Regression Test-Selection Technique Using Component Model Based Modification: Code to Test Traceability

Ahmad A. Saifan
Department of Computer Information Systems
Yarmouk University
Irbid, Jordan

Iyad Alazzam
Department of Computer Information Systems
Yarmouk University
Irbid, Jordan

Mohammed Akour
Department of Computer Information Systems
Yarmouk University
Irbid, Jordan

Feras Hanandeh
The Hashemite University
Zarqa, Jordan

***Abstract*—Regression testing is a safeguarding procedure to validate and verify adapted software, and guarantee that no errors have emerged. However, regression testing is very costly when testers need to re-execute all the test cases against the modified software. This paper proposes a new approach in regression test selection domain. The approach is based on meta-models (test models and structured models) to decrease the number of test cases to be used in the regression testing process. The approach has been evaluated using three Java applications. To measure the effectiveness of the proposed approach, we compare the results using the re-test to all approaches. The results have shown that our approach reduces the size of test suite without negative impact on the effectiveness of the fault detection.**

Keywords—Regression Test; Regression Test selection technique; Meta-Model; Models Traceability

I. INTRODUCTION

The importance of software testing is increasingly driven by an extensive dependability on software systems. Software testing is one of the main techniques to enhance and increase the quality of software. Regression testing is a type of software testing that has a clear impact on the quality of software systems that evolve extremely over time in order to meet the needs for new requirements.

Regression testing is one of the methods used in increasing the quality of software. Regression testing is mostly used when new changes are made on the software and it aims to ensure that the introduced changes do not incur errors and change the intended behavior of the software. However, the cost of regression testing is very high since the tester needs to rerun all test cases of the previous test suites. Regression Test Selection (RTS) is an approach used in reducing the number of test cases to run on the modified software.

The main objective of regression testing is to uncover errors in the software after a new modification has been made. Moreover, it is to ensure that the new changes have not introduced more errors in the software. A quite good amount of research has been conducted in the area of regression testing including traceability of regression [1], test automation, test environment [2], reduction of the code size [3] [4] [5] [6] where several regression testing techniques and tools are used and compared. Regression testing uses the previous test suites to find if the new modification caused errors or not, as such, it would be very expensive to run all the test cases. Regression test selection is used to minimize the cost of regression testing by selecting sub-set of test cases in each test suite in the testing process. Many regression test selection techniques have been proposed [7] [8] [9] [10] [11] [12] [13].

Regression test selection techniques are directed to address the problem of reducing the regression test after software system modification [17, 18, 19, 20]. To the best of our knowledge, no work investigates the test suite reduction based on the specific reduction in software structure. Except for the research on change propagation [8], they provided a model-driven approach that maps structural adaptations in autonomic software, to update for its runtime test model.

The main contributions of our work are summarized as follows: we introduce an approach that employs Meta models in test case reduction; it is based on creating several models that are associated with different targets. We created two models that represent the test and component structure models of the software systems under study. We design and build the meta-models using Eclipse Modeling Framework [14]. Our approach synchronizes test models with their corresponding structure model. When any changes takes place in the component structure model of the system under test (reductive modification), component meta model will specify and transmit changes that should be taken to update the test model. In this work, we address the modification of software system when an existed component is removed. After removing test cases that belong to particular component, they were updated/ deleted according to the role of the targeted components. We performed several reductive modifications of three Java software systems that are placed in different domains. To measure the effectiveness of our proposed approach, automatic

inter-class mutants were inserted into the source code by MuJava tool [15]. MuJava tool is widely used to perform mutation analysis [16].

II. BACKGROUND

This section discusses the related work on regression testing, regression testing techniques, model synchronizations and naming convention techniques:

A. Regression Testing

Let *S* be a system or software, Let *S'* be a modified version of *S* and TC be a test case for S. The standard regression testing process is described as following:

Select *TC'* subset if *TC*, a set of test cases to execute on *S'*

Test S' with TC', ascertaining the correctness of S' through TC'

Construct TC'', a set of additional test cases for S'

Test S' with TC'', ascertaining the correctness of S'' through TC''

Construct TC''', a new test case for S' from TC, TC', and TC''.

B. Regression Test Selection Techniques

Regression test selection is the process of choosing a subset of suitable test cases from an original set of test cases to test and ensure that the changes introduced do not reveal errors. Regression test selection process involves two main steps: (1) discovering and highlighting the modified segments of the system , (2) test case selection which means selecting a subset of test case from the original set of test cases that can successfully test the unchanged segments of the software [17].

Many researchers have proposed approaches on techniques of regression test selection [18] [19] [20]: Following are some of the approaches in the literature:

- **Minimization technique:** this technique reduces the number of test cases through selecting a minimum set of test cases with the intention of getting coverage of changed or altered segments of the software. This technique depends on the finding and expressing the relations between basic blocks, test cases, and selecting set of test cases that ensure that each modified basic block is covered by at least one test case [21].
- **Dataflow techniques:** this technique uses the definition-use pair in reducing the number of test cases through selecting test cases that cover each changed definition-use pair.
- **Safe technique:** this technique differs from the above techniques in that the selected test cases have the ability to reveal errors in the modified and updated system. One technique in safe regression test selection is using control flow graph to represent the system under test;
- **Ad hoc (random technique):** this technique is used when the development team does not have enough time to execute all test cases and when the too is not available for test selection. Testers select number of test cases arbitrarily.
- **Retest all:** this technique uses all test cases and runs them against the modified software without excluding any of them which is very expensive computationally especially when there is a huge amount of test cases.

C. Model Synchronization and Naming Convention Strategy

Model Synchronization is the process of confirming the correspondence between two models as soon as one model is changed. Originally, the model synchronization is offered in the Model Driven Architecture (MDA) in order to obtain instrument for obtaining uniformity and modification traceability [22] [23]. Two approaches are used in model synchronization: explicit and implicit. In the implicit approach the relative among models are enclosed in higher order expressions. In the explicit approach the dependence relatives among models are enclosed and encoded directly.

Name convention is the process of concluding beneficial information from a set of harmony data. The strategy of naming convention is used to control the traceability correlation in the functional requirements that structure of the system which helps the engine of modification propagation to search for certain test according to the requirements [24] [25].

We utilize naming convention strategy towards managing the relationship of several components in structure model and their associated tests in the test model. Naming convention strategy allows automatic search for certain related-test items in the test model. In order to deal with the consistency between included models (i.e., models of test and component) our synchronization approach is founded on traceability relations among the interconnected models. Conventions involved use unique and distinctive identifiers for entire test cases and components, and using again component IDs inside test IDs for traceability.

III. METHODOLOGY

In this section, we describe our approach to regression test selection (i.e. reduction) using change traceability of software structure to its test model that occurs during software maintenance.

Figure 1 summarizes the main steps of our approach. The shaded boxes represent the major steps, and ovals represent inputs and outputs associated with each step. The approach consists of five main steps: dependency extraction, creation of test and software structure dynamic models, simulates reductive changes experiments, mutants' generation, and fault detection effectiveness measurement. The following subsections describe each step in detail.

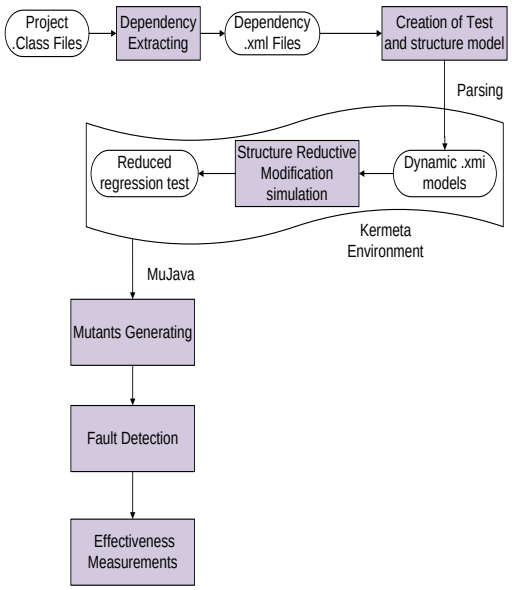

Fig. 1. Input, output and major steps of the proposed approach

A. Dependency Extraction

Dependency happens when one class in the system makes use of another in order to accomplish a specific task. For instance, this can occur when an object of one class is used in another class. These dependencies are helpful and valuable for both programmers and testers when making modifications on the system. The process of finding and discovering dependencies among classes within the system is called extraction. Dependency finder is an open source software tool which is available on [30], and it has been used in many research areas [26, 27, 28]. This tool discovers and reveals three different and specific dependencies: (1) feature to feature (2) feature to class and class to class. The feature means any part of class such as attribute name, method name and constructors. This tool extracts all the dependencies from any type of compiled Java such as Class files, ZIP files, and JAR files.

B. Dynamic test and structure models

We studied test cases dependencies in Java software system. In particular, we created a met model that can be utilized to help in reducing the regression tests after software system modification. In this paper, the meta-model plays a major role in propagating the changes from software structure to test model.

Figure 2 presents a meta model revealing the dependencies in a test model for a given Java software system.

In order to design and build a meta-model, we have chosen Eclipse Modeling Framework [14].EMF is a modeling framework and code generation facility for building tools and other applications based on a structured data model.

The EMF code generation facility is capable of generating everything needed to build a complete editor for an EMF model.

As shown in Figure 2, each test case in the model is composed mainly of two dependencies. (1) Test Hierarchical: for the current test to be run, other tests must be executed and pass (2) Internal: consisting of the Component Under Test (CUT), test drivers, and test stubs. Keeping information on the component under test allows maintaining the traceability links with associated elements in the test model such as scaffolding test.

In order to automatically create the dynamic test and structure models (.xmi), we created a Java based parser to catch the required information from the .xml files that were populated by Dependency finder tool. Dependency tool provides method to address the entire test suite and component structure dependency. We picked naming convention a strategy to manage the traceability.

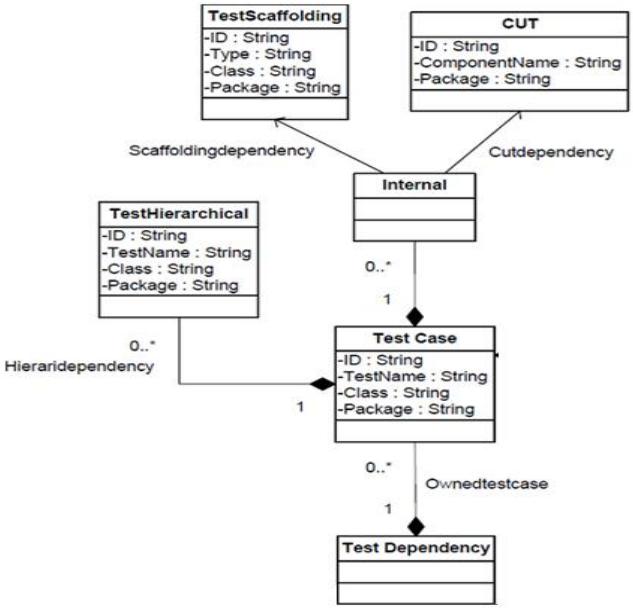

Fig. 2. A meta-model to support regression test reduction technique

C. Simulating Reductive Modification

Reductive modification occurs when the software maintenance or evolution is about to remove existing component interfaces and their implementations from the system. Removing component from the body of the software system is required to remove the unit tests that belong to that element from the test suite. As the unit test checks a single assumption about the behavior of the component.

In this paper, we address the cases when the targeted component for reductive simulation has a dependent and/or dependee. Where the dependees are components and/or tests that are called by the targeted component. In the case of the targeted component has dependees, the integration test that validated the interaction between the targeted components and dependees might be removed from the test suites.

Our assumption is that there is no cyclic dependency in the software projects (i.e. component A depends on component B means, removing A will not affect B as the dependency is unidirectional). The last case occurs when the targeted component has dependents where removing the component

will affect the work of its dependents. Necessary updates to the integration tests that validate the interaction between the targeted component and its dependents will be made.

Software structure and test models instantiation and propagation were achieved using Kermeta, which facilitates the programmatic manipulation of EMF models (.ecore files).

Project Name	# of Reductive Experiments	Percentage of Unit Test Case Reduction	Percentage of Integrated Test Case Reduction
BlackJack	**5**	**43%**	**18%**
PureMVC	**10**	**23%**	**77%**
RealState	**13**	**29%**	**43%**

Kermeta [29] is a meta-modeling language which allows describing both the structure and the behavior of models. Kermeta is intended to be used as the core language of a model

oriented platform. It has been designed to be a common basis to implement Metadata languages, action languages, constraint languages or transformation languages.

Kermeta therefore provided us with a programming environment with which we could set up our simulation.

Mutants generating, and fault detection effectiveness measurement will be described in details in E.1 and E.2 subsections.

D. Experimental Data

The projects that were used in this study are four open-source applications implemented in Java. Table I shows a summary of the selected applications. The selected applications are different in the development processes, features, goals, and the domain.

TABLE I. APPLICATIONS UNDER STUDY

Project	# Classes	# Methods	Source
RealState	57	336	http://realsearchgroup.com /rose/
PureMVC	22	129	http://puremvc.org/
BlackJack	18	102	https://code.google.com/p/ blackjack

The two game applications (RealState and BlackJack) represent applications where systems have to interact to satisfy the logic rules of the underlying strategy of the game logic. Since Java web-applications are widely used nowadays, we chose PureMVC which implements the famous web design pattern Model-View-Controller (MVC).

E. Result and Discussion

Eclipse Modeling Framework (EMF) provides a method to help us in loading, changing and saving software structure and test models by using Kermeta [29]. We used Kermeta language and environment to simulate a reductive change in the abovementioned Java systems. This was achieved by creating and applying a transformation mechanism to the projects under study. Our approach created and applied a set of transformative actions to update and reduce the test models. We arbitrarily simulated 14 reductive changes in RealState project, 10 reductive changes in PureMVC project, and 6 reductive changes in BlackJack project.

1) Test Suite Size Reduction

Table II shows the total number of reductive experiments in each system under test, along with the percentage that reveals the ability of our proposed approach is to reduce test suite size of the three systems. We measure the percentage of test cases reduction for each component that was targeted in the reductive simulation, and then we calculate the mean reduction percentage for the unit and integration test in each system.

TABLE II. TEST SUITE SIZE AFTER SELECTION

We performed 5, 10, and 13 reductive experiments in BlackJack, PureMVC, and RealState, respectively. Our approach was able to reduce 43% of total unit cases and 18% of the integrated test cases in BlackJack system. Interestingly, in PureMVC the proposed approach performed the best reduction in the integration test cases which is about 77%. Finally, the approach reduced 29% and 43% of the total unit and integration test suite respectively in RealState system.

1) Fault Detection

Test selection techniques and after system modification are targeted to reduce the cost of regression test by choosing a portion of an existing test suite, these techniques might lead to lower fault detection effectiveness by neglecting crucial test cases that detect an existing faults. The trade-off between selecting a subset of test cases in order to reduce test suite and fault detection effectiveness should be addressed when we run a specific regression test selection technique.

To measure the effectiveness of fault detection using the proposed approach, we compared it with retest-all approach. Retest-all technique simply reuses all existing test cases after system modification.

In order to measure the effectiveness of the proposed approach, inter-class mutants were seeded into the source code of the above mentioned systems automatically by MuJava tool [15]. We ran retest all technique to examine how many mutants could be killed by executing all the test cases associated with each project (both unit and integrated test cases). We then execute the same mutants against the reduced test suite that is produced by our approach. Finally, we compare the number of killed mutants using the two approaches, that is, retest all test cases approach and the suggested reductive test case approach. Table III shows the fault detection effectiveness of selected test suites using our approach in comparison with rerun-all test cases regression technique.

TABLE III. FAULT DETECTION EFFECTIVENESS AFTER SELECTION

Project Name	# of Mutants	Killed Mutants	
		Retest All	After Reductive Changes
BlackJack	**25**	**13**	**13**
PureMVC	**86**	**35**	**35**
RealState	**121**	**68**	**68**

Table III shows the total number of mutants generated by MuJava and the number of killed mutants using our and retest all approache. MuJava generates 25 mutants from the

BlackJack project where 13 of them have been killed after executing all test cases (without reduction). The results show that our approach and after selection a subset of existing test cases is able to achieve the same degree of effectiveness in uncovering mutants in comparison with retest all technique. After executing the subset test cases which were selected by our approach on the systems under study, retest all and our approach killed equal number of mutants 13 out of 25, 35 out of 86, and 68 out of 121. Although the selected test cases were not detecting all seeded mutants, yet they reduced the test suite and achieved the effectiveness of retest all technique.

IV. Conclusion and Future Work

RTS is an approach used in reducing the number of test cases to run on the modified software. We employ meta-models to support regression test reduction. Our approach facilitates tracing crucial items in test models and its corresponding item in structure model of a Java system, when any changes take place in the component structure model of the system under test (reductive modification), component meta model will specifies and transmits changes should be taken to update the test model(removing, updating, and adding test cases). The result of our experiments reveals how our approach reduced test suite effectively without influence the fault detection effectiveness in comparison with retest-all regression test selection technique.

In future, we intend to perform controlled experiments to compare our approach with other regression test selection techniques are existed in the literature. We intend to use big Java applications to measure the effectiveness of our approach in detecting errors.

References

[1] Leung, H. K. N., and White, L. 1989. Insignts into regression testing. Proceedings of the Conference on Software Maintenance—1989 October, 60–69.

[2] Brown, P. A., and Hoffman, D. 1990. The application of module regression testing at TRIUMF. Nuclear Instruments and Methods in Physics Research Section A, .A293(1–2): 377–381.

[3] Binkley, D. 1992. Using semantic differencing to reduce the cost of regression testing. Proceedings of the Conference on Software Maintenance—1992 November, 41–50.

[4] Leung, H. K. N., and White, L. 1990. A study of integration testing and software regression at the integration level. Proceedings of the Conference on Software Maintenance—1990 November, 290–300.

[5] Leung, H. K. N., and White, L. J. 1991. A cost model to compare regression test strategies. Proceedings of the Conference on Software Maintenance—1991 October, 201–208.

[6] Lewis, R., Beck, D. W., and Hartmann, J. 1989. Assay—a tool to support regression testing. ESEC '89. 2nd European Software Engineering Conference Proceedings September, 487–496.

[7] H. Agrawal, J. Horgan, E. Krauser, and S. London, "Incremental Regression Testing," Proc. Conf. Software Maintenance, pp. 348–357, Sept. 1993.

[8] R. Gupta, M.J. Harrold, and M.L. Soffa, "An Approach to Regression Testing Using Slicing," Proc. Conf. Software Maintenance, pp. 299–308, Nov. 1992.

[9] T. Ball, "On the Limit of Control Flow Analysis for Regression Test Selection," Proc. Int'l Symp. Software Testing and Analysis,ISSTA, Mar. 1998.

[10] S. Bates and S. Horwitz, "Incremental Program Testing Using Program Dependence Graphs," Proc. 20th ACM Symp. Principles of Programming Languages, Jan. 1993.

[11] P. Benedusi, A. Cimitile, and U. De Carlini, "Post-Maintenance Testing Based on Path Change Analysis," Proc. Conf. Software Maintenance, pp. 352–361, Oct. 1988.

[12] D. Binkely, "Semantics Guided Regression Test Cost Reduction," IEEE Trans. Software Eng., vol. 23, no. 8, Aug. 1997.

[13] D. Binkley, "Reducing the Cost of Regression Testing by Semantics Guided Test Case Selection," Proc. Conf. Software Maintenance, Oct. 1995.

[14] Eclipse Foundation, *Eclipse Modeling Framework*, August 2003, http://www.eclipse.org/modeling/emf/ (July 2013).

[15] Yu-Seung Ma, Jeff Offutt, and Yong Rae Kwon, Mujava: an automated class mutation system: Research articles, Softw. Test. Verif. Reliab. 15 (2005), 97–133.

[16] Ammar Masood, Rafae Bhatti, ArifGhafoor, and Aditya P. Mathur, Scalable and effective test generation for role-based access control systems, IEEE Trans. Softw.Eng. 35 (2009), 654–668.

[17] Swarnendu Biswas and RajibMall . Regression Test Selection Techniques: A Survey. Informatica 35 (2011) 289–321

[18] E. Engström, P. Runeson, and M. Skoglund. A systematic review on regression test selection techniques. Information and Software Technology, 52(1):14–30, January 2010.

[19] E. Engström, M. Skoglund, and P. Runeson. Empirical evaluations of regression test selection techniques: a systematic review. In Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement, pages 22–31, 2008.

[20] J. Bible, G. Rothermel, and D. Rosenblum. A comparative study of coarse- and fine-grained safe regression test-selection techniques. ACM Transactions on Software Engineering and Methodology, 10(2):149–183, April 2001.

[21] FISCHER, K., RAJI, F., AND CHRUSCKICKI, A. 1981. A methodology for retesting modified software. In Proceedings of the National Tele. Conference B-6-3 (Nov.). 1–6.

[22] H. Larsson and K. Burbeck. Codex - an automatic model view controller engineering system. In Proceedings of the Workshop on Model Driven Architecture: Foundations and Applications, Enschede, The Netherlands, June 2003.

[23] Igor Ivkovic and Kostas Kontogiannis. Tracing Evolution Changes of Software Artifacts through Model Synchronization. Proceedings of the 20th IEEE International Conference on Software Maintenance (ICSM'04).

[24] Abdallah Qusef, Rocco Oliveto, and Andrea De Lucia, Recovering traceability links between unit tests and classes under test: An improved method, Proceedings of the 2010 IEEE International Conference on Software Maintenance (Washington, DC, USA), ICSM '10, IEEE Computer Society, 2010, pp. 1–10.

[25] Bart Van Rompaey and Serge Demeyer, Establishing traceability links between unit test cases and units under test., CSMR (Andreas Winter, Rudolf Ferenc, and JensKnodel, eds.), IEEE, 2009, pp. 209–218.

[26] McCartin J. Tempero E. Dietrich, J. and S. M. A. Shah, On the existence ofhigh-impact refactoring opportunities in programs, Australasian Computer ScienceConference (ACSC 2012) (Melbourne, Australia) (M. Reynolds and B Thomas, eds.),CRPIT, vol. 122, ACS, 2012, pp. 37–48.

[27] RdigerLincke, Jonas Lundberg, and WelfLwe, Comparing software metrics tools., ISSTA (Barbara G. Ryder and Andreas Zeller, eds.), ACM, 2008, pp. 131–142.

[28] Alexander Serebrenik, Serguei A. Roubtsov, and Mark van den Brand, Dn-basedarchitecture assessment of java open source software systems., ICPC, IEEE ComputerSociety, 2009, pp. 198–207.

[29] Triskell Team, *Kermeta - Breathe life into your metamodels*, October 2005,http://www.kermeta.org/ (July 2013).

[30] Tessier J., Dependency finder, 2008, http://depfind.sourceforge.net