing strategy on those faults which are related to pointcut expressions and advices. They have mentioned some faults associated with the relationship of advices and pointcuts with the modules of base code. Following expected faults are proposed by the authors,

- “Picking up the wrong kind of elements to affect
- Wrong identification of the run-time context where to apply the advice logic
- Execution of an unintended advice at a join point
- Wrong order of advices execution at a join point
- wrong alteration of control and data flow at class scope” [13]

For the implementation of their technique authors have introduced six criterian to be followed, these criterian are basically the excercising of the integrational behavior of different pointcuts, joinpoints and advices with the methods of the base code. According to authors by effectively following these criteria in different combinations the proposed faults can be revealed in a systematic way [13].
4.4.8 Unit Test Sequence Generation for AO program

4.4.8.1 Introduction

There are three research papers found in literature [12][36][38] which highlights this testing strategy. Massicotte et al. [12][36] and Mourad Badri et al. [38] have presented the idea of development of test sequences based on diagrams. Authors have emphasized on the usage of UML state diagrams to test dynamic behavior of AO programs [38]. The behavior of more than one aspects and a base class is observed during the incremental integration of aspects and base classes [12][36][38]. In [12] the main intention of authors is to test the resultant behavior in case of integration of multiple aspects with one class and for the integration of multiple classes with one aspect. Authors have presented an iterative methodology in [12] for the testing of AO programs which consists on four steps. In the first step test sequences are generated for the base code before integration in order to identify the faults related specifically to base code. XML is used to graphically represent the base code. Second step contains the iterative integration of aspects to the base code. This step not only track the aspect and class behavior during integration but also dynamically control any unwanted behavioral change [12].

In [36][38] authors have introduced following four types of testing criteria in the presence of four guided conditions,

- Transition Criterion
- Sequence Criterion
- The Advice Execution Criterion
- Multi Aspect Integration Criterion

In the transition criterion there is a condition that any change of state which occur due to the influence of an aspect is tested atleast once. This criterion considers the change of state for only individual variables and it did not account the sequences of the changes. The sequence criterion has covered those sequences of transitions with a defined condition that the sequences of changes must be tested atleast once which are effected by one or more than one aspects [12][38]. Masicotte et al. [36][38] in their research work extended the work of Mourad Badri et al. in a way that they presented enhance criterion to cover the multiple aspects integration with a class, which are as following,

- Modified sequences coverage criterion
- Simple integration coverage criterion
- Multi-aspects integration coverage criterion

After integration process aspects bring in new messages to the classes which ultimately modify the transition sequences in the collaboration diagram. According to this criterion every modified sequence should be re-tested at least once. Simple integration criterion is guided by the condition that if a class method is used in the collaboration diagram which was affected by a single advice then all the sequences in which that method is used should be re-tested. Whereas multi-aspect criterion is followed by the condition that if a method is used with the collaboration diagram which is affected by multiple advices then all the sequences which use that method should be re-tested [12]. This research work mainly based on identification of behaviors which results by the incremental integration of aspects with base classes. Authors have used UML and XML to generate collaboration diagrams which ultimately helps in understanding integrational behaviors between aspects and classes [12][36][38].
4.4.9 Aspect Flow Graph for Testing AO Programs

4.4.9.1 Introduction

This approach is presented by Weifug Xu et al. [41] and according to authors they have introduce a way of testing AO programs through the combination of state models and flow graphs. Their technique is blend of responsibility-based testing and implementation-based testing. This technique combines class state models and aspect state models and then generates an aspect scope state model (ASSM). After getting the output from ASSM about the state transitions the flow graphs for advices and methods are constituted. This output helps to construct an aspect flow graph (AFG). ASSM helps in developing transition trees and with the help of transition trees and AFG effective and efficient code based test suites are then developed. According to [41] this testing strategy concentrates more on the understanding of behavioral interaction between classes and aspects. In their research work authors have first elaborate the behavior of more than one classes after interaction with a stack example. Flattened Regular Expression (FREE) state model is used in combination with the method flow graph to derive the appropriate test suite for OOP [41].

4.4.10 Analysis of all the structural unit testing techniques for AO programs

Structural unit testing, mainly depends on both static and dynamic analysis of source code in order to reveal syntactic and semantic faults. After studying in detail all structural unit testing techniques which are proposed for AO programs, first most important subject which is prominent in research of almost every author is to understand behavior of special AOP constructs and effect of those constructs after integration of AO program with base code [5][7][10][13][18][21][22][23][26][29][34][36][37][41][43]. Whereas there is a research paper present in which authors have introduced a method of testing AO programs using OO testing techniques and without taking into account special AOP constructs [24]. Second important endeavor which is intended by almost all researchers is preciseness and reusability of non-redundant test cases for exercising different testing techniques [3][10][11][16][18][20][23][26][29][39][41][43]. Besides almost all the researchers have recommended the use of control flow and data flow graph in order to generate relevant test cases. Control and Data flow are the basic graphs and many graphs are extended from these two graphs for the effective and efficient development of test cases in different testing techniques for AO programs. Third important thing for the effective and efficient testing technique is the ability to uncover maximum number of faults [1][5][6][7][13][18][19][22][26][27][37][41][42][43]. For this attribute an already developed fault model has been followed generically which was proposed by [42]. This fault model is an important milestone in the area of testing AO programs for better comprehension of AOP related faults. Due to complex behavior of AO program especially after weaving process, it is difficult to pointout the accurate origin of faults. So according to authors of this thesis still lot of research required to know reactions of integrated behavior of different special AOP constructs.

Contributions from various researchers is shown in Table 6. Although many researchers have contributed for unit testing of AO program but after detailed study of given testing strategies it is found that there is no industrial evidence available for these testing strategies. AOP is an emerging programming paradigm and less efforts has been implied on this area. There is a need to develop effective and efficient testing strategies for AO program in order to obtain the advantages it provide as stated in research. Effective and efficient structural testing strategies are also required to understand the origin and cause of the faults, and also to retrieve and rectify the faults which can occur not only due to the integration of aspects and base code but also due to the incremental modifications of aspects. In addition there is a
need for the development and industrial implementation of frameworks and tools for testing AO programs for effective and efficient systematic structural testing.

Table 6: Tools/Frameworks for AO testing

<table>
<thead>
<tr>
<th>Article #</th>
<th>Author</th>
<th>Testing Technique</th>
<th>Tool/Framework/Model</th>
<th>Purpose</th>
<th>Contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Parsanath &amp; Tao Xie</td>
<td>Mutation testing</td>
<td>Un-named Framework</td>
<td>For automatic generation of mutants</td>
<td>Saves tedious human effort for mutant generation</td>
</tr>
<tr>
<td>10</td>
<td>Guoqing Xu &amp; Atanas Rountev</td>
<td>Regression test selection for AspectJ</td>
<td>AspectJ Inter-module Graph (AJIG)</td>
<td>Safe and Precise test case selection</td>
<td>By comparing old and new graphs, precise test cases are selected</td>
</tr>
<tr>
<td>13</td>
<td>Bernardi &amp; Lucca</td>
<td>Integration testing based on IACFG</td>
<td>IACFG</td>
<td>For Inter-procedural interactions among advices and methods</td>
<td>Reveals the maximum advice-method faults</td>
</tr>
<tr>
<td>16</td>
<td>Tao Xie et al.</td>
<td>Unit tests generation for AspectJ</td>
<td>Raspect</td>
<td>Detection of redundant unit tests</td>
<td>Non-redundant test selection reduced the testing efforts</td>
</tr>
<tr>
<td>19</td>
<td>Chuan Zhao &amp; Alexander</td>
<td>Fault Based Testing</td>
<td>Fault Model</td>
<td>AO program testing using fault model</td>
<td>Covers maximum AspectJ fall types and support systematic &amp; efficient testing of AspectJ</td>
</tr>
<tr>
<td>20</td>
<td>Tao Xie &amp; Jianjun Zhao</td>
<td>Unit tests generation for AspectJ</td>
<td>Aspectra</td>
<td>Automation of generation of test inputs for AspectJ code behavior</td>
<td>Efficient reusability of existing test cases and reducing tedious manual efforts for test cases generation</td>
</tr>
<tr>
<td>29</td>
<td>Guoqing Xu</td>
<td>Regression testing technique</td>
<td>RETESA Framework</td>
<td>To implement Regression test selection technique</td>
<td>RETESA helps in precise test case selection</td>
</tr>
</tbody>
</table>
5 EFFECTIVENESS OF STRUCTURAL TEST TECHNIQUES FOR AO PROGRAMS

5.1 Introduction

Software testing is a cost consuming standard practice in the software industry which usually costs between 40 and 80% of the total development cost for safety critical systems [54]. Reusability of software components is measured in terms of number of possible configurations they can work in. Software testing methods that are being used are heuristic in nature and follow ad-hoc principles. Testers normally base their test cases on functional requirements in most of systems and these test cases are not sufficient for thorough testing. Researchers have given various test methods to test programs. The decision like, which test method is more efficient in terms of finding faults for a particular kind of software, is very tricky. This decision is important for saving the overheads of cost, time and effort on testing, and for improving the quality of software [54]. Very few researchers have focused their research on this particular area.

5.2 Evaluation of the effectiveness of test technique

Effectiveness of test technique can be compared in terms of their ability to find different types of faults [6]. Weyuker [55] has discussed effectiveness of test technique as “effectiveness of a test technique is only possible to measure if you can compare two techniques for the same set (i.e. software), but the result is not general” [54].

Eldh et al. has mentioned three approaches for comparing and evaluating testing techniques that are commonly used [54]:

1. The use of coverage criteria to compare test techniques. This method is often commonly matched or made-up for a particular technique.
2. Comparing two or more techniques of same group with each other, also called intra-family comparisons. This method is commonly to compare a new technique that is evolved from the original technique.
3. Comparing a particular test technique with others test technique of different family of techniques, also called inter-family techniques comparison.

Any test technique can reveal faults and failures. The difficult aspect in evaluating a test technique, is estimating how effectively the test techniques do that.
5.3 Method to Evaluate Test Technique Effectiveness

Eldh et al. [54] has given a framework based on mutation testing and they claim that their framework can be used to compare different software test design techniques in general. In [54] authors have given the step by step process for their proposed framework. This framework is aimed at investigating different variants of test techniques in a particular group. The steps proposed for the framework are following [54]:

1. Prepare samples of code by injecting known faults.
2. Select a test technique.
3. Perform experiment i.e. apply test technique and collect test results.
4. Analyze data, compare testing techniques and evaluate results of collected data and repeat experiment with new code samples (with new faults injected) and collect sufficient data for important measurements like efficiency and effectiveness of test techniques.

These steps are shown in the figure 6 below.

Figure 6: Overall process of evaluating test design techniques [54]

Eldh et al. argue that faults can propagate at different levels during testing process so test techniques efficiency can be different at different levels so it is important to know at which level test technique is more efficient. The frame work proposed in [54] involves fault seeding so it is important to have knowledge about different types of faults and the failures these faults represent. Eldh et al. [54] state that there exist no straightforward classifications which can be used to decide, what faults to inject in an experiment for test techniques evaluation.

Test techniques are usually evaluated using small code samples because it is manageable and practical. On the other hand the result using small code samples are not scalable and cannot be generalized. In industry there is lack of well founded guidelines to decide that when different test design techniques are effective, efficient and applicable [54]. Authors have mentioned in [54] that it is difficult to efficiently instrument and measure data-
flow coverage and control-flow coverage for complex systems. The analysis and evaluation of test results in the test techniques evaluation experiment, is subjective and can involve reactivity, researcher bias and respondent bias [54].

5.4 Effectiveness of test techniques for AO programs

Literature study reveals that there is no method or framework to evaluate the effectiveness of test techniques for AO programs. A.A. Naqvi et al. [1] have surveyed three test techniques developed for AO programs for effectiveness. The test techniques include data flow based unit testing of AO programs, state based testing, and aspect flow graph based testing. Reference [6] has surveyed the effectiveness of five testing approaches for AO programs, the test techniques include data flow based unit testing of AO programs, state-based approach, aspect flow graph, unit-testing aspectual behavior, model based approach to test generation for AO programs.

5.4.1 Effectiveness Evaluation Criteria

In both of studies [1] and [6] authors have used the same criteria for examining the effectiveness of test approaches. This criterion is the fault model developed by Roger et al. [42]. This fault model is based on different features of AO programs and explains the six types of faults that an AO programs written in AspectJ may have. This fault model is briefly explained as following:

5.4.1.1 Incorrect strength in pointcut patterns

Join points of a particular type are selected according to signature that include pattern. These specifications of signatures are contained in pointcuts. A concern C can have matching join point J. A pointcut P has each matching join point J associated with advice for concern C. Pattern strength determines the join points to be selected. Too strong pattern will not select some necessary join points and too weak pattern will select additional unnecessary join points. If pattern is strong or weak, in either case is likely to cause incorrect behavior of the woven target code [42]. Woven advice statements and the statements that are executed after woven join point determine whether the strength error of pointcut will introduce any fault [1] [42].

5.4.1.2 Incorrect aspect precedence

Situations in AO program where more than one advice from different aspects are affecting the same join point. In such a situation it is important to control the order of execution of advices i.e. the order in which the advices are applied to join point [1] [42]. The order control in which advice is being applied can be established by assigning the precedence order to various aspects whose advice is affecting join point [1] [42]. In case if no precedence is assigned to aspects then the precedence between the aspects is arbitrarily determined and thus shows up unpredictable behavior upon execution of AO program[1][42].

5.4.1.3 Failure to establish expected post conditions

Aspects have the ability to change the flow of control of class’s code; such a change can result in a class not being able to fulfill its post conditions for the class contract [1] [42]. The clients of that class (core concern) expect those core concerns (class) to behave according to the contract and method post conditions to be fulfilled irrespective of whether or not aspects are weaved with the concern [1][42]. Thus to behave correctly, woven advice must not disturb methods of core concerns to satisfy their post conditions. Writing code for
such an advice is difficult challenge for aspect programmers and can be a source of errors [42].

5.4.1.4 Failure to preserve state invariants

Methods of a class must ensure that state invariants are satisfied in addition to the post conditions of methods [1] [6]. The behavior of a concern is defined in terms of physical representation of its state and methods that act on that state [6] [42]. AOP enables aspect programmer to introduce new methods and instance variables into a class (core concern) using ‘introduction’ concept. ‘Before’, ‘After’ and ‘Around’ constructs of AOP can be used to introduce additional behavior that is not visible to the class being affected [1]. These constructs of AOP if used in AO program, can introduce new states and can cause a class to violate its state invariants, after weaving process and can be another source of errors [42].

5.4.1.5 Incorrect focus of control flow

A pointcut designator selects which of a method’s join point to capture, at the weave time [6]. In some situation this information to select a method’s join point is available at runtime [6]. In some situations joinpoints should be selected in a particular execution context, this context can within control structure of particular object or within the control flow that occurs below a point in the execution [6] [42].

5.4.1.6 Incorrect changes in control dependencies

When around advice is applied on any method it can significantly change its control flow [42]. New code is inserted in the method and new branches appear that alter the dependencies among statements and new data may also be inserted [6]. In such situation control dependencies among statements in case of AO program should be represented and tested [6] [42].

Data flow based unit test technique for AO programs developed by Zhao [3] is one of the first test approaches developed for AO programs. Zhao has described in [3] how combination of the unit-testing and data-flow-based testing can be used to test aspects and those classes that may be affected by the aspects. For each class or aspect testing is performed at three levels intra-module, inter-module, intra-aspect or intra-class level testing. This technique uses control flow graphs to compute def-use pairs for aspects and classes; algorithm for building control flow graph is given in [3]. The issue with this technique is that around advice can alter behavioral control or data dependencies of methods which mean that the def-use pairs of methods can be altered by around advice as suggested by [42]. This testing approach is capable to find ‘failure to preserve state invariants’ faults because these faults are associated with erroneous data flows [1].

A state-based approach to test AO programs is given by Nygard et al. [43]. Authors of this approach claim that suit of test cases can be generated in terms of sequences of messages to adequately test object behavior and interaction between classes and aspects. To achieve possibility of adequate testing authors have transformed aspektual state model to a transition tree, and each branch of the transition tree from root node to leaf node represents a template of test case. Authors of this approach extended FREE (Flattened Regular Expression) to ASM (Aspectual State Models). The issue with this approach is it suffers from state explosion problem and it is unable to find faults in aspect composition [43]. This strategy can reveal state transition faults; corrupt states sneak paths [1]. According to [1] state-based approach can reveal two types of AO program specific faults out of six mentioned in [42] and these include incorrect strength in pointcut patterns and failure to preserve state invariants.
An aspect flow graph (AFG) [41] is a test technique that uses a hybrid testing model which combines responsibility-based testing model and implementation-based testing model. AFG is constructed by combining Aspect Scope State Model (ASSM) and the flow graph for methods and advices. ASSM is a state transition diagram for an AO program and it is in turn created from state models: 1) FREE state model which state transition diagram and 2) Aspect State model which comprises of four states and three edges. Flow graph for methods and advices represent the flow of control within the methods and advices. Aspect Flow Graph for an AO program makes it possible to apply number of white box testing strategies and test coverage criteria [41]. The main contribution of [41] is test suits created using aspect flow graph are manageable, code based and executable [6].

The issue with AFG test approach is that it is unable to handle dynamically applied advices, i.e. the advices that are applied depending on a particular execution context [6] [1]. In reference [41] authors claim that aspect flow graph test approach is able to detect two types of faults in candidate faults model proposed in [42]. These fault types include: incorrect focus of control flow and incorrect changes in control dependencies. But according to the study by Naqvi et al. [1] aspect flow graph test approach for AO programs is unable to detect faults of incorrect focus of control flow. In [1] authors argue that aspect flow graph is model for static weaving and faults of incorrect focus of control flow arise due to dynamic behavior of AO program so it is unable to detect this kind of faults.

According to research literature the test techniques for AO programs are able to detect some of the faults mentioned in [42]. The following table 7 shows the test techniques and faults types that it is able to detect and faults types that it is unable to detect.

<table>
<thead>
<tr>
<th>Test Technique</th>
<th>Detects Fault Type</th>
<th>Unable To Detect Fault Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data flow based unit testing of AO programs</td>
<td>D</td>
<td>A, B, C, E, F</td>
</tr>
<tr>
<td>The State Based Testing of AO program</td>
<td>A, D</td>
<td>B, C, E, F</td>
</tr>
<tr>
<td>Aspect Flow Graph based Testing</td>
<td>F</td>
<td>A, B, C, D, E</td>
</tr>
</tbody>
</table>

Faults types:

A. Incorrect strength in pointcut patterns.
B. Incorrect aspect precedence.
C. Failure to establish expected post conditions.
D. Failure to preserve state invariants.
E. Incorrect focus of control flow.
F. Incorrect changes in control dependencies.

The table 7 shows none of above mentioned three test techniques is able to detect fault type B (incorrect aspect precedence), fault type C (Failure to establish expected post conditions) and fault type E (incorrect focus of control flow).

Fault type D (Failure to preserve state invariants) is the only fault type which is addressed by more than one test strategy, no other faults is addressed by more than one test technique. Results of table 7 show that state based test techniques is most effectiveness of the all three because it has the ability to detect two kind of faults i.e. type A and type D. There is no empirical evidence on evaluation of effectiveness of the any of above mentioned test techniques. All the above mentioned facts in table 7 are based on the surveys by [1] and [6].
6 EPILOGUE

This chapter presents the conclusion of the master thesis report and it also explains the area where future work might be needed.

6.1 Conclusion

This thesis presents a systematic review of AO software testing in the context of issue/challenges, techniques and their effectiveness. The main contribution of this thesis is that it provides detailed, yet brief state-of-the-art research on AO software testing in the defined domain over the last decade from 1999 to 2008. This systematic review is based on the guidelines proposed by Kitchenham [44] and it has fulfilled all the requirements of standard systematic review.

The thesis has answered all the research questions and has achieved its aim and objectives. The number of publication on testing AO software started to rise after the year 2003 (See section 2.2.2) therefore this attempt of presenting systematic review can be considered as timely help and support for research community. Systematic review results could be of interest to research community as it contributes in identification of the areas where research lacks in the domain of AO software testing issues, test techniques and their effectiveness.

Systematic review results reveal that AO program testing challenges relate to the way AO programming technology is used. If the program is written using OO paradigm and concerns cross-cutting multiple modules are handled using the AO technology and aspects do not change the control and data dependencies of primary concerns i.e. classes, then there are less testing challenges in testing AO programs. This way of using AO technology in fact complements the programs written in mature programming paradigms like OOP. If the AO implementations for cross-cutting concerns are written independent of base classes and base classes’ implementation functionality is over ridden by aspects this creates serious test challenges for AO programs (See section 3.3) as well as it would result in bad structure of the program.

The issues pertaining to testing AO programs become more complex due to complex nature of weaving process especially for the AO systems having larger number of interacting aspects involving multiple common base classes. It becomes hard to locate and isolate the faults in the artifact of weaving process. In addition weaving process makes it hard to determine test adequacy for AO programs (See section 3). Aspect can change the structure and behavior of its related classes. Aspect developer has to check the code segments where the aspect has impact on its related classes and weaver has limitations in providing information about it. So aspect developer has to do it by himself/herself. The studies on evaluating the coverage criteria ability to detect faults have shown that test coverage criteria have been better in detecting particular kind of faults for small sized AO programs.

There are fewer number of structural testing techniques developed and proposed (See section 4) for AO programs and existing test techniques that are being used for object oriented program testing cannot be used to test AO programs. Only one study has been done which proposes the possibility of applying OO test method to test AO program but the study is done for small size program and there are no empirical studies in research literature showing how it can be done for larger AO programs (See section 4.3). The test
techniques developed for AO programs offer a serious limitation in handling the type of faults that occur in AO programs (See section 5).

There is no method or framework proposed in research literature to evaluate test technique specifically developed for AO programs. Eldh et al. [54] has given a framework which the author claim is general framework to evaluate test techniques but it does not describe the semantics of this framework for AO program.

Research on evaluating the effectiveness of test techniques for AO programs is based on surveys and results from these surveys suggest that state based test technique is only technique which is able to cover two of six fault types described in the fault model [42] for AO programs. There is not a single test technique developed for AO programs which can detect faults all the faults types mentioned in [42]. Moreover empirical studies have not been done in the area to evaluate the effectiveness of test techniques for AO programs.

6.2 Validity Threats

Validity threats are factors that can influence the accuracy of research design and it is important to identify and tackle these factors to make results reliable.

Conclusion validity also called reliability is concerned with the results of research study and it ensures that these results are reliable and can lead to correct conclusion. It pertains to the issues that affect reliability of results [57]. The main purpose of Review protocol in systematic review is to reduce the researcher’s bias [44] and to present clear and transparent methodology for systematic review. The review protocol (See section 2) was reviewed by our thesis supervisor who himself is a researcher and a faculty member at BTH. Relevance of search strings and search resources was examined by conducting pilot searches in consultation with the librarian of BTH, Sweden.

Aspect oriented programming has not gained confidence of software industry and there is little use of AOP reported yet for building big and complex software systems. So this aspect has not given authors an opportunity to gain industrial perspective of testing AO software and gain insight of the issues involved in the industry. So conclusions made in the study, are based totally on the research literature.

Internal validity is concerned with the evidences of the claim in research design and it allows researchers to draw conclusion from the causes and effects [57]. In systematic review the internal validity involved is publication bias. Authors have tried to strictly follow the quality assessment criteria mentioned in review protocol for the primary studies involved so that both positive and negatives aspects of the claim in primary study can be assessed and presented in the systematic review report. In this way authors have minimized the internal validity threat.

External validity also referred as generalize-ability, is to analyze, how general are the results of the research design [57]. Authors have proposed systematic review which follows a clear documented methodology, but as authors have taken research literature into consideration so the results presented in systematic review lack input from industry so in order to make the results of this study generalize-able, industry experiments are needed to validate and generalize the results.

Construct validity is to recognize association between theory and surveillance [57]. This systematic review has been hypothesized in the review protocol and after completion of
reporting phase it has successfully achieved its aim and objectives by answering the defined research questions. In this way it has productively verified the construct validity.

6.3 Future Work

This section reports the areas that are considered important for future work. Weaving process of aspect and base classes adds complexity in understanding the structure of the program and further study is needed to know if the complexity due to weaving process can be predicted, how can we model the effects of a set of aspects, on primary abstraction. This will make the effects of weaving process more explicit and apparent.

The confidence that software is adequately tested is important for the release of software product, for this it is important to know how we can adequately test AO program, what new test criteria must be defined to adequately test AO programs. The study results show that test criteria for AO program for evaluation of test technique demands a systematic methodology to empirically assess the ability of a particular test technique to detect the faults of AO programs. Code reusability is critical aspect to minimize the cost of AO software so for the import of aspects it is important to know the semantic impact of reweaving foreign aspects.
7 REFERENCES


Appendix I

Terminology
This section gives brief definition of the terms used in the thesis report.

Aspects
There are various concerns which are difficult to contain in one class while working in OO paradigm, for example logging code is implemented in a single class i.e. server class but it is required in different client classes. In this logging example code is spread all over the system and tightly coupled with each other [35]. The concerns which are tightly coupled throughout the system are called ‘cross-cutting’ concerns. Basically these cross-cutting concerns when put in a nutshell that nutshell is called an ‘Aspect’. These are the self-sufficient units which contains point cuts and advices [58]. ‘Software properties that are not directly represented by the functional decomposition of the software system are expressed as aspects’ [42].

Tangled code
This term is used to explain the dependencies of code in different software components. There are some tasks in complete software and to perform those tasks some specific lines of code are present in almost every module of software, this will increase the coupling and redundancy in the different modules. So basically that redundant code is called tangled code.

Crosscutting concerns
OOP has a limitation that it cannot handle the modularization of the code scattered in different components. So there is a problem of code duplication in different modules which not only increase the workload of developers but also increase the size of the software. Nowadays there is a continuous change in the software requirements especially in the versioning of market driven products. It is very much problematic for programmers to modify the code in order to include some additional functionality after the completion of software development work. AOP can better manage this issue as to alter some specific code an AOP programmer do not need to go into all the components however he/she only has to deal with the distinct aspects [58].

Weaving
This is the merger of cross-cutting concerns with the base code (business logic) [35]. Weaving process involves join points and point cuts which support in the integration of both aspect and base code. According to [35] AOP do not change the source code after the weaving process, but on the other hand it leaves some impact on the compiled code.

Joinpoint
While execution of an AO program the AspectJ compiler identify some positions in a flow of a code at which there is an execution call for some predefined action e.g. method calls etc. these positions are called joinpoints [35]. These are events which occur during the runtime execution of a program. Joinpoint is used to introduce some new functionality in an existing system by weaving that functionality. In AspectJ language Joinpoint did not used as a key word however this is a concept. In a program flow the joinpoints happened during the
execution time. According to [35] there can be 11 types of joinpoints in AspectJ language which are used in different scenarios e.g. method call, method execution etc.

**Pointcut designator (PCD)**

This is basically the exact place at which a joinpoint is required [27]. In a flow of a program pointcut mentioned the place in the base code where the before, after or around advice has to be implemented, in other words pointcuts basically holds different joinpoints [35]. A pointcut can be either named or nameless. and pointcuts cannot be overloaded, and there are different types of PCDs.

**Advice**

This is the whole body of that method or exception which we need to include in the flow of code for the purpose of handling some cross cutting concern [27]. According to [58] Advice is a piece of code which indicates that what to do by specific pointcuts at different joinpoints. Advice can be of three kinds,

- **Before Advice**: Before Advice is executed before the execution of that pointcut which will be selected by joinpoint.
- **After Advice**: as the named shows After Advice execute after that pointcut which has been selected by a specific joinpoint. After Advice is complicated sometime because it is very hard to decide its execution precedence in case of exception thrown by the system. AspectJ provide three types of After Advice to manage this issue.
  - After returning advice: this Advice will run only in the case if there is no exception thrown by the system.
  - After throwing advice: when After Returning Advice did not run and an exception is throw by the system then After Throwing Advice executed.
- **Around Advice**: this is very powerful advice type in AspectJ language, around advice has the ability to change reached joinpoint with some randomly selected code in the program execution.

  Proceed is a special type of variable, this is used to facilitate the around advice.

**Introduction**

This is a very important construct in AOP, this is used to add or alter the features of the base code during the weaving process of both kind of programming paradigm. The behavior of base class code changes and it did not retain those post conditions which were expected in the initial modeling of base code. Introduction where helps in good implementation of advice code at the same time it can create faults which are mentioned by Alexander in [42].

**Domination**

This is a special construct in AspectJ, basically domination deals with the precedence purpose of different aspects and their advices. The rule implied in AspectJ is that the child has more priority than the parent [58].
Terminologies for unit testing strategies

Data flow based unit testing

Control flow graphs are used to implement data flow based unit testing technique for the exploration of anomalies of data in a software. There are different testing strategies in a data flow based testing technique and the basic purpose is to ensure the coverage of definition and usage of each and every data variable in first step. After that it is also necessary to track the paths of a control flow graph which is developed due to the interactions of all the variables and modules in software [3]. Data flow technique mainly focuses on the initialization, definition, release or killing and use of different variables. Basic purpose to track all the above mentioned activities is to develop effective test cases which provide complete test coverage for a specific task.

The usage of variable can be of two types i.e. computational use (c-use), when any variable is used in a statement which compute values and other is the predicate use (p-use) i.e. when a variable is used in decision for selection of a specific path to follow for example the usage of a variable in an ‘if’ condition is known as p-use.

According to [59] an effective and efficient CFG generates all the possible combinations in form of different nodes and edges which actually represent the definitions, c-uses and p-uses of a specific variable. Any structural testing strategy can be adapted to check the transitions of variables after initialization. To evaluate variable initialization, its sequential transition and the current position of a variable a combination of strategies can be used which is known as def-use (du) association of a variable. Following are some strategies which can be adapted individually and in combination for evaluation the variable position in a program.

All definitions: it means that in a program every variable and the definition of every variable should be tracked at least once including of its computational and predicate usage [59].

All-c-uses/some-p-uses: this is to track at least once the computational use of every variable and every definition of that variable. If this will not cover the all definitions only then the predicate use test cases can added to provide complete coverage [59].

All-p-uses/some-c-use: in this strategy test cases are developed by tracking at least once the predicate use of every variable and every definition of that variable. Additional test cases can be included if this strategy will not cover all the definitions of those variables [59].

All-c-uses: this is almost the same as All-c-uses, except the usage of additional p-use [59].

All-p-uses: this is almost same as All-p-uses, except the additional usage of c-use [59].

All-uses: in All-uses at least one path is tested from every definition of every variable to each of its usage. According to [59] All-uses although contains almost twice than any other number of test cases, but it is a very good usage for getting better results and data flow testing with All-uses is the best combination for code coverage.

All-du-paths: this strategy requires All-du paths should be exercised under test by every definition of every variable to its every usage. According to [59] this is the strongest data flow strategy for structural testing.
Figure 7: Relationship among Data-Flow strategies [59]
Appendix II

List of Rejected Articles


