TestMine: Mining test evolution for improved software testing

Wasif Afzal

1 Introduction

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. Figure 1 shows an example of how testing is an integrated part of an agile increment.

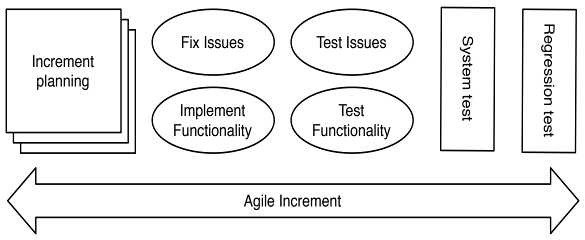

Figure 1: Testing in an agile increment (adapted from [1]).

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 (Figure 1), regression testing has to be done at its best. According to Engström et al. [2], 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 [3]. It is than 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 [7]. 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 [3]. This trade-off is also highlighted by Engström et al. [2]: *[...] 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* [2]. 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'* [4]. So with safe RTS techniques the faults found with tests in *T* are also found with tests in *T'* [2]. Harrold [4] 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 [7]. However Rothermel and Harrold [6] 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 [2].

Apart from the safe-unsafe classification, Yu et al. [8] 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. [5] 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. The model-based approaches represent a departure from source code being an input to having other software artefacts, such as activity diagrams and other UML based design documents.

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

[3] 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.

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

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

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

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

[8] 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.