

An Approach to Test Aspect-oriented Programs

M. N Qamar^{1*}, A. Nadeem², R. Aziz¹

Abstract— Software testing is a perennial problem, consequently it scores scant attention. An inclusion to testing challenges is aspect-oriented paradigm, which has a dichotomy of core and crosscutting concerns. Since emergent behavior of the aspects during their interaction with objects, and inter dependencies not only incurring challenges for testing, but also alludes to creation of innovative testing techniques. Several faults are introduced by aspects. In this paper, we have surveyed all the existing testing techniques including static verification techniques for aspect-oriented programs. These factors appeal us to devise a comprehensive testing framework to meet obliviousness of harmful aspects. Various approaches are lacking in automation or either can't deal with testing of large programs. This paper suggests the need of regression testing by analyzing the prevailing testing techniques for aspect-oriented programs, and investigates efficacy of regression testing. The paper proposes an architecture of a framework (Aspect Driven Regression Testing framework) ADRT, and underlying details for testing of aspect based application, and a tool, (Objects-Aspects Regression Analysis) OARA for automation of testing process for aspect-oriented programs.

Index Terms—Aspect-oriented programs, regression testing, test case generation, regression faults

I. INTRODUCTION

Aspect-oriented programming is a new methodology claiming enhanced concepts. Expected software design, and acquiring maximum modularity leads us to the most innovative, and seminal aspect-oriented paradigm, as it exhibits the effectiveness, and robustness of required level. Code scattering and code tangling are some of the major vulnerabilities, which any other paradigm has to come across. Aspect-oriented programming provides effective solution for aforementioned bottleneck. The separation of concerns is under focus in aspect-oriented paradigm to avoid code scattering, and tangling, by treating separately core and crosscutting concerns at design level. Implementation tools are e.g., AspectJ [28], AspectC++ [27]. The main constructs, added to object-oriented paradigm are; join points, pointcuts, advice, introductions, and aspects. In aspect-oriented technology, an aspect is woven into core concerns; aspect is realized as a crosscutting concern. The aspect code observes the base program, and when certain pointcuts reach, the aspect code is woven there. Several tools

can be found out, e.g., AspectJ [28], AspectC++ [27], and HyperJ. Aspect based programming is not matured yet, so analysis of aspects regarding their issues can be found in [2], [3], [4], [5] [6], [7]. One of the attributes of aspect-oriented paradigm is weaving process, which is called *dynamic weaving*. Although it is not cost effective [1], but many people are working to cater tool issues, which of course will be solved, as just one in [1]. AspectJ is a tool to write aspects, solves it up to some extent.

Though aspect-oriented focuses on design problems, and promises to develop best solution but absence of errors can't be assured by any paradigm. Ultimately developments of efficient testing techniques, and tools, which assist in the creation of high-quality software, have become one of the most important research areas. Researchers often demonstrate the effectiveness of their techniques using tools that function on contrived or toy systems. We are in need to perform additional research that provides analytical, statistical, or empirical evidence of the effectiveness of the test-selection criteria in revealing faults [31], specifically for aspect-oriented programs. Exhaustive testing is impossible in both principle and practice. Bach suggests that testing should be based on circumstances [30]. Apparently some techniques, which have been proposed for aspect-oriented programs implicitly accepting the testing criterion flaws, and don't have practical aptitude. While valuation of pertinent object-oriented testing techniques show their inadequacy for aspect-oriented testing or even using them as it is for aspect-oriented programs, a dissent has been shown by the behavior of aspect-oriented programs.

In this paper, we have surveyed several testing strategies. It is obvious that found solutions are infeasible or their automation seems impractical. This paper presents a solution considering the two main aspects, i.e., automation, and testing adequately (w.r.t faults) considering the reuse of test cases. ADRT (Aspect Driven Regression Testing) is presented for sufficient testing of aspect-oriented programs. Besides regression fault, the proposed approach ADRT comprises of three components, which assist overall testing activity by using their own scheme. They are equally useful for OARA (Objects-Aspects Regression Analysis) for automation. ADRT framework contains following components:

- (i) Testing Support Architecture (TSA)
- (ii) Test Architecture (TA) – testing environment
- (iii) RF (Regression Faults)

This paper attempts to imbibe a general overview of static testing techniques for aspect-oriented programs. A detailed discussion also resides in the paper about testing issues of aspect-oriented programs, tradeoffs, and then analyzes the testing techniques to see whether they are sufficient, and capable whether they support testing of large programs and automation.

M. N Qamar¹ is with the COMSATS Institute of Information Technology, Islamabad, Pakistan and a senior member of Centre for Software Dependability (phone: +923455893722; fax:+92519257164; e-mail: nafees_qamar@comsats.edu.pk).

A. Nadeem is with Centre for Software Dependability, Muhammad Ali Jinnah University, Islamabad, Pakistan (email: anadeem@jinnah.edu.pk)

R. Aziz is with the COMSATS Institute of Information Technology, Islamabad, Pakistan (e-mail: romana@comsats.edu.pk).

Paper is organized in the following way: Section 2 states related work. In Section 3 architecture of the proposed framework for testing aspect-oriented programs including ADRT, TSA, TA, RF. Section 4 explains the implementation details for ADRT using a tool, OARA, and its structuring. Finally, section 5, future directions, and conclusion have been discussed.

II. RELATED WORK

Robust design features of aspect-oriented paradigm has many proofs, e.g., [18], [25], [27], but unfortunate perspective is that the testing frameworks are engineered using aspect-oriented paradigm (especially due to its aspect part characteristics) e.g., [18], [21], [22], but aspect-oriented programs itself to test is an issue. The methodologies, which exist for testing of aspect-oriented programs, other than the above, are [9], [10], [11], [12], [13], [14], [16], [17], [18] [20]. The general discussion about these techniques is as follows:

The work presented by Zhou et al., in [9], discussed the approach of testing is in four steps which explores woven application. In first step, class testing is done, then aspect is tested as unit, in third step integration testing, and in the last system level testing of the software is exercised. Their technique requires more causal details. Another technique presented in [17] focuses on unit testing framework, and test oracles from aspects in aspect-oriented programs. In their approach, top level aspects are picked from generic aspects, residing in aspect-oriented program. A language AOTDL (Aspect-Oriented Test Description Language) is used for building of application specific aspects for testing. Zhao, and Martin [14], are extending their previous work based on object-oriented programming, as SDG (System Dependence Graph) is constructed for Aspect-Oriented programs. A bit similar approach is presented by Weifeng et al.,[11], where they create AFG (Aspect Flow Graph) by combining state models (class & aspect) with flow graph (method & advice) for aspect scope coverage, for building test suites. It is a hybrid solution, which comprises of responsibility based testing model, and implementation based testing model. Another technique, introduced by Xu and Xu present a state-based approach for testing aspect-oriented programs [29]. Program slicing is an effective method to perform testing process; Zhao does a similar work. ASDG is built (Aspect-Oriented System Dependence Graph) to represent aspect-oriented programs, found in [24].

By Zhao [12], a data-flow-based unit testing of aspect-oriented programs is introduced. But according to Rapps, and Weyuker; it is inadequate to just examine the control flow of the program [23]. Another very similar approach [19] by Otavio et al. is proposed as a model using control and data flow testing criteria. These concepts would be implemented on woven artifacts. [7] considering aspect as unit, have been classified. They developed a tool JamlUni for aspectual behavior. Tensen, and Alexander [16] presented a testing approach, which is a hybrid technique based on two methods, i.e., coverage, and mutation testing. Static analysis is done for aspect code fragments. They introduce a set of mutation operators to evaluate if a test suite is sufficiently

sensitive to find errors in pointcuts and aspect precedence. A well-versed approach is presented by Rajan, and Sullivan [13]. They proposed language-centric approach to automate test adequacy analysis, called them as *concern coverage*. In their approach, they claim that it is possible to do explicit, precise, abstract, and machine-readable representation of the tester's intent, so that it can ease the testing by eliminating the need for manual selection, and explicit maintenance of test adequacy criteria. Some other techniques like [8], [2], [4], [5], [6], [32], [3], [1] but reliability through formal specification and verification but only focused chunks of problems, more instances are [33], [17], [12], [9], [11], [13], [16].

III. PROPOSED APPROACH: ASPECT DRIVEN REGRESSION TESTING

This section describes the motivation for ADRT as well as the architecture of the proposed framework, classification of aspects, regression faults, and components of ADRT, i.e., TSA, TA, RF to adequately test the aspect-oriented programs.

From the regression testing perspective, for aspect-oriented programs, we treat core concerns as baseline components, and aspects as delta components. ADRT (Fig.1) is based on the regression faults, which are basic notion for implementing ADRT. Let us how it works:

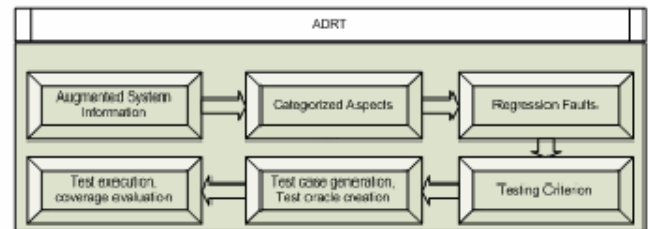


Fig. 1. Abstract level components of ADRT

ADRT starts from the woven application, keeps the records of the woven aspects, and imminent faults are related to those aspects. Based on the testing criterion, the testing environment is set, keeping in view the pros and cons of application to be tested. ADRT addresses specifically the test cases generation, test cases reuse, test oracle creation, test execution environment and the coverage evaluation. Aspects categorization and regression faults are very significant factors of ADRT. After looking in to those in next discussion, we will see how the ADRT components e.g., TSA, TA and RF take part in ADRT. They have been explained explicitly in this section.

ADRT is designed in an impressive fashion to detect faults; therefore, harmful aspects are easy to witness. We apply regression testing on the woven system because this testing strategy focuses on to locate undesirable changes (errors made by aspects) in the new system. To analyze the impact of aspects on the correctness of core concerns (which are free of language compulsion) we can adopt this regression testing framework. Generally, regression testing is applied on system, subsystems, and modules. It is performed to correct faults, improvements in functionalities, or to adopt it to environmental changes as aspects made to existing system. First the technique will

analyze the regress on the core concerns considering the flaws that can occur in woven systems and those faults are considered as Regression Faults. Binder [35], regression testing is affective for revealing faults by the two components, which are named delta/baseline component. Robert also concludes that despite some limitations, regression testing plays an important role in revealing bugs. Through regression testing we may able to find delta/ baseline incompatibilities, and undesirable feature interactions between a baseline, and a delta. All the other approaches, which exist for aspect-oriented testing, are based on using the object-oriented techniques with some enhancements or replacements. Though regression testing is also one of those techniques, which is being used for object-oriented testing, but it is purely addressing the regression faults, which can occur. So the best suitable technique towards testing of aspect-oriented programs would be to choose regression testing based on the different faults that can occur in woven system.

This approach comprises of five main characteristics:

1. As mentioned above, we are classifying the faults, which would not only be used for the current system, but their *reusability* would also be a distinctive factor of this approach.
2. Secondly, the technique facilitates the testing process, as the faults always stand with programmers. Through classification, and providing the programmers flaws as checklists, the automation is strengthened and overall strategy designs a suitable environment for testing purposes.
3. Its automation tempts towards practical use for testing of large programs.
4. It also encompasses the facility of testing classes with any suitable technique, which would be considered as a best one by the testing group.
5. Components, like TSA, TA, RF, and OARA overall distinctive features of our approach.

3.1 Categorization of Aspects and Their Behavior

The aspects' classification [33], a detail is resides over here to make their use in TSA. Just a few aspects with their details are as follows:

Invasive aspects have ability to change the values of variables in the underlying system, and change the transformations (actions) of the underlying system.

Spectative aspects do not influence the values of underlying variables or conditions for underlying events, and only gather information in local variables.

Regulative aspects can affect the control of the underlying system (e.g., by short circuiting underlying computation sequences [8]) in addition to the capabilities of spectative aspects.

The first six faults are proposed by Alexander et al., [20]. While as are we working on large software application to demonstrate the use of our tool, we identified some other faults as well, which have been enlisted below.

3.2 Regression Faults

“A regression fault occurs when both a stable baseline system B and a delta component D pass individually adequate test suites, but fail when used together.”[34]

Counterparts of Aspect-oriented paradigm are normally adopted by the industry, and this compelling practice is helpful in analyzing these, but conversely aspect-oriented programming doesn't have that much advantage. We need to realize its benefits for software applications, so its other dark sides may be addressed. Following are some of faults, which show the strength of regression testing for aspect-oriented programs. Gradually the repository of regression faults grows, as the applications are built. Further faults can be seen in [35]. Regression testing has shown its validity and effectiveness for object-oriented programs but regarding aspect-oriented applications there doesn't exist any methodology to test the programs.

In ADRT, by gathering the information of aspects, which have been implemented on the core system, is being utilized in TSA component, regression faults (RF-described above) are those that are consequences of woven system.

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

In addition to above fault model, following are the other regression faults, which we observed:

- Faults of Exceptional control flow changes
- Faults of Inter-type declarations
- Faults by polymorphic calls
- Join points faults
- Any undesirable feature interaction between object, and the aspect. There may arise situations where the behavior is completely unobservable.
- Incompatibilities among aspects and classes even if the requirements and the implementation have not changed.

While TA supports test cases reuse, generation of new test cases, and test data generation issues and problems. Distinctive feature of this framework is automation.

3.3 Testing Support Architecture (TSA)

The TSA (Fig. 3), part of ADRT provides the basic information regarding application for ADRT, and also for the automation process of large applications. Usually the augmented system is built by the weaving of aspects on the classes. Numbers of regression faults have been described for aspect-oriented programs (see regression faults), while it is likely to grow when large applications are built using the aspect-oriented paradigm. Complete information is maintained in TSA for the faults, which are obviously their by different kinds of aspects. Still the classification [33] is not that much mature, which can possibility classify all types of aspects and

sources of errors, but of course industry practice would add to it more. In related work many testing techniques have been described or even through static verification, which can test aspect as single entity [3], and obviously there is no question about testing of classes as single units. The major characteristic of TSA is that it provides complete set information about the application which we have to test, as mentioning the woven aspects, and their expected faults. We tag the faults information with aspects categories. In support to this process we can have RF module, which provides complete assistance while testing any aspect-oriented application. As the faults are always likely to occur, we can use a sub-module named as

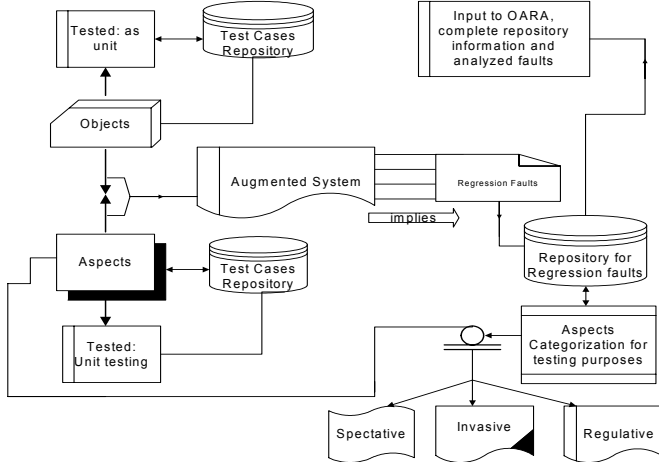


Fig.2 TSA. Application information aspect categorization, and regression faults

RF (Fig. 4) in OARA, to facilitate testing process. Through this module, faults repository is gradually built. By this we can prioritize, and categorize faults for testing aspect-oriented applications. The TA highlights the approach in a robust way.

3.4 Test Architecture (TA)

Test cases and test data's reusability is still a challenge in testing process. TA (Test Architecture Fig.5) is one of the attempts to demonstrate feasible and manageable solution for aspect-oriented applications. The TA addresses several issues, which are still unanswered for aspect-oriented programs.

Since the test cases have to be applied on augmented application. There is a need to perform serious effort for prioritization, selection, and generation of test cases. OARA has the capability to prioritize, select, and generation of test cases through TA. We have two issues in test cases, one is the re-use of existing ones, and the other is generation of new. Prioritization of test cases can be analyzed by the last two modules, i.e., TSA and RF, which look after for faults including the aspects categorization. On the basis of those faults, the relevant test cases are automatically selected, and would be executed.

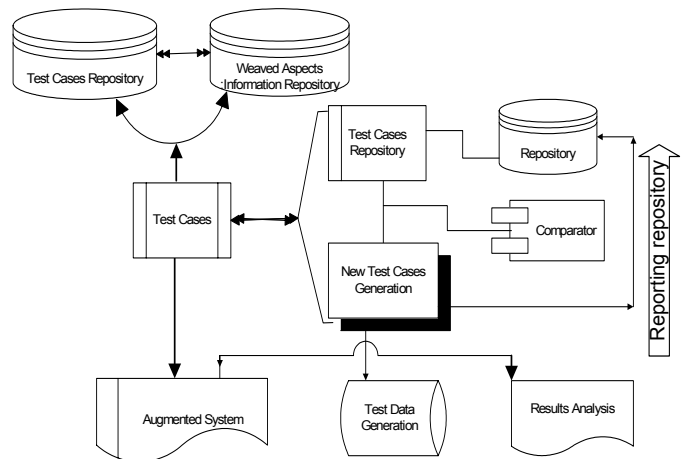


Fig. 3 A component Comparator for test-cases generation and reuse

In case, the faults' test cases doesn't exist in the repository (this is analyzed by the comparator component of TA), new test cases are generated to complete the whole testing process. After that we can prepare complete set of test cases to be exercised on the aspect-oriented program. For identification of faults, we can adopt Safe Techniques for test selection or any other which is meeting the criterion. Safe technique is one of the amortizing solutions as described in [10]. The following algorithm precisely describes the procedure:

```

do
While criterion not satisfied
Select test cases, which already used using TSA
or new test cases as per required by the faults
comparator organizing test cases generation, and reuse
generate test data
if test cases satisfies criterion
end;
else reduction or addition of new test cases
end;
    
```

IV. IMPLEMENTATION DETAILS

It is inadequate to test applications manually or even testing activity becomes unmanageable, and costly, which is infeasible to adopt. To the best of our knowledge OARA (Objects-Aspects Regression Analysis, Fig.7) is only a new methodology to test large aspect-oriented applications automatically (the algorithm is given below). In our testing process, we are exploiting the attractive attributes of regression testing to perform testing applications, and implicitly this information is used in our tool, named OARA. We are working to bring its implementation for challenging aspect-oriented paradigm. Intuitively numbers of standard features have been added for thorough testing of aspect-oriented programs, but still needs enhancements.

```

01: Get Information about the aspect-oriented
application
    // its classes information, woven
    aspects and their categories
    determined from TSA
02: Get the testing Criterion
03: LOAD TSA Information in CONSTRUCTOR
04: UPDATA (Delete or Add) RF Information
    //if already exist
05: SETUP TA Environment
06: Use Comparator to trace existing Test
Cases for Reuse
    //including Also search for existing
    test cases applied to aspects
07: If criterion/ testing environment doesn't
satisfy,
    create New Test Cases
    //Generate Test Data along with new Test
Cases
08: Tester's verification for starting
testing process
09: Complete Setup for testing environment
10: while (testing criterion not true)
10: TEST AOP()
11: for(i=0; i< Test_Cases; ++i)
12: if (Test(i))is true
13: save the result
14: else interpret the test case result
15: end for
16: end while
17: Print Results for Analysis
18: End of Testing Activity
    
```

OARA Algorithm

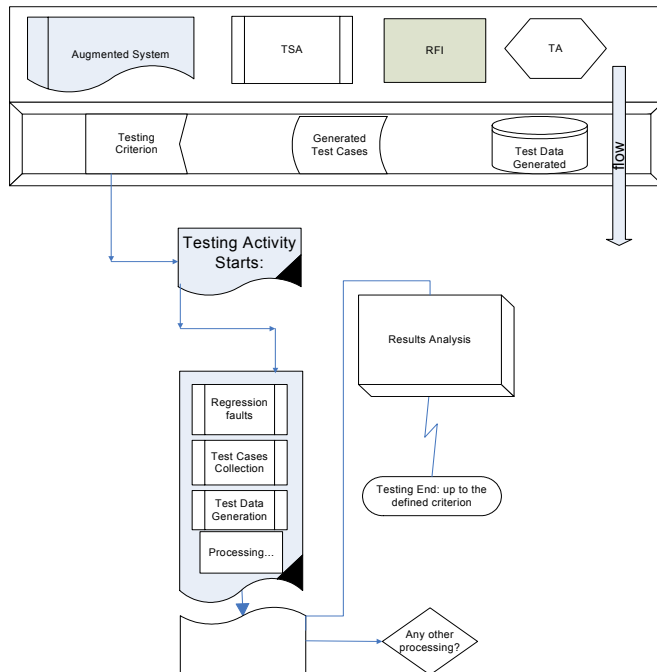


Fig.4 OARA support for ADRT

After first implementation of our tool, we gradually enhance its functionality, and to make it more useful and appropriate for aspects based faults in any system. Hence the complete process is being automated to meet the aspect-oriented challenges as well as to address the large applications.

V. CASE STUDY

Aspect-oriented applications are result of weaving process. A woven application, we say the new system is Δ , we want to test all the changes which aspects made in the core system, which was functional as standalone system. To test the Δ , we use OARA for reuse of existing test cases, generation of new test cases along with data, TSA provides information regarding woven application i.e., Δ . Result analysis is automated against the set criterion.

5.1 Description

The core concerns are implemented using any object-oriented language as aspect-oriented paradigm allows developers to do so. Specifically the crosscutting concerns are written using any aspect-oriented language e.g., AspectJ, HyperJ, AspectC++. After specifying pointcuts, join points, advices etc., in the aspects which are implementing crosscutting features, by compiler, the aspects are woven to core system. Number of interdependencies exists among objects and aspects. The overall system is called augmented system.

As the aspects make numerous changes to the existing system, and there remains no validity that the implemented core concerns function as desired. In our case the classes have been tested already, as single units. Aspects can also be individually tested, same as unit testing. We may assume that independently the core and crosscutting features are performing their desired functionalities. Weaving together the objects and aspects, requires validation of desired functionalities.

Several faults are introduced by weaving process of aspects with objects. Through ADRT, all faults are categorized, especially the types of aspects, which have been used as crosscutting features. TSA component of ADRT supports the testing activity as it keeps the comprehensive information regarding the built application. TA element is another major feature, which is managing the overall test cases activity. A comparator component is being employed to decide reusability of test cases. Regression faults are explicitly maintained in RF, which is result of any aspect-oriented application. Repository is maintained for RF, as faults in one aspect-oriented product may be seen in many other aspect-oriented applications.

There are two situations when the reusability of test cases is obvious. One part of the reusability factor is to execute the test cases which were used for testing of classes, and still they effectively testing those parts of the classes, which are affected by the aspects. In second situation, reusability of test cases is more certain because those test cases, which were generated earlier for any aspect-oriented application, and considering the faults in results of weaving process, those test cases can be executed more obviously and effectively. Reusability may also contain existing test data.

The major focus of ADRT framework is to analyze the augmented system by testing process, which either can contain reusability of existing test cases (as discussed above) or generation of new test cases along with test data, test oracle generation, test execution, and in the last coverage evaluation is

carried out.

The overall testing process is automated, to minimize the cost of testing; less man-machine interaction is focused. A tool, OARA is used to automate the testing process, which gathers all the information, which is adequate to start testing process. Testing criterion is a human activity as it can be effectively specified by the tester. But the coverage evaluation may be automated, which would be against criterion. The overall testing process gets ends according to criterion, if satisfied.

VI. CONCLUSION & OUTLOOK

Commendable research is in progress for testing of aspect-oriented software. Instead of fluff and crippled testing techniques, we should adopt productive testing solutions like regression testing. Significant attention is paid to the challenging issues, which are set by the new paradigm, but it is very important that fault proven, high in cost, difficult to adopt techniques should not be visited in the context of aspect-oriented testing. It should also be on the agenda to absorb the testing techniques for object-oriented systems and their pay-offs. We have presented a testing framework (ADRT) for aspect-oriented programs, which includes the analysis of aspects interaction with core system. In this framework, we have identified test cases, test data, reusability, automation aspects of the framework, and specifically addressing the faults, which occur during aspect-oriented programming. In our future work, we will present our fully implemented tool, OARA for ADRT framework, specifically considering the test cases generation, reduction in test cases, and different types of flaws in aspects woven applications. Regression testing has much impressive results if the process is automated. ADRT will be analyzed for feasibility, its adoptability for practical use, and to make it more feasible. Obviously experimental study will be presented to demonstrate the strength of ADRT. Currently we are working on an aspects based application, and also the proper tool configuration.

REFERENCES

- [1] Sereni, D., Moor, O. D.: Static Analysis of Aspects, Oxford. In AOSD 2003 Boston, MA USA. In proceedings of ACM (2003)
- [2] Sakurai, K., Masuhara, H.: Ubayashi N., Matsuura S., Komiya S.: Association Aspects. In AOSD (March 2004) also in proceedings of ACM (2004)
- [3] Denaro, G., Monga, M.: An Experience on Verification of Aspect Properties. In IWPE 2001, Austria, also in proceedings of ACM (2001)
- [4] Douence, R., Fradet, P., Südholt M.: Composition, Reuse and Interaction Analysis of Stateful Aspects, In AOSD 04 (March 2004), Lancaster UK. Also in proceedings of ACM (2004)
- [5] Storzer, M., Krinke, J., Breu S.: Trace Analysis for Aspect Application. Universität Breu, Passau, Germany
- [6] Ubayashi, N., Tamai, T.: Aspect-Oriented Programming with Model Checking. In Proceedings of ACM (2002)
- [7] Lopes, G.V., Ngo, T.C.: Unit-Testing Aspectual Behavior, Donald Bren School of Information and Computer Sciences, University of California, Irvine.
- [8] Katz, S.: A Survey of Verification and Static Analysis for Aspects. AOSD-Europe Technion-1(10 July (2005)
- [9] Zhou, Y.: Towards a Practical Approach to Test Aspect-Oriented Software, Department of Informatics, Donald Bren School of Information and Computer Sciences, University of California.
- [10] Graves, T.L., Harrold, M.J., Kim, J., Porter, A., Rothermel, G.: An Empirical Study of Regression Test Selection, In proceedings of IEEE (1998)
- [11] Xu, W., Xu, D., Goel, V., Nygard, K.: ASPECT FLOW GRAPH FOR TESTING ASPECT-ORIENTED PROGRAMS, Department of Computer Science, North Dakota State University Fargo, ND 58105. U.S.A.
- [12] Zhao, J.: Data-Flow-Based Unit Testing of Aspect-Oriented. Proceedings of the 27th Annual International Computer Software and Applications Conference (COMPSAC'03), In proceedings of IEEE (2003)
- [13] Rajan, H., Sullivan, K.: Generalizing AOP for Aspect-Oriented Testing, in conference '05, month 1-2, 2005, 2004 ACM 1-58113-000-0/00/0004.
- [14] Zhao, J., Rinard, M.: System Dependence Graph Construction for Aspect-Oriented Programs, Cambridge, USA
- [15] Hailpern, B., Santhanam, P.: Software debugging, testing, and verification, IBM SYSTEMS JOURNAL, VOL 41, NO 1, 2002.
- [16] Mortensen, M., Alexander, R.T.: Adequate Testing of Aspect-Oriented Programs, Colorado State University, Fort Collins, Colorado, USA, Technical report CS 04-110, December 2004.
- [17] Xu, G., Yang, Z., Huang H., Chen, O., Chen, L., Xu, F.: JAOUT: Automated Generation of Aspect-Oriented Unit Test Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04), Also in proceedings of IEEE
- [18] Hughes, D., Greenwood, P.: Aspect Testing Framework, Computing Department, Lancaster University, UK
- [19] Augusto, O., Lemos, L., Maldonado, J.C., Masiero, P.C.: Data Flow Integration Testing Criteria for Aspect-Oriented Programs, Universidade de São Paulo, Av. do Trabalhador São-Carlense, 400, São Carlos, SP.
- [20] Alexander, R.T., Bieman, J.M.: Towards the Systematic Testing of Aspect-Oriented Programs, Colorado State University, Department of Computer Science, Safety Systems Research Center, University of Bristol, Bristol, UK, 2004 Published by Elsevier Science B. V.
- [21] Xiaoguang, M., May, J.: A Framework of Integration Testing using AspectJ, 2004 published by Elsevier Science B. V.
- [22] Bruel, J.M.: Using Aspects to Develop Built-In Tests for Components, Submitted in July 03 to the AO modeling with UML Workshop at UML'03, San Francisco, USA.
- [23] Rapps, S., Weyker, E.J., Data Flow Analysis Techniques for Test Data Selection, In 1982 IEEE, 0270/82/0000/0272.
- [24] Zhao, J.: Slicing Aspect-Oriented Software, Proceedings of the 10th International Workshop on Program Comprehension (IWPC'02). IEEE (2002)
- [25] STRUCTURING OPERATING SYSTEM ASPECTS, COMMUNICATIONS OF THE ACM, October 2001/vol. 44, No. 10
- [26] Walker, R.J., Baniassad, E.L.A., Murphy G.C., An Initial Assessment of Aspect-oriented Programming, Dept. of Computer Science, University of British Columbia, 20 1-2366 Main Mall, Vancouver, BC V6T 124 Canada, In ICSE '99 Los Angeles CA USA, In ACM 1999, 1-58113-074-0/99/05, IEEE (1999)
- [27] AspectC++ Homepage: <http://.aspectc.org>.
- [28] AspectJ Homepage: <http://eclipse.org/aspectj>.
- [29] Xu, D., Xu, W., Nygard, K.: A State-Based Approach to Testing Aspect-Oriented Programs, Technical Report NDSU-CS-TR04-XU03, North Dakota State University, September, 2004.
- [30] Bach, J.: Reliable Software Technologies, A Framework for Good Enough Testing, In proceedings of IEEE Computer Society (1998)
- [31] Harrold, M.J.: Testing: A Roadmap, In Future of Software Engineering, 22nd International Conference on Software Engineering, June 2000.
- [32] Okun, V., Black, P.E.: Issues in Software Testing with Model Checkers, National Institute of Standards and Technology, Gaithersburg, 2003.
- [33] Rinard, R., Salcianu, A., Bugrara S.: A Classification System and Analysis for Aspect Oriented Programs, In proceedings of ACM (2004)
- [34] Binder, R.V.: Testing Object-Oriented Systems, Models, Patterns, and Tools. Printed by Addison-Wesley (1999)
- [35] Xu, D.: Test Generation from Aspect-Oriented State Models, Technical Report NDSU- CS-TR-05-XU02, (September 2005)