TestMine: Mining test evolution for improved software test selection

Wasif Afzal

1 Introduction and motivation

There is an increasing reliance on software in our daily activities. This reliance mandates that software is able to meet its users' expectations, otherwise the consequences can be undesirable. During software development a range of verification and validation (V&V) activities are performed with the goal of developing software that meets its requirements. Software testing is one major V&V activity which aims to find *as many defects as possible, as early as possible in the development phase, under given constraints of cost and schedule* [1]. Clearly this is a challenging task, simply because there are not enough resources to test a software exhaustively. As a result, a lot of research has been done to make software testing efficient and effective.

Software testing is typically performed at different levels, namely unit/component, integration, system and acceptance test levels. Also testing is performed as part of a specific life cycle model; agile development being the latest model. In agile, as the software is built incrementally, testing has to be a continuous process that needs to adapt with the changing nature of requirements. Issues such as timely identification of integration and regression issues and efficient system testing becomes critical concerns in an increment.

In agile the iterations are short, e.g., in Scrum each sprint is typically lasting one and four weeks. With the addition of new functionality and continuous fixing of issues in an increment, regression testing has to be done at its best. According to Engström et al. [4], iterative development strategies require frequent retesting of previously tested functions due to changes in related code and therefore it is important to use regression strategies that are efficient. As the number of increments increase, typically the regression test suite grows in size and becomes more expensive to run in each increment. This means that re-running every test case in the test suite will be expensive. This approach is generally called as *retest all* and is the simplest regression test strategy [6]. It is then natural to think about regression test selection (RTS) techniques as part of the overall regression test strategy.

RTS techniques select subset of test cases from the test suite. It differs from test suite minimization or test suite reduction problem in the sense that it is modification-aware, i.e., the selection is specific to the current version of the program and is focussed on the modified parts of the program [13]. RTS thus verifies the behaviour of modified software. More formally, given an initial program *P*, an original test suite *T* associated with *P* and a modified version of the program *P'*, RTS techniques find a subset of *T*, *T'* to test *P'*.

RTS techniques have its own share of challenges. Such techniques can be costly to run and can miss potentially fault revealing test cases. Therefore it is always a trade-off between time required to select and run test cases and the fault detection ability of test cases [6]. This trade-off is also highlighted by Engström et al. [4]: *[...] there is a trade-off between cost reduction and defect detection ability. This is the case in all test selections, and none of the evaluated techniques seems to have done any major breakthrough in solving this trade-off.*

There are different classification schemes for RTS techniques but the most used classification property is whether these techniques are *safe* or *unsafe* [4]. A safe RTS technique ensures that the test suite *T'* has all tests in *T* that execute code that was changed from *P* to *P'* [7]. So with safe RTS techniques the faults found with tests in *T* are also found with tests in *T'* [4]. Harrold [7] calls such tests as *modification traversing* for *P* and *P'*. In other words if there is a modification traversing test case in the test suite, it will definitely be selected by a safe RTS technique [13]. However Rothermel and Harrold [10] shows that precisely identifying modification traversing tests in T is difficult in practice due to non-determinism in programs and test execution, e.g., testing of a system ported to different operating system. As a result, several RTS techniques can be found in literature with varying precision and efficiency. This, again, helps explain the trade-off that exists in RTS techniques between cost and defect detection effectiveness. An unsafe RTS technique is one that may miss tests from *T'* that are modification traversing. An unsafe RTS technique may be beneficial in cases where parts of *P'* remain untested (or not covered) after running *T'*. A simple example of unsafe RTS technique is random selection of test cases [4].

Apart from the safe-unsafe classification, Yu et al. [14] presents another interesting multidimensional way to classify (safe) RTS techniques in terms of *software*, *bug* and *data* dimensions. Within the *software* dimension, the two most important research directions are categorised as *code-based* and *model-based* approaches. In deed most of the RTS research falls in the software dimension where techniques operate at different granularity levels; Orso et al. [9] differentiates between techniques operating at high-level (such as at the method or class level) and those operating at low-level (such as at the level of statements). This is to say that a large body of literature on RTS techniques analyse code, at different abstraction levels, to seek modifications in the program under test. This large body of evidence on RTS techniques focus at unit level testing. This is also acknowledged in the survey paper by Yoo and Harman [13]: *"A majority of existing regression testing techniques reply upon structural information about the SUT, such as data-flow analysis, CFG analysis, program slices and structural coverage"*. The model-based approaches differ from source code being an input to having other software artefacts, such as activity diagrams and other UML based design documents. These specification-based RTS methods aim to cover certain features of the model such as all transition-pairs and all round trip paths. These methods are applicable at integration and system test levels, but as Yoo and Harman [13] points out, these model-based approaches assume the presence of traceability from specifications to code-level artefacts and test cases, which is harder to achieve in practice.

From a *bug* dimension, there are RTS methods based on the fix-cache algorithm proposed by Kim et al. [8]. The test cases are linked to source code files and are prioritised depending on the fault-proneness of the files, if they are changed [5, 12].

The third and the last dimension of RTS techniques by Yue et al. [14] is based on *data* where test cases that are most likely to reveal a fault are selected (e.g., by using execution profiles as in [15]).

What is interesting here is that the *bug* and *data* dimensions of RTS techniques open up opportunities to apply machine learning and data mining algorithms for RTS. This represents, in one way, a significant departure from *code-based* approaches where much effort is applied to identify modifications in the program under test. Also unlike the code-based approaches for RTS, the *bug* and *data* dimensions of RTS techniques are more likely to be applicable to higher levels of testing, such as integration and system. Support for *high-level* regression testing is also a way to extend the applicability of RTS techniques from over-simplistic, small-scale academic problems to real-world software systems as a whole. The techniques that are based *only* on the data generated during software development are likely to add little overhead in terms of additional effort. It seems rational to leverage upon such techniques for reducing the cost of regression testing.

This data-centred approach of solving problems is at the heart of what we now know as *software analytics*. Software analytics is about using potentially large amounts of data for real and actionable insights [2]. Software analytics employs several well-known mechanisms such as trend analysis, classification algorithms, predictive analytics, optimisation, simulation and data- and text-mining.

The fundamental theme of this project is to investigate if software analytics (using test evolution data) can help select a more cost-effective and efficient test suite at system test level. Such an investigation is challenging from multiple perspectives. First, there is scarcity of research on test selection at *system* test level. Given that testing at system test level occurs late in the life cycle, with potentially less time left, any intelligent selection at this level would help increase confidence in the effectiveness of system test effort. Second, many of the non-functional properties manifest best at the system level, therefore customised test suites targeting one or more of these properties can be effectively selected. Third, it is still not evident how the test selection strategies at system test level would get integrated in newer development methodologies like agile and lean. Fourth, in cases where code coverage information is missing or is deemed too expensive to gather, it is logical to make use of existing test related data to assist a selection strategy and measure how effectively the new test suite covers artefacts other than code, such as requirements.

2 Scientific questions

The purpose of this project is to use data generated during test executions (or test evolution) to help decide which test cases to select for testing at system test level. The focus here is to leverage data that is already generated and is available to be used for a selection strategy. This project will propose methods and ways to achieve cost-effective, high-level test selection. The project aims to answer the following research questions:

RQ 1 How can regression test selection be done in a cost-effective manner using test evolution data?

RQ 2 What is the effectiveness and efficiency of using test evolution data for regression test selection at system test level?

RQ 3 How can such an approach integrate effectively in agile/lean software development?

3 Project's expected results and effects

It is expected that the project will result in following outcomes:

1. RTS techniques based on software test execution or software test evolution data, thus applicable in cases where detailed code coverage information might be missing or hard to gather.

2. RTS techniques that are applicable at higher levels of testing, e.g., at integration and system test levels.

3. Empirical evaluation of RTS techniques on large, industrial projects to improve scalability and to promote technology transfer.

3.1 Project's impact on business/industrial partners

Selection of an appropriate subset of test cases to execute so that the objectives of testing are met is expected to reduce the cost of testing. This is especially true for regression testing which remains one of the most expensive activities performed during a software system's lifecycle [7]. There is no use in running test cases with no added value, i.e., if there are no gains in terms of coverage and if the tests are not fault revealing. Removing such test cases will not only improve the test suite; it will allow more time to be spent on developing newer and more interesting (fault-revealing) test cases. Intelligent variation is essential for revealing faults in regression testing since if the same older tests are repeated for regression testing then they are incapable of finding newer faults or faults which arise due to potential side-effects of a code change. As Whittaker [11] mentions, variation of testing focus is the key in revealing faults in regression testing. Regression test selection (RTS) techniques is one way to provide intelligent insight in the capability of a test suite.

Also, as Cartaxo et al. [3] notes, in practice test case selection is usually a manual and an ad hoc process that is error prone; the results of this project shall enable a systematic and an intelligent selection of test cases. This shall enable the industrial partners to have sufficient guarantees that their selection strategy is grounded in facts and real data, and is thus more effective in terms of reducing cost and finding more faults.

In more concrete terms, the results of this project will benefit the industrial partners in the following ways:

1. The results will improve the performance of regression testing at higher test levels (integration and system test).

2. The improvement in the performance of regression testing at integration and system test levels would mean less cost for the development project and improved fault detection effectiveness (faults are more costly to fix if found at the customer end).

3. The techniques proposed in this project would help in general test selection (not necessarily in the context of regression testing) where the goal is to find a subset of the original test suite that guarantees the execution of fault revealing test cases.

4. The software engineers learn better and systematic ways of testing software. This includes injecting intelligent variation in the regression testing suite, thus improving test suite effectiveness and efficiency.

5. The insight and experience gained during the implementation of this project can be extended to company-wide projects for broader benefits.

3.2 Project's impact on individual research and the research area

Industrial scalability and cost-effectiveness of RTS techniques at higher-levels of testing are still open research challenges, despite that there is lots of research on RTS techniques in general. There is certainly a lack of applicable methods for RTS at high-level of testing that makes use of test execution or test evolution data. Then there is evidence to suggest that RTS techniques are evaluated against a limited number of programs and test suites and hence it is desirable to engage more actively with industry to show the cost-effectiveness and usefulness of such techniques [13]. This project aims to fulfil these stated gaps in RTS research, working on which is expected to yield substantial research impact. The plan is to disseminate the expected results of this project through several conference and journal papers, thus strengthening the research profile of the applicant as well as MDH.

The goals of this project are complimentary to the expertise of the applicant and MDH. Working in co-production with [partner 1] and [partner 2] will enhance applicant's already existing knowledge in software verification and validation. This also promises to contribute positively in strengthening the newly created Software Testing Laboratory[1] at MDH. The applicant is part of this laboratory which has a vision of promoting industry-relevant research in software testing. The laboratory also hosts the recently awarded KKS project on Testing of Critical System Characteristics (TOCSYC)[2] in which the applicant is also a participant. [say something about spin-off or refine previous sentence more or talk to Paul on this].

4 Project implementation

The planned duration of the project is 24 months. The implementation of the project is divided into several work packages (WPs) as described below.

4.1 WP1: Literature review and domain gathering

Considering that test selection at system test level is not researched extensively, this WP shall gather what research is available on the topic. This will build a knowledge-base for the activities coming in forth-coming WPs. In parallel to the review, this WP shall start gathering domain knowledge regarding designated projects by our partner companies. This will educate the project leader about contextual variables specific to different projects under study. **Deliverable:** The deliverable for this WP will be a conference paper for the review part.

4.2 WP2: Data gathering and cleansing

Since the target of the project is to make use of test evolution data, a fundamental step is to investigate what data is available and what is its quality. This WP will also consider if certain data pre-processing steps need to be carried out. Attribute selection and removal of redundant variables will also be considered a part of this WP. **Deliverable:** The deliverable for this WP will be an internal artefact that will serve as an input for later WPs.

4.3 WP3: Test selection support and solution formulation

In this WP, we will devise mechanisms for relating test evolution data for test selection. This mechanisms would include finding similarities and diversities among tests based on test evolution data. (See more from Hemmati et al. paper). It is expected that several of such mechanisms would be evaluated for both test selection effectiveness and efficiency. **Deliverable:** The deliverable for this WP will be an internal artefact that will serve as an input for later WPs.

[1] `http://www.es.mdh.se/research-groups/27-Software_Testing_Laboratory`

[2] `http://www.es.mdh.se/projects/349-TOCSYC___Testing_of_Critical_System_Characteristics`

4.4 WP4: Experimental evaluation

In this WP, the actual test selection using test evolution data from previous WPs will be carried out and the results will be evaluated for efficiency and effectiveness. Suitable measures for efficiency and effectiveness will be used to assess the performance. **Deliverable:** The deliverable for this WP will be a journal paper.

4.5 WP5: Solution release

This WP shall work towards transferring the solution to more projects in our partner companies. Collection of relevant measurements shall enable any customisations to be made to the solution and would kick-start followup studies. This WP shall help make the solutions sustainable at our partner companies. **Deliverable:** The deliverable for this WP will be a technical report.

5 Project time plan

Include the excel sheet thingy.

6 Professional development plan

The applicant will be working within the Division of Embedded Systems (ES) and more specifically in the Software Testing Laboratory (STL). The Software Testing Laboratory at MDH is newly established with a goal of conducting industry-relevant research in testing of complex software-intensive systems. The attachment with ES and STL will potentially provide a number of exciting growth opportunities in research, education and industrial collaboration.

The applicant is currently a postdoctoral research fellow at STL since late August 2013. This means that the environment is not new and hence the professional development can progress smoothly. The applicant participates in meetings and discussions in STL regularly. There are few research collaborations which are on-going within STL and are set to grow in coming months. It is also useful to mention that the applicant is part of the KKS funded project called TOCSYC. The participation in this project is also resulting in collaboration not only within STL but across different higher education institutes in Sweden.

The applicant is partly responsible for a PhD course on empirical methods in software engineering during Spring 2014. In coming months the applicant aspires to lecture and contribute in the postgraduate course of software verification and validation. Also there are plans to undertake one or more of higher education pedagogy courses. There are also plans to attend to a PhD supervision course during Fall 2014.

There are currently strong relations between MDH and industry. The applicant hopes to leverage such existing strong relations. It is important to mention that Volvo has MDH now as a preferred academic partner. MDH and ABB have such a relationship for sometime now. STL has a member who is associated with SICS partially (Dr. Daniel Sundmark) and therefore this provides a natural source of collaboration. Ericsson in Kista, Stockholm also collaborates with STL through Dr. Sigrid Eldh.

6.1 Career development plan

Following activities are planned in future for applicant's career development:

- Apply for an assistant professorship position at MDH.
- Discuss opportunity and timing to apply for a docent.
- Complete the PhD supervision course.
- Complete a course on higher education pedagogy.
- Get secondary supervision for one or possibly two PhD students.
- Write further funding applications, both individually and in collaboration with other researchers, in and out of MDH.

6.2 Collaboration with the reference group

7 Budget

References

[1] Software and systems engineering – software testing part 1:concepts and definitions. *ISO/IEC/IEEE 29119-1:2013(E)*, pages 1–64, 2013.

[2] R. P. Buse and T. Zimmermann. Analytics for software development. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research (FoSER '10)*, New York, NY, USA, 2010. ACM.

[3] E. G. Cartaxo, P. D. L. Machado, and F. G. O. Neto. On the use of a similarity function for test case selection in the context of model-based testing. *Software Testing, Verification and Reliability*, 21(2):75–100, 2011.

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

[5] E. Engström, P. Runeson, and G. Wikstrand. An empirical evaluation of regression testing based on fix-cache recommendations. In *Proceedings of the 2010 3rd International Conference on Software Testing, Verification and Validation (ICST'10*, Washington, DC, USA, 2010. IEEE Computer Society.

[6] T. L. Graves, M. J. Harrold, J.-M. Kim, A. Porter, and G. Rothermel. An empirical study of regression test selection techniques. *ACM Transactions on Software Engineering Methodology*, 10(2):184–208, 2001.

[7] M. J. Harrold. Testing evolving software. *Journal of Systems and Software*, 47(2-3):173–181, 1999.

[8] S. Kim, T. Zimmermann, E. J. Whitehead Jr., and A. Zeller. Predicting faults from cached history. In *Proceedings of the 29th International Conference on Software Engineering (ICSE'07*, Washington, DC, USA, 2007. IEEE Computer Society.

[9] A. Orso, N. Shi, and M. J. Harrold. Scaling regression testing to large software systems. *SIGSOFT Software Engineering Notes*, 29(6):241–251, 2004.

[10] G. Rothermel and M. J. Harrold. Analyzing regression test selection techniques. *IEEE Transactions on Software Engineering*, 22(8):529–551, 1996.

[11] J. Whittaker. *Exploratory software testing: Tips, tricks, tours, and techniques to guide test design.* Pearson Education, 2009.

[12] G. Wikstrand, R. Feldt, J. Gorantla, W. Zhe, and C. White. Dynamic regression test selection based on a file cache: An industrial evaluation. In *Proceedings of the 2009 International Conference on Software Testing, Verification and Validation (ICST '09)*, 2009.

[13] S. Yoo and M. Harman. Regression testing minimization, selection and prioritization: A survey. *Software Testing, Verification and Reliability*, 22(2):67–120, 2012.

[14] L. Yu, C. Liu, and Y. Zhang. A multidimensional classification of safe regression test selection techniques. In *Proceedings of the 2012 International Conference on Systems and Informatics (ICSAI'12)*, 2012.

[15] C. Zhang, Z. Chen, Z. Zhao, S. Yan, J. Zhang, and B. Xu. An improved regression test selection technique by clustering execution profiles. In *Proceedings of the 2010 10th International Conference on Quality Software (QSIC'10)*, Washington, DC, USA, 2010. IEEE Computer Society.