A Case Study to Compare Regression Test Selection Techniques on Open-Source Software Projects

Uğur YILMAZ[1] and Ayça TARHAN[2]

[1] Aselsan A.Ş. Ankara, Turkey
uguryilmaz@aselsan.com.tr
[2] Hacettepe University. Ankara, Turkey
atarhan@cs.hacettepe.edu.tr

Abstract. Regression testing is the type of testing performed on a modified software to validate integrated parts are functioning properly. Especially with agile development practices being increasingly used, regression testing needs to be fast and practical enough to coexist with the nature of agile development. To satisfy this need, Regression Test Selection (RTS) techniques are proposed to reduce number of tests. Although there are many studies analyzing these techniques and their effectiveness in terms of the number of reduced tests, time and cost; the applicability and practicality aspects have been mostly neglected. To this end, in this paper a case study is carried out to compare highly cited and mostly used RTS techniques. Selected techniques are applied on extensively used and tested open-source software projects, and the result of their comparison in regards of practicality, applicability, performance and cost-effectiveness are discussed.

Keywords: Regression Testing, Regression Test Selection, Case Study, Software Testing.

Açık Kaynak Kodlu Yazılım Projelerinde Regresyon Test Seçim Tekniklerininin Karşılaştırılması için Bir Vaka Çalışması

Uğur YILMAZ[1] and Ayça TARHAN[2]

[1] Aselsan A.Ş. Ankara, Turkey
uguryilmaz@aselsan.com.tr
[2] Hacettepe University. Ankara, Turkey
atarhan@cs.hacettepe.edu.tr

Özet. Regresyon testi, değiştirilmiş bir yazılımda entegre parçaların düzgün çalıştığını doğrulamak için gerçekleştirilen test türüdür. Özellikle çevik geliştirme tekniklerinin daha çok kullanılmasıyla birlikte, regresyon testinin çevik geliştirme ile bir arada bulunmak için hızlı ve pratik olması gerekmektedir. Bu ihtiyacı karşılamak amacıyla regresyon test sayısını azaltmak için regresyon test seçme teknikleri önerilmiştir. Bu tekniklerin azaltılmış test sayısı, zaman, maliyet gibi açılardan etkilerinin incelendiği birçok çalışma olmasına rağmen, uygulanabilirlik ve pratiklik hususları çoğunlukla ihmal edilmiştir. Bu amaçla, çok kullanılan RTS tekniklerini karşılaştırmak için bir vaka çalışması düzenlenmiştir. Seçilen teknikler yaygın olarak kullanılan ve test edilen açık kaynak kodlu yazılım projelerinde uygulanmış ve pratiklik, uygulanabilirlik, performans ve maliyet etkinliği açısından karşılaştırmalı sonuçları paylaşılmıştır.

Keywords: Regresyon Testi, Regresyon Test Seçimi, Vaka Çalışması, Yazılım Test.

1 Introduction

Regression testing is the type of testing performed on a modified software in order to validate newly introduced changes function properly with the existing software [1]. The brute-force way of regression testing is the *retest all* approach in which all of the tests are rerun. Although this approach is the most simple, it is not feasible in terms of performance and cost [2]. Since there is no silver bullet in regression testing, a variety of regression test selection techniques have been proposed in the literature throughout years (e.g. [3]–[13]). There are many techniques developed to address the different issues of regression test selection such as performance, safety and cost-effectiveness, and many empirical studies are carried out to analyze different techniques (e.g.[1], [14], [15], [16]). To evaluate a technique, it is applied to a software with different versions and for each version measured data such as time, number of detected faults and etc. are acquired and then compared to different techniques [17]. However with the advent of agile development practices, the applicability and practicality aspect of regression testing techniques have been mostly neglected. Most techniques are not evaluated enough to make an informed selection of a RTS technique [16].

In this paper, we inspected and reported several well-acknowledged and used modern RTS techniques in respect to their applicability and practicality to ensure a smooth, headless integration to existing open-source software systems, as well as performance and cost-effectiveness. We applied 3 techniques to 3 large open-source systems with multiple revisions and reported as an embedded case study [18].

This paper is structured as follows: Section 2 provides brief background information on RTS techniques used in the study. Section 3 provides an outline of the overall case study including research objectives, design and measures. Section 4 reports the results obtained in the experiments and discussions. Section 5 presents threats to validity of the article. Section 6 provides the conclusion and future work.

2 Background

2.1 Regression Test Selection

Graves et. al. [15] describes Regression Testing as follows: " Let *P* be a procedure or program; let P' be a modified version of *P*; and let *T* be a test suite for *P*. A typical regression test proceeds as follows:
(1) Select $T' \in T$, a set of test cases to execute on P'.
(2) Test P' with T', establishing P''s correctness with respect to T'.
(3) If necessary, create T'', a set of new functional or structural test cases for P'.
(4) Test P' with T'', establishing P''s correctness with respect to T'." Main goal of the Regression Test Selection is the select fewest possible tests that results in most fault location.

A *safe* RTS means all the selected test cases are guaranteed to be modification-revealing [19], hence a test that is omitted that could be affected by the modification could

cause faults to be not detected, but to make a technique *safe* may impose overall performance reductions and implementation overhead.

A *dynamic* RTS is when the technique uses runtime information to analyze the dependencies between two code revisions and select tests to run while running tests [20].

A *static* RTS technique uses compile time information and static analysis methods to determine which tests to run [20].

2.2 RTS Techniques

In this study, 3 RTS techniques are used: Ekstazi (v5.2.0) [12], STARTS [5] and RETEST [4].

Ekstazi collects test dependencies on the file level using checksums of the files. Then, after a code change, it dynamically analyzes the bytecode and determines which tests to select using previously determined dependencies. It can also integrate with other test libraries and handles newly added test files adequately [12]. Ekstazi is a safe dynamic RTS technique.

STARTS is a safe static RTS technique operating at the class level using checksums of the classes during compile time to build test-code dependencies like Ekstazi and uses this information to select impacted test after a code change has occurred [5].

RETEST is a safe graph-traversal algorithm, mainly adopting Rothermel and Harrold's algorithm [3] to Java systems. It constructs a Java Interclass Graph (JIG) based on the Control Flow Graphs (CFG) of the program and iterates on the graph to obtain a coverage matrix between code and tests. Then, when a code change is occurred JIG is constructed again and RETEST selects new tests using new JIG and coverage matrix created earlier [4].

There are many more RTS techniques proposed in literature that address different platforms/languages (C, C++, Web, embedded), and this study focuses on highly adopted/used techniques adopted to JAVA language.

3 Case Study Outline

3.1 Objective

To compare different RTS techniques applicability, practicality, performance and cost-effectiveness, following research questions (RQ) are formed:

- RQ1: *How do RTS techniques compare to each other in terms of applicability and practicality?* This research questions aims to answer how easy it is to setup RTS technique and learn to use it with existing codebase.
- RQ2: *How do RTS techniques compare to each other in terms of performance?* This research question aims to detail a comparison of the performance of different techniques in terms of time, number of tests selected.
- RQ3: *How do RTS techniques compare to each other in terms of cost-effectiveness?* This research question aims to study each technique's cost-effectiveness whether it is beneficial to use it or not.

3.2 Design

The study was designed as an embedded case study. The 3 RTS techniques applied to large open source projects shown in the Table 1. To simplify experiment phase only 10 revisions of the projects are used regardless of total revisions on respected repositories.

Table 1. Open Source Projects Used.

Project Name	kLOC	Rev	Tests
Apache Commons Lang	69.0	10	133
Ant	80.5	10	34-150
JBoss	116.6	10	105-361

Apache Commons Lang is a JAVA utility library that extends functionality of default java.lang packages [21]. Ten revisions are obtained from GitHub repo with the associated tests. Approximate test numbers are given in Table 1.

Ant is a build tool developed for JAVA. It executes task defined in a XML based configuration files [22]. Ten revisions are obtained with associated tests starting from the first revision to the last revision.

JBoss is a JAVA cross-platform application server targeting web-based applications. 10 revisions is obtained with associated tests starting from first revision to last revision [23].

The following measures are computed during experiments:

- End-to-end runtime of tests including regression test selection and testing phases, and
- Number of tests selected.

The experiment steps are shown in Fig 1. Each one of the three RTS technique is applied to each project consecutively in a separated manner and required measures are calculated in each revision of the project. The average values of calculated measures are taken and reported in Section 4.

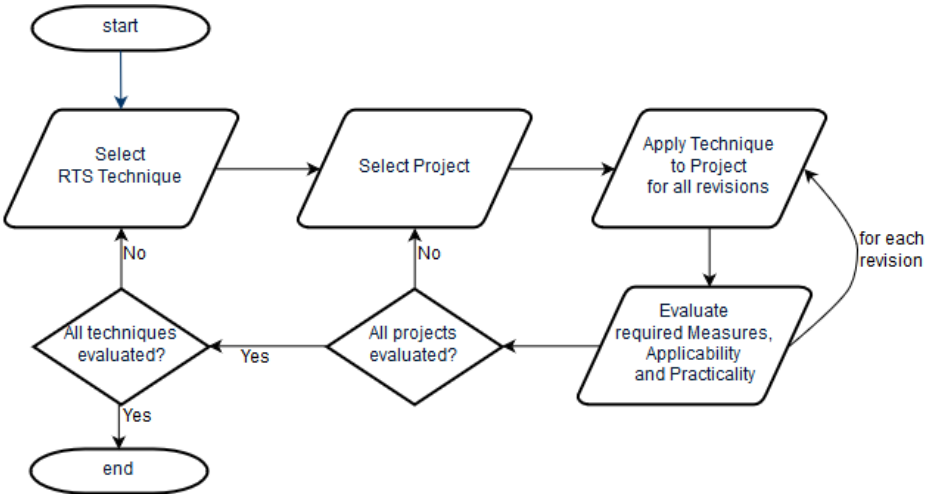

Fig. 1. Experiment Steps

4 Results

4.1 RQ1: How do RTS techniques compare to each other in terms of applicability and practicality?

Ekstazi has different integration options available (Maven, Gradle, Java Agent, Ant target) [12]. All of the options are well documented and easy to use. In addition, Ekstazi could integrate itself into Junit test process meaning there is zero work to be carried out when integrating Ekstazi into an established testing cycle using Junit. However, while running Ekstazi, only two options were available namely, ekstazi (selects and runs tests) and clean (cleans previously collected dependencies). This resulted in a limited available functionality, i.e., if, only selecting the tests were needed it couldn't be performed without also running the tests.

STARTS has only Maven integration which cripples it to work with Non-Maven repositories. STARTS couldn't integrate itself to test frameworks like Ekstazi so specific commands needed to be run while running tests with STARTS. However, unlike Ekstazi STARTS has well documented and different functionalities (only select, only get difference between test, only run tests, etc.) which makes it easier to integrate third party build tools.

RETEST is not available off the shelf line Ekstazi/STARTS so we implemented the algorithm ourselves which was an unsure task which introduced an overhead implementation time and uncertainty about the correct implementation of the technique. Although the implementation is a one-time task, the implementer should consider usage with other build tools or third-party test tools which also introduces maintaining/testing times with supported third party tools.

Table 2. Summary of RTS Techniques' features

Technique/Feature	Availability	Documentation	Functionality	Dependencies*
Ekstazi	High	High	Low	Low
STARTS	Medium	High	Medium	Low
RETEST	Low	Low	N/A	Low

* Low is better

The results are given as a summary in Table 2. Overall, the applicability and practicality of the already available tools is a huge benefit to adopters. While presenting new RTS techniques, the main objective heavily focuses on the technique itself but it is observed that with a broader spectrum of third-party integration support from RTS techniques and easy to implement structures, the usage of the technique would improve.

4.2 RQ2: How do RTS techniques compare to each other in terms of performance?

RTS techniques are assessed in the performance with respect to *retest all* approach to better visualize each technique's outcome.

Table 3. Time relative to *retest all* approach

Project Name	Time (%)		
	Ekstazi	STARTS	RETEST
Apache Commons Lang	55	45	412
Ant	61	55	520
JBoss	64	60	590
Average	60	53	567

Table 3 summarizes the time percentage that took each technique to select and run tests relative to *retest all* approach (%100). In each project the average time of 10 revisions are reported. Ekstazi runs %60 fast compared to retest all approach. STARTS has close results to Ekstazi and runs almost twice as fast compared to retest all approach. RETEST runs 5.5 times slower on average compared to retest all approach but this result could be connected to its JIG generation on each revision. This process introduces a high overhead time especially in large software projects.

Table 3. Number of Tests

Project Name	Tests (%)		
	Ekstazi	STARTS	RETEST
Apache Commons Lang	25	32	60
Ant	40	43	55
JBoss	45	47	42

Table 3 depicts the percentage of tests selected for each project on average with respect to the *retest all* approach. Ekstazi generally selected the lowest ratio of the tests. This can be linked to it being a dynamic RTS technique. STARTS is close to Ekstazi on selecting relatively same number of tests. This can be explained since both of the techniques relies on test dependency on file level. RETEST is the finest-granular technique of all and expected to select the fewer tests but it selected mostly the highest number of tests. This phenomenon could be reasoned with since it operates on much higher sensitivity, it selects more tests that affect the modified code whereas coarser approaches select fewer tests [19].

4.3 RQ3: How do RTS techniques compare to each other in terms of cost-effectiveness?

Due to the lack of fault information related to each revision of the projects used in this study, it is impossible to compare the techniques in regards of the measures such as Average Percentage of Fault Detection (APFD) [19]. Likewise, we couldn't measure the precision and safeness of each technique. Therefore, in comparison of the cost-effectiveness of each technique, other empirical analysis are investigated and reported with the addition of qualitative experiment results.

It is observed that Ekstazi is easy to implement, fast and capable of being integrated to third party software tools used in testing large projects, which makes it a cost-effective regression test selection technique [12], [20], [24]. In our experiment we determined that Ekstazi has negligible (in terms of milliseconds) overhead implementation cost to large repositories.

STARTS has the ability to quickly integrate with existing software projects, although it lacks some of the third party support of other tools. But we experimented the overhead cost of implementing STARTS is relatively low compared to other RTS techniques [20].

RETEST is hard to implement and has a high overhead of implementation/maintenance/testing of the three RTS techniques. Since this technique is slower than the other techniques, it is safe to assume that this technique has lowest cost-effectiveness among other RTS techniques [4], [16].

5 Threats to Validity

In this section, the validity of the case study is discussed in terms of *construct, internal* and *external* validities and reliability[18].

The *construct* validity is related to measurement instruments not being capturing the correct concepts [18]. In order to alleviate this threat, time measured against *retest all* approach since it could be used as a base ground to compare against.

The *internal* validity is related to inner instruments used in study that can affect the authors' judgements [18]. Since we implemented RETEST technique ourselves, it might contain bugs that resulted in skewed outcomes. Moreover, the applicability and

practicality aspect of the techniques are evaluated considering writers' own software ecosystem and trends on modern software developments as of writing. Hence, if the study is repeated in a different time or with different authors having marginally distinct background, diverse decisions could be reached. Furthermore, to eliminate same tests appearing on different revision due to same bug, hence, altering the result of number of tests selected, the failed tests after each revision is reviewed to ensure the cause is not the same fault. This threat could be erased completely by using a much larger number of revisions.

The *external* validity is related to conditions that limits the researchers' generalization capabilities [18]. To eliminate this threat, large and highly used open source projects are selected. However, only inspecting large sized projects, the techniques' effectiveness against small sized projects is overlooked. Some techniques, especially RETEST, could perform better on a small sized project due to having relatively omissible overhead JIG computation costs. Another threat is only 10 revisions are evaluated due to time constraints and since the definition of revision could have different meaning for different version control systems, this could lead to ambiguity when repeating the experiment. Some modern software projects may not run regression tests on each revision. In addition, long-time cost and performance of the RTS techniques could deviate from found results.

The reliability requires that the study can be repeated with the same steps by other researchers. To address the issue, the steps taken performing this study explained in design section (3.2) and in Fig-1.

6 Conclusion and Future Work

Regression Test Selection is an important task that needs to be tackled fast and in an effective way especially in large software projects. To satisfy this need, many RTS techniques, each targeting different goals are proposed in the literature. In this article, we presented the comparison of the mostly used RTS techniques in terms of applicability, practicality, performance and cost-effectiveness in an embedded case study environment. Highly cited, finer-granular RTS techniques developed over years perform worse timewise in today's modern large software projects; on the other hand, the modern RTS techniques use coarser methods to achieve smaller number of selected tests, better performance and easy to use approach which all result in higher adaptation rate from industry.

Unfortunately, we could not have the chance of measuring any fault related metrics which would give more detailed insights about each technique's cost-effectiveness. Advancing on this topic while including more software projects (large and small), with fault history and evaluating more RTS techniques to paint a broader picture could be a promising direction for future work.

References

1 S. Yoo and M. Harman, "Regression Testing Minimisation, Selection and Prioritisation :

A Survey," *Test. Verif. Reliab*, vol. 00, pp. 1–7, 2007.

2 J.-M. Kim and A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments," *Proc. 24th Int. Conf. Softw. Eng. - ICSE '02*, p. 119, 2002.

3 G. Rothermel and M. J. Harrold, "A safe, efficient regression test selection technique," *ACM Trans. Softw. Eng. Methodol.*, vol. 6, no. 2, pp. 173–210, 1997.

4 M. J. Harrold *et al.*, "Regression test selection for Java software," *ACM SIGPLAN Not.*, vol. 36, pp. 312–326, 2001.

5 O. Legunsen, A. Shi, and D. Marinov, "STARTS: STAtic regression test selection," *ASE 2017 - Proc. 32nd IEEE/ACM Int. Conf. Autom. Softw. Eng.*, no. iv, pp. 949–954, 2017.

6 D. Willmor, S. M. Embury, S. E. -, 2005. ICSM'05. Proceedings of The, and U. 2005, "A safe regression test selection technique for database-driven applications," in *ieeexplore.ieee.org*, 2005, vol. 2005, pp. 421–432.

7 M. Ruth *et al.*, "A Safe Regression Test Selection Technique for Web Services," in *Proceedings of the 31st Annual International Computer Software and Applications Conference - Volume 02*, 2007, pp. 0–5.

8 W. S. A. El-hamid, S. S. El-etriby, and M. M. Hadhoud, "A General Regression Test Selection Technique," pp. 893–897, 2010.

9 N. Mansour and W. Statieh, "Regression Test Selection for C# Programs," *Adv. Softw. Eng.*, vol. 2009, pp. 1–10, 2009.

10 R. Malhotra, A. Kaur, and Y. Singh, "A Regression Test Selection and Prioritization Technique," *J. Inf. Process. Syst.*, vol. 6, no. 2, pp. 235–252, 2010.

11 T. Yu, X. Qu, M. Acharya, and G. Rothermel, "Oracle-based Regression Test Selection," *2013 IEEE Sixth Int. Conf. Softw. Testing, Verif. Valid.*, pp. 292–301, 2013.

12 M. Gligoric, L. Eloussi, and D. Marinov, "Ekstazi: Lightweight Test Selection," *Proc. - Int. Conf. Softw. Eng.*, vol. 2, pp. 713–716, 2015.

13 R. Kazmi, D. N. A. Jawawi, R. Mohamad, and I. Ghani, "Effective Regression Test Case Selection," *ACM Comput. Surv.*, vol. 50, no. 2, pp. 1–32, 2017.

14 H. Do and G. Rothermel, "An Empirical Study of Regression Testing Techniques Incorporating Context and Lifetime Factors and Improved Cost-Benefit Models," *Sigsoft'06/Fse-*, pp. 141–151, 2006.

15 T. L. Graves, M. J. Harrold, J.-M. Kim, A. Porter, and G. Rothermel, "An empirical study of regression test selection techniques," *ACM Trans. Softw. Eng. Methodol.*, vol. 10, no. 2, pp. 184–208, 2001.

16 E. Engström, P. Runeson, and M. Skoglund, "A systematic review on regression test selection techniques," *Inf. Softw. Technol.*, vol. 52, no. 1, pp. 14–30, Jan. 2010.

17 G. Rothermel and M. J. Harrold, "A framework for evaluating regression test selection techniques," *Proc. 16th Int. Conf. Softw. Eng.*, pp. 201–210, 1994.

18 R. K. Yin, *Case Study Research. Design and Methods.*, vol. 5, no. 5. 2009.

19 S. Biswas, R. Mall, M. Satpathy, and S. Sukumaran, "Regression test selection techniques: A survey," *Inform.*, vol. 35, no. 3, pp. 289–321, 2011.

20 O. Legunsen, F. Hariri, A. Shi, Y. Lu, L. Zhang, and D. Marinov, "An extensive study of static regression test selection in modern software evolution," *Proc. 2016 24th ACM SIGSOFT Int. Symp. Found. Softw. Eng. - FSE 2016*, pp. 583–594, 2016.

21 “Apache Commons Lang,” 2018. [Online]. Available: https://github.com/apache/commons-lang. [Accessed: 20-May-2018].
22 “Ant.” [Online]. Available: http://sir.unl.edu/content/bios/ant.php. [Accessed: 20-May-2018].
23 “jBoss.” [Online]. Available: http://sir.unl.edu/content/bios/jboss.php. [Accessed: 20-May-2018].
24 N. Dini, A. Sullivan, M. Gligoric, and G. Rothermel, “The Effect of Test Suite Type on Regression Test Selection,” *2016 IEEE 27th Int. Symp. Softw. Reliab. Eng.*, pp. 47–58, 2016.