Project plan for degree projects

DV 2572: Master Thesis in Computer Science

October 8, 2017

Thesis	Tentative title	Generate test selection statistics with automated selective mutation : Industrial Application
	Classification	1) Automated testing 2) Test selection 3) Selective mutation
Student 1	Name	Gamini Devi Charan
	e-Mail	dega16@student.bth.se
	Social security no.	950331-8256
	Visa expiration date	12-31-2017
Student 2	Name	
	e-Mail	
	Social security no.	
	Visa expiration date	
Supervisor	Name and title	Omid gholami
	e-Mail	omid.gholami@bth.se
	Department	Department of Computer Science and Engineering
External	Name and title	
	e-Mail	
	Company/HEI	

1 Introduction

Regression testing is the important part in revising the system when changes are done. When a new functionality is introduced it is required to check if the system hasn't regressed from the bugs that might been introduced from these new functionalities [1]. In recent years, with the increase in code bases and test suites there isn't enough time to run the whole regression tests especially when you are trying to continuously integrate changes every day. So making regression testing time efficient without losing its bug detection

capabilities is the new policy for this continuous integration generation. The usage of traditional methods makes it very expensive to implement in CI.

1.1 Regression testing strategies

The main approaches used to address regression testing challenges are [2]:

1. Test suite reduction(minimisation)
2. Test case prioritization
3. Test selection

1.1.1 Test suite reduction

Test suite reduction method aims to find the redundant test cases and then take them out of the test suite to reduce its size [2]. Reduced test suites are quicker to run. Schroder and Korel had an approach, each output variable has a set of inputs that affect the outcome. Combine these to make set of inputs to make a test suite [3]. This works on the assumption that each requirement can be satisfied by one test case and if the test requirement is functional rather than structural then reduction doesn't work in this case [2].

1.1.2 Test case prioritization

Prioritization is the ordering of test cases for testing depending on their relevance. Benefits testers, even if the testing stops at some point the most relevant tests are run first [2]. So test cases are executed in a specific order for early bug detection. Sherrif gave a technique on combining clusters of software artefacts to get the order [4]. Use of mutation score for prioritization was used by Rothermel [5]. Prioritise test cases based on their mutation score (mutants exposed by test cases). The objective of these strategies is mainly to reduce the fault detection time i.e. finding faults earlier and increase reliability at faster rate [5].

1.1.3 Test selection

Test selection is choosing a subset of tests from test suite. This is based on metrics that calculate coverage by tests for new code. In test selection test cases that are relevant to changes done to SUT are selected. Harrold and Soffa gave a data flow analysis technique for unit testing in maintenance phase [6]. Taha, Thebaut and Liu made attest selection framework on incremental data flow algorithm. One flaw with these data flow is they don't detect modifications that are unrelated to data flow change [7]. Agarwal gave new selection techniques on program slicing technique. A dynamic slice of program corresponding to test case is that statement in execution slice that influences output. If new statement doesn't have any variable this won't affect existing slices result in empty selection [8].

1.2 Mutation

Mutation analysis of a program p is generating a set of faulty of programs p' called mutants which are syntactically correct but have different output than original program.

Program p	Mutant p'
...	...
if ($a > 0$ && $b > 0$)	if ($a > 0$ || $b > 0$)
return 1;	return 1;
...	...

Table 1: Example of mutation [9].

Although mutation testing is very reliable on testing the quality of test suites. It is very expensive to practice because of the execution costs of the large sets of mutants produced [9]. Other problems are the amount of human involvement required that is checking the original program's output to that of the mutated ones. These make mutation analysis very hard to practice on industrial level. Barbosa proved that certain selected mutation operators have achieved 99.6% from 65% reduced number of operators used [9].

I.e. Test case reduction removes redundant test cases. Prioritization is to change order of test cases but run all tests eventually and selection is to run only relevant test cases and running those only each time [2].

1.3 Problem statement

These strategies mainly involve execution of code (white box analysis) and require analysis each time a new change is done to code base. Can't be applied to functional test suites. No research is done in this context on industrial level.

My research at Axis communications is to prove the concept that mutation (automatic) of code packages using selective mutation and testing these error injected code against functional test suites can correlate code packages and test cases with error data. These correlations data can be saved and used for test statistical test selection without the need to run analysis every time like in traditional methods [10]. Also explore methods to reduce costs of mutation analysis while able to make correlations.

2 Aim and objectives

2.1 Aim

The aim of this research is correlating code packages to test cases and use this statistical test selection data - the correlation of code packages to test cases is done by automated selective mutation. These correlations can help for faster tests selection [10].

Motivation: New software agile development methods continuous integration and continuous development are very important to stay ahead. The regression test suites require an extreme amounts of time to run complete tests and no technique for functional tests. We find our approach quicker as it doesn't require analysis of code every time test selection is

being done from historical data [10] and this method can also be used for functional test suites.

2.2 Objectives

- Analyse the existing research on automated mutation analysis and choose a method to reduce computational costs.
- Perform automated mutations to functional code packages with an existing or designed mutation tool.
- Run automated functional test suites on mutated code and correlate mutated code packages to failed test cases.
- Results are noted down as dictionaries of correlations (code to tests). These can be used for future test selections on these code packages.

3 Research questions

RQ 1) How to correlate/map code packages to test cases? Motivation: one way to correlate is to have the failed tests data from builds done by developers and correlate the code to tests from them. But not possible in many cases so we want to get the failed tests data by introducing errors and use this data for correlations [10].

RQ 2) How can selective mutation manage the increase in execution and computational costs from mutations? Motivation: mutation is going to increase the Lines of code that we are going to test. This in return will increase the execution and computational costs. So we need to reduce the execution costs or this approach might be more expensive. So use of selective mutation could improve situation for this case.

RQ 3) what are the ways to make sure that tests are definitely going to fail for mutated code? Motivation: we need the tests to definitely fail for our introduced mutants. If not, we won't make any correlations for tests selection. We need to make the mutants very hard to kill for tests.

4 Expected outcomes

The following are the expected outcomes for the formulated research questions respectively:

- Working proof of concept to obtain correlations from code packages to tests.
- Review on better ways to reduce the operational and computational costs of mutation by using available tools.
- Dictionary of correlations from code packages to tests which can be used for test selection.

5 Method

The research methodologies that will be used to answer the research questions are as follows:

5.1 Experimentation

Research problem is to correlate functional code packages to test cases. We can't use traditional static analysis as we want to use a method for functional tests [2]. To solve this we need this research by experimentation as we are going to work on different operators of mutation to get the right effective ones. A mutation tool is going to be used to make changes, these mutations are repeated until correlations are made. This experiment is done with an automated mutation tool (existing or designed depending on needs) and then testing on an automated functional testing framework. The inputs are going to be the mutated code packages and output is failed test cases. The data collection is done from the automated testing framework and then can be viewed as dictionaries. The correlations are from the failed tests cases caused by mutations. So there is no reason or way to validate the results.

5.1.1 Independent variables

The independent variables are manipulated to get the quantifiable or qualitative results from the experiment. Independent variable is the input, in our case mutated code packages are the inputs. Which are controlled and manipulated to get the results for our research.

5.1.2 Dependent variables

The dependent variables are the ones affected by the change in independent variables. In our case dependent variables, output are failed test cases from mutations. With these output we can answer RQ 1.

5.2 Literature review

A literature study on the mutation testing methods can help us find a way to reduce the execution and computational costs of mutation and how to steadily make the tests fail for error data i.e. make mutations stronger, which can get the results for RQ1. This can help us answer the RQs 2 and 3.

6 Time and activity plan

Phases of project	List of activities	Start date	End date
Phase I	Selection of research domain	06-01-2017	06-01-2017
	Identifying the research gap	06-01-2017	06 -03-2017
	Initial literature study(automated testing)	06-04-2017	06-20-2017
	Project plan documentation	06-21-2017	06-29-2017
	Submission of Project plan	-	10-08-2017
Phase II	Secondary literature study(mutation)	06-30-2017	07-15-2017

	Review	07-16-2017	07-30-2017
	Mutation tool selection	07-31-2017	08-18-2017
	Mutation of code and testing	08-19-2017	10-20-2017
	Data collection	10-21-2017	11-10-2017
	Prepare dictionaries of correlations	11-11-2017	11-30-2017
	Thesis draft documentation	12-01-2017	12-31-2017
Phase III	Submission of Thesis draft	-	01-10-2018
	Preparation for thesis defence	01-11-2018	01-21-2018
	Submission of opposition report	-	01-21-2018
	Thesis presentation and defence	01-22-2018	01-24-2018
	Final thesis submission	-	02-04-2018

Table 2: Time and Activity plan

7 Risk management

Risk	Severity	Cause	Possible Mitigation
Failed mutations or weak test cases	High	Improper mutations can cause the experiment to fail as the test cases can detect the mutations or weak test cases can't detect the mutations. Eventually we will be left with no correlations.	Proper selected mutations and work on stable test suites.
Leftover test cases	Medium	The leftover test cases can be because they are irrelevant to the code or failed mutation.	Irrelevant test cases should be omitted and code must be tested again.

Table 3: Risk management

References

[1] P. E. Strandberg, W. Afzal, T. J. Ostrand, E. J. Weyuker, and D. Sundmark, "Automated System-Level Regression Test Prioritization in a Nutshell," *IEEE Softw.*, vol. 34, no. 4, pp. 30–37, 2017.

[2] "S. Yoo, M. Harman, 'Regression testing minimization selection and prioritization: A survey', Software Testing Verification and Reliability, 2010."

[3] korel B, Schroeder PJ, "Black-box test reduction using input-output analysis. SIGSOFT Software Engineering Notes 2000; 25(5):173–177."

[4] "Sherriff M, Lake M, Williams L. Prioritization of regression tests using singular value decomposition with empirical change records. Proceedings of the The 18th IEEE

International Symposium on Software Reliability (ISSRE 2007), IEEE Computer Society: Washington, DC, USA, 2007; 81–90."

[5] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Test case prioritization: an empirical study," in *IEEE International Conference on Software Maintenance, 1999. (ICSM '99) Proceedings*, 1999, pp. 179–188.

[6] "Harrold MJ, Soffa ML. An incremental approach to unit testing during maintenance. Proceedings of the International Conference on Software Maintenance (ICSM 1998), IEEE Computer Society Press, 1988; 362–367."

[7] "Taha AB, Thebaut SM, Liu SS. An approach to software fault localization and revalidation based on incremental data flow analysis. Proceedings of the International Computer Software and Applications Conference (COMPSAC 1989), IEEE Computer Society Press, 1989; 527–534."

[8] "Agrawal H, Horgan JR, Krauser EW, London SA. Incremental regression testing. Proceedings of the International Conference on Software Maintenance (ICSM 1993), IEEE Computer Society, 1993; 348–357."

[9] Y. Jia and M. Harman, "An Analysis and Survey of the Development of Mutation Testing," *IEEE Trans. Softw. Eng.*, vol. 37, no. 5, pp. 649–678, Sep. 2011.

[10] E. D. Ekelund and E. Engström, "Efficient regression testing based on test history: An industrial evaluation," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2015, pp. 449–457.