A Survey of UML Based Regression Testing

Muhammad Fahad and Aamer Nadeem

Mohammad Ali Jinnah University Islamabad, Pakistan.

mhd.fahad@gmail.com, a.n@acm.org

Abstract: Regression testing is the process of ensuring software quality by analyzing whether changed parts behave as intended, and unchanged parts are not affected by the modifications. Since it is a costly process, a lot of techniques are proposed in the research literature that suggest testers how to build regression test suite from existing test suite with minimum cost. In this paper, we discuss the advantages and drawbacks of using UML diagrams for regression testing and analyze that UML model helps in identifying changes for regression test selection effectively. We survey the existing UML based regression testing techniques and provide an analysis matrix to give a quick insight into prominent features of the literature work. We discuss the open research issues like managing and reducing the size of regression test suite, prioritization of the test cases that would be helpful during strict schedule and resources that remain to be addressed for UML based regression testing.

1 Introduction

The purpose of regression testing is to selectively retest the software after certain modifications to ensure that they have not caused unintended effects on unchanged parts and changed parts of the software behave as intended [1]. Therefore, regression testing process focuses on identification of changes so that those unchanged parts that are already tested should not be tested again to reduce cost, and only changed parts corresponding to those changes should be tested. The objectives of regression testing include not only selective retesting of the software to check its conformance to the new specification, but also enhancing the confidence of the clients that the software product can be changed according to their requirements and the environment [2]. Through the effective regression testing, the programmer also comes to know about the implications and side effects of the changes that have been made. Reusing previous test cases not only reduces the cost of newer test case generation but also reduces other costs of creating test case execution set-up, building oracle and crafting data that can be used [2].

Software has to go through a repetitive process of refinement during its development lifecycle. Software engineers have to pay much attention to produce high quality bug free software and it may require many testing techniques at various levels. Regression testing can be applied at any level of testing i.e. unit testing, integration testing, and system level testing. It is different from development testing as in regression testing an existing test suite is available for reuse [3]. It is the most costly process in software lifecycle and according to a study, about 80% of testing budget and one-third of the total cost of software is spent on regression testing and maintenance of the product [4]. Many techniques exist in the literature for maintenance and regression testing of software. Most of the work has been done on code based regression testing in which test suite is built about the delta change between the original code and the changed code, and a survey on code based regression test selection techniques is provided by Rothermel and Harrold [5]. Very few techniques use specification or UML design for change impact analysis to revalidate the software. The main effort is to reduce the cost of testing by selecting cost minimized subset of test cases for regression suite maintenance because rerunning all test cases would be time-consuming and would result in huge cost [2]. Besides cost, a trade-off between the selection and execution of test cases and the fault detection ability of the test cases that are executed is paid great attention during regression test selection. Cost-effectiveness of testing techniques depends upon many factors. Rothermel et al. identify the effect of grouping of test inputs into test cases on the cost-effectiveness of regression testing techniques [6]. Some researchers provide test case prioritization techniques that help when to test an artifact with limited budget and strict schedule [7]. Test case prioritization is also important for UML based regression testing techniques but none of the existing UML regression techniques incorporate this feature.

In this paper, we survey the UML based regression testing techniques. Although there is not much work in the literature that uses UML design for regression testing but this has certain advantages over code based regression testing. We highlight these advantages and significance of using *UML Design* versus *Code* for Regression Testing. While working with UML based regression testing techniques, we observed that UML with OCL constraints can be modeled for Regression Testing of Component Based Systems for systematic regression testing.

Rest of the paper is organized as follows: Section 2 discusses the advantages and significance of using UML design rather than code for regressing testing. Section 3 discusses categories of regression test selection techniques based on certain criteria. Section 4 comprises of survey on UML based regression testing techniques with their salient features. Section 5 discusses our analysis on existing techniques, and provides an analysis matrix. Section 6 concludes the paper.

2 UML Design versus Code based Regression Testing

UML design based regression testing techniques have many advantages over code based regression testing techniques, as outlined below:

- *Traceability:* Identification of change is easily traceable from design rather than the code [8]. Finding delta change in modified and original code is much difficult, and is protracted without code change history that is often ignored by developer during the implementation.
- *Scalability:* Code based regression testing is done only on a small scale, i.e., at unit level [9]. When applied to test large components, scalability becomes the main hindrance to manage all the information and to create corresponding traceability matrices. UML design based regression testing techniques are practical at all levels of testing and of large software applications as well.
- *Understandability:* Tester has to understand the code programmed by others which is a tedious and time-consuming task [9]. UML design is easier to understand and gives quick insights about the requirements and specification.
- *Language dependence:* Code based regression testing process is language dependent. Software that is built on different languages needs many code based regression-testing techniques [9], which increases complexity of the whole process. Regression testing by means of UML designs is free from this limitation as they are based on the standard UML notations [8].
- *Cost:* Code based regression testing detects faults at later stages of software lifecycle and thus consumes huge amount of cost in correcting them. But regression testing at design time gives early detection of faults and reduces the overall cost to apply correction procedures earlier during design phase [10].
- *Code Dependence:* Code based regression testing techniques are only applied when source code is available and hence they are not practical for component based software engineering. Component based systems are built-up by reusing existing components whose implementation is not available [11]. The only thing that component users have is interface specification and modified data information. Thus UML design based regression testing is effectively used for maintenance and correction purposes without the dependency of code.
- *Complexity:* UML designs provide an easy retrieval of relevant static and dynamic information from its various static and dynamic diagrams [8]. This task would be much difficult while extracting information about dynamic bindings between methods from code.
- *Executable UML:* UML based regression testing techniques are also effectively used for validation of executable forms of UML such as Executable UML and the UML virtual machine.

UML based regression testing techniques have some drawbacks [8] too:

- *Invisible Changes:* There are certain changes that may not be visible in design and need special ways to document them, e.g. a change in a method’s body.
- *Consistent and up-to-date Design:* UML design based regression testing techniques assume that the diagrams used are consistent with each other. Change can only be detected if this assumption holds; violating this assumption makes the technique awkward and generates poor performance. Furthermore, they require design to be complete and up-to-date.

- *Low Precision:* UML design based regression testing techniques do not precisely build test suite as compared to techniques that utilize detailed code analysis. Precision of a technique means that testing strategy only selects required test cases from existing tests to build regression test suite, i.e., obsolete test cases are detected and ignored.

3 Regression Test Selection Techniques

To achieve successful regression testing, Hsia et al. identify four phases to ensure that the system behaves as intended after changes have been made [12]. First, the process starts by identification of changes made in certain parts, because we have to analyze that software has not been adversely affected by the modifications. Second, we have to build regression test suite by identifying three types of testcases from original testcases, i.e., i) test cases which are no more valid due to the changes made, as invalid test cases are no more useful, ii) test cases which are still valid but not useful as they are already tested and iii) testcases which should be retested to ensure correct software behavior with newer changes. Third, a cost effective testing strategy is made. Finally, selection of cost-minimized subset of test suite to retest the system after changes has been made.

Graves et al. [4] categorize the Regression test selection techniques as:

- *Minimization Techniques:* These techniques focus on selectively retest the software with minimum testcases covering modified or affected portions.
- *Dataflow Techniques:* These techniques select those test cases, which execute data interactions that have adverse effect by changes made.
- *Safe Techniques:* These techniques are designed to reveal the same faults as a retest-all strategy reveals. Thus those test cases that exercise the suspected portion having faults are more focused because they can reveal other most likely faults and exercise the critical functionality.
- *Ad-Hoc /Random Techniques:* These techniques build regression test suite by choosing randomly test cases from original test suite. Randomly rerunning test cases do not address the coverage of affected portions and may not find the most severe faults.
- *Retest-All Techniques:* These techniques rerun the entire original test suite to ensure that modifications have not regress the software functionality, but this requires enough time or resources to rerun the entire test suite.

4 Survey on UML Based Regression Testing Techniques

A variety of regression testing techniques have been described in the research literature. This section throws the light on their summarized features.

- **Specification-based Regression Test Selection with Risk Analysis.** Chen et al. [9] use activity diagram that describes the requirements, behaviors and workflows of underlying system to test. For regression testing, they select two types of tests, i.e., Targeted Tests and Safety Tests. Targeted Tests focus on those features that are still valid in newer version. Safety tests are built to test the modification parts. Chen et al. have uses the Amland's [13] proposed risk model, and emphasis on the cost minimization by detecting the most critical defects first. For regression testing, they apply CFG-based algorithm to activity diagram for detection of affected entities. Then, they form *Targeted Test* which executes the affected edges for regression analysis. For safety tests, they calculate the *Risk Exposure* for each test case. Safety Tests are chosen from the tests that have the highest value of risk exposure. The cost estimation and risk exposure calculations would be more attractive when time and cost is short.
- **Automating Impact Analysis and Regression Test Selection Based on UML Designs.** Briand et al. [8] use consistent sequence diagram, class diagram and use case diagram for identification of changes made to generate regression test suite. For regression testing, they detect changes by comparing previous and new version of Sequence diagrams and Class diagrams. Changes in sequence diagrams are obtained by viewing messages with different conditions, due to change in triggered messages and deleted sequence of boundary messages. Detected changes refer to changes in actions i.e. changed operations and changed classes. Then, they compare two versions of Class diagram to detect the set of changed attributes, operations, relationships and classes. They emphasize on OCL expression analysis of both versions in order to detect changes in operation's contract or in messages. On basis of identification of changes obsolete, retestable and reusable testcases are chosen for regression analysis. They evaluated their work on three industrial case studies and showed effectiveness of their work. Their case studies showed that the number of reusable test cases represented a large proportion (up to 100%). Moreover, they gave evidence about automation of their work by providing Regression Test Selection tool (RTSTool).
- **Maintaining Evolving Component-Based Software with UML.** This UML-based technique was proposed by Wu et al. [14] for component-based software systems that are particularly built on reusable components. Component-based systems need three types of maintenance i.e. Corrective, perfective and adaptive maintenance. In this paper, author gave a regression testing strategy for corrective maintenance as it involves modification on individual classes in a component, leaving none effect on the structure of component as a whole. They use collaboration diagram and statechart diagram to identify changes. For each change in collaboration diagram, test cases are selected which traverse such modified or changed parts. Furthermore, they analyze impacts of change on control sequences and on data dependencies separately to build regression test suite. For identification of change on control sequences, they suggest to retest the modified artifacts in collaboration diagram and all possible affected scenar-

ios that are represented in the statechart diagram. For identification of change on data dependencies, they suggest to retest all the dependent interfaces as well.

- **Efficient Object-Oriented Integration and Regression Testing.** Traon et al. [15] propose a strategy for integration and regression testing from an object oriented model. They produced a model of structural system, Test Dependency Graph (TDG) mapped from the class diagram that evolves with the refinement process of the OO design. Vertices of this graph represent the component and directed edges represent dependencies between classes or methods. Once the TDG is constructed, integration and regression testing strategies are applied on decomposition of the TDG. To build regression test suite, dependencies of both versions of TDG are compared for identification of changes. When the edges are found to be modified that represent dependencies between vertices (components), test cases are build up to cover all the dependant vertices and edges. They formulate two coverage criteria's for testing a component C in a system. Weakest criteria suggest that only those components are tested which are directly dependent from C. But the Strongest coverage criteria suggest testing each component that is included into a path containing C.
- **Model-based Testing and Maintenance.** Deng et al. [16] propose a Semantic Software Development Model (SSDM) for object oriented software and model-based regression test selection for software testing and maintenance. This model is more complete as it incorporates all the phases of the software development process: requirements, design, implementation, testing and maintenance. Information captured by the testing objects and maintenance objects are utilized in order to select regression test suite from original test cases. First, they define the tight-coupled relationships between UML diagrams for efficient and flexible testing and maintenance. For test selection they suggest that when a particular operation is modified, find all the operations that are dependant on this operation, and all the dynamic UML diagrams that include the corresponding behaviors for that operation. Then find all the use cases that are described by the found dynamic UML diagrams. For regression testing, test all the use cases whose corresponding operations need to call modified operation.
- **Regression Testing UML Designs.** Pilskalns et al. [10] propose a safe and efficient regression testing technique based on test cases for UML designs, where test cases always map to sequence diagram scenarios. They use the knowledge of existing approaches to build their regression testing approach, i.e., as a general framework[17], to identify changes[8], to classify test cases[18]. But unlike others, his work was the initial work that was done on identifying change impact for UML test cases rather than code test cases and map changes between UML model and UML test cases. For the purpose of testing, first they made an integrated model named Object Method Directed Acyclic Graph (OMDAG) from Class Diagrams, Sequence Diagrams and OCL. When the OMDAG integrated model is created, test cases are generated which are sets of inputs by using a non-binary analysis technique to partition values that can be

assigned to variables in conditional nodes. When the test case is executed it traverses a path in the OMDAG. And when a path changes, it affects one or more test cases associated with the path. They classify changes into three sets i.e. NEWSET, MODSET, and DELSET, according to whether they create, modify or delete elements in the design. They use delta function to find the test cases affected by a design change, which compares vertices and edges affected by the change made to the paths associated with a test case. Only Pilskalns et al. claimed by experimentation that their strategy selects the test cases with run-time less than as compared to retest-all technique.

- **Integrating White- and Black-Box Techniques for Class-Level Regression Testing.** Beydeda et al. [1] first propose a Class-Level testing of object-oriented prototypes by integrating two existing white box [17] and black box [19] techniques. Rothermel's idea [17] of white box testing is based on traversing both versions of a class, represented by class control flow graphs (CCFGs) to detect and analyze changes. Hong's idea [19] is based on identifying def-use pair of each attribute from class flow graph (CFG) and test suite is built by covering these def-use pairs. For regression testing, Beydeda et al. used a CFG and CCFGs to construct an integrated model called class specification implementation graph (CSIG) [20]. They built regression test suite by the algorithm which takes two versions of CSIG and previous refined test suite. For analyzing safe regression, previous refined test suite is obtained by manually deleting obsolete test cases from original test suite. First test cases are generated by white box testing criteria in which both graphs are traversed to analyze changes in the statements against the nodes. Once changes are identified, test cases covering those changes are generated. Then the algorithm generates test cases from black box criteria by testing inter-method data flow for def-use pairs.
- **An Approach for Selective State Machine based Regression Testing.** Farooq et al. propose an approach for selective state machine based regression testing [21]. For change identification, they use Behavioral state machine (UML 2.1) and class diagram, and classify the changes as class-driven changes and state-driven changes. For building the regression suite, they adopt Briand's test suite classification mechanism, i.e., Obsolete, Reusable, and Retestable. First, they generate class-driven changes by comparing original class diagram, and modified class diagram. The identified class-driven changes are propagated to state machine comparator that identifies state-driven changes that are passed as input to the regression test selector that separates the Obsolete, Reusable and Retestable test cases. The validity of the approach is tested on a small case study.
- **UML Based Regression Testing for OO Software.** Mansour and Takkoush [22] propose a UML based regression testing for object-oriented software by using the interaction overview diagram, class diagram and sequence diagram. Their strategy works by assuming that the test suite contains tests for unit level testing as well as system level testing, and works in phases by selecting tests for each level. First, they identify changes from class diagram. Then, they iden-

tify unit and system level tests from interaction overview diagram that are directly affected by the changes detected in the first phase either by traversing or dependency analysis. If a change is identified in sequence diagram, their algorithm suggests selecting the test cases that execute changed methods. They provided the empirical results of their experiment on nine subject applications and showed that their strategy identified all the tests similar to the retest-all strategy. Their experiment also showed the good precision results by ignoring non-modification test case.

Table 1. Comparison of UML based Regression testing Techniques.

Parameter/Reference	[9]	[8]	[16]	[10]	[20]	[14]	[15]	[21]	[22]
UML Notation*	AD	CD, SD, OCL	All	CD, SD, OCL	CSM	COD, SCD	CD	CD, BSM	IOD,CD, SD
Risk Based	Yes	No	No	No	No	No	No	No	No
Transformation needed	No	No	Yes	Yes	Yes	No	Yes	No	No
Test case classification*	Safety, targeted	ORR	No	ORR	No	No	No	ORR	No
Cost Analysis	Yes	No	No	No	No	No	No	No	No
Change impact*	CT	CT	CT	UMLT	CT	CT	CT	UMLT	IMLT
Safety	High	High	No	High	Low	Low	Low	High	High
Tool Support	No	Yes,	No	No	No	No	No	No	No
Case Study Evidence	Yes	Yes	No	No	No	No	Yes	Yes	Yes
Feasibility	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
Generality	Yes	Yes	No	No	No	No	No	Yes	Yes
Precision	High	High	No	High	No	Low	Low	High	High
Inter. Model name	No	No	SSDM	OMDAG	CSIG	No	TDG	No	No

*Notations used in comparison matrix

CD: Class Diagram, **IOD**: Interaction Overview Diagram, **SD**: Sequence Diagram, **AD**: Activity Diagram, **SCD**: State Chart Diagram, **COD**: Collaboration Diagram, **BSM**: Behavioral State Machine, **OCL**: Object Constraint Language, **CT**: Code testcases, **UMLT**: UML testcases, **ORR**: Obsolete, Reusable and Retestable classification

5 Analysis

This section narrates the identified analysis parameters to compare the efficiency and effectiveness of existing regression techniques. On basis of these parameters, analysis matrix is created to give quick insights on each of the approaches explained above as shown in Table 1. The parameters are:

- **UML Artifact:** Which UML diagram is used for change identification for building regression test suite.
- **Risk-Based:** Whether a technique builds risk matrices to quantitatively measure the safety of a test suite. By safety we mean whether a technique has the ability to reveal a fault in the modified program and build test cases that exercise the suspected portion having faults. While analyzing UML based regression-testing techniques, we found only one technique by Chen et al. that uses the risk model and calculates the safety of a test suite.
- **Transformation Needed:** Whether the technique is capable of identifying changes from the diagrams directly or builds the intermediate model that is efficient and conveys easy interpretation while identification of change impact analysis. [8,9,14,21,22] have not built any intermediate model, while Pilskaln's OMDAG is easy to understand because nodes of the model are similar to the classes in Class Diagram, and edges represent sequences between classes. But others form very complex intermediate model.
- **Test Case Categorization:** Whether a technique divides the original test suite to build regression test selection. [9] builds safety and targeted tests, and [8,10,21] identify Obsolete, Reusable and Retestable regression test cases.
- **Cost-Analysis:** Whether a technique calculates costs of each test case, and involves cost efficient strategy to select build regression test suite. Only Chen et al. involve the cost aspects in their strategy.
- **Change impact on Code Testcases/UML Testcases:** Some techniques identify changes that impact code test cases rather than UML test cases. [10,21,22] focused on identifying changes that affect UML test cases and their classification upon mapping changes between a UML design and UML test cases.
- **Safety:** Whether a strategy selects the test cases that reveal the same faults and helps in exposing errors caused by changes as a retest-all strategy reveals. Only works from Pilskalns et al. and Mansour et al. is very safe in this regard.
- **Feasibility:** Whether the testing criterion is feasible in a sense of identifying the impact of changes in the artifact, and is cost effective to be used for particular scenarios during software lifecycle.
- **Generality:** Whether the technique can be extended and applied to a wide and practical range of situations.
- **Precision:** Whether the strategy detects obsolete test cases and ignores them effectively, and change impact analysis only selects those test cases that are really beneficial to ensure the revalidation of software.

- **Tool Support:** Whether proposed technique is tool supported. Only Briand et al. provided the tool named “*RTSTool”* along the technique.
- **Case Study Evidence:** Whether the authors have made some experiment or case study to give evidence of their testing strategy. [8,9,15,21] build a case study evidence while analyzing their testing techniques to promote understanding.

6 Conclusion and Future directions

Regression testing, as a means of quality control measure, is one of the most costly testing techniques to ensure that modifications have not affected the working correct behavior of system and newly created modifications behave as intended. This paper surveys the regression testing techniques based on UML designs. We analyze that UML based regression testing opens a number of advantages and is practical for small and large applications. Classification of regression test suite into Obsolete, Retestable and Reusable test cases is highly significant and most of the literature techniques employed the same classification. UML models with OCL expressions can be effectively used for regression testing of component based systems. Safe techniques that identify the same test cases as the retest-all strategy identifies, are good for small scale test suites and small applications. However, safety for large applications and test suites is difficult to achieve as prioritization is needed for the selection of cost minimized subset for retesting. Identification of changes that affect on UML test cases and Code test cases are different and needs special attention. Little research has been done on identifying changes that impact UML test cases and classify test suite based on mapping changes between UML design and UML test cases, unlike others consider the behavior of the code. One of the future directions on this topic is to perform more work on classifying test cases based on UML designs. Another direction could be to analyze different aspects of cost, test suite minimization, testing of UML executable models, systematic revalidation of UML models and test case prioritization for UML, as these are important during regression testing in a controlled environment.

References

[1] S. Beydeda, and V. Gruhn, Integrating white- and black-box techniques for class-level testing object-oriented prototypes. In Software Engineering and Applications Conference, Las Vegas, Nevada, pp. 23–28, 2000.

[2] H. K. N. Leung, and L. J. White, A Cost Model to Compare Regression Test Strategies. Proc. Conference on Software Maintenance, Italy, pp. 201-208, October 15-17, 1991.

[3] Y. Chen, R. L. Probert, D.P. Sims, Specification based Regression test selection with risk analysis, IBM Center for Advanced Studies Conference. Proceeding of the Conference of the center for advance studies on collaborative research, 2002.

[4] T. L. Graves, M.J. Harrold, J. Kim, A. Porter, and G. Rothermel, An Empirical Study of Regression Test Selection Techniques, ACM Transactions on Software Engineering and Methodology, Vol. 10 (2001), No. 2, pp 184-208.

[5] G. Rothermel, and M.J. Harrold, Analyzing Regression Test Selection Techniques. IEEE Transactions on Software Engineering, Vol. 22 (1996), No.8, pp. 529-551.

[6] G. Rothermel, S. Elbaum, A. Malishevsky, P. Kallakuri, B. Davia, The Impact of Test Suite Granularity on the Cost Effectiveness of Regression Testing. Proceedings of the 24th International Conference on Software Engineering Orlando, Florida, pp.130-140, 2002.

[7] G. Rothermel, R.H. Untch, Chu. Chengyun, M.J. Harrold, Prioritizing test cases for regression testing. Transactions on Software Engineering, Vol. 27 (2001), No.10, pp. 929 – 948.

[8] L.C. Briand, Y. Labiche, G. Soccar, Automating Impact Analysis & Regression Test Selection Based on UML Designs, Proc. of Intl. Conference on Software Maintenance, IEEE,2002.

[9] Y. Chen, R.L. Probert, D.P. Sims, Specification based Regression test selection with risk analysis, Proc. of the center for advance studies on collaborative research, 2002.

[10] O. Pilskalns, G. Uyan, A. Andrews, Regressin Testing UML Designs, 22nd IEEE international Conference on software maintenance (ICSM'2006).

[11] S.A.M. Sajeev, and B. Wibowo, UML modeling for regression testing of component based systems. Published by Elsevier Science, B.V., 2003.

[12] P. Hsia, X. Li, D.C. Kung, C. Hsu, L. Li, Toyoshima, A technique for the selective revalidation of OO software, software maintenance: research and practice, Vol. 9(1997), pp. 217-233

[13] S. Amland, Risk Based Testing and Metrics: Risk analysis fundamentals and metrics for software testing including a financial application case study, The Journal of Systems and Software, Vol. 53(2000), pp. 287-295.

[14] Y. Wu, J. Offut, Maintaining Evolving Component-based Software with UML, Proc. of 7th European Conference on Software Maintenance and Reengineering (CSMR'03), 2003, IEEE.

[15] Y.L. Traon, T. Jeron, J. Jezequel, and P. Morel, Efficient Object-Oriented Integration and Regression Testing, IEEE Transactions on Reliability, Vol. 49 (2000), No. 1.

[16] D. Deng, P. C.Y. Sheu, Model-based Testing and Maintenance, Proceedings of International Symposium on Multimedia Software Engineering (ISMSE'04), IEEE, 2004.

[17] G. Rothermel, M.J. Harrold, and J. Dedhia, Regression test selection for C++ software. Software Testing, Verification & Reliability, Vol. 10(2000), No. 2, pp. 77–109.

[18] H.K.N. Leung, and L. White, Insights into Regression Testing. Proc. IEEE Intl. Conference on Software Maintenance (ICSM), Los Almitos, pp. 60-69, October 16-19, 1989.

[19] H.S. Hong, Y.R. Kwon, and S.D. Cha, Testing of object oriented programs based on finite state machines. In Proc. of the 2nd Asia-Pacific Software Engineering Conference, Brisbane, Australia, pp. 234–241, 1995.

[20] S. Beydeda, and V. Gruhn, Integrating White- and Black-Box techniques for Class Level Regression Testing. IEEE computer society, 2001.

[21] Q. Farooq, M.Z.Z. Iqbal, Z.I. Malik, A. Nadeem, An approach for selective state machine based regression testing, Proceeding of AMOST '07, July 2007, UK, pp. 44-52, ACM.

[22] N. Mansour, and H. Takkoush, UML based regression testing for OO software, Proc. of 11th IASTED conference software engineering and applications. Nov, Cambridge, USA.