An empirical evaluation of regression testing based on fix-cache recommendations

Engström, Emelie; Runeson, Per; Wikstrand, Greger

Published in:
[Host publication title missing]

DOI:
10.1109/ICST.2010.40

2010

An Empirical Evaluation of Regression Testing Based on Fix-cache Recommendations

Emelie Engström
Software Engineering Research Group
Dept. of Comp. Science, Lund University
Lund, Sweden
emelie.engstrom@cs.lth.se

Per Runeson
Software Engineering Research Group
Dept. of Comp. Science, Lund University
Lund, Sweden
per.runeson@cs.lth.se

Greger Wikstrand
KnowIT YAHM Sweden AB
Lund, Sweden
gwikstrand@ieee.org

***Abstract*—*Background*: The fix-cache approach to regression test selection was proposed to identify the most fault-prone files and corresponding test cases through analysis of fixed defect reports. *Aim*: The study aims at evaluating the efficiency of this approach, compared to the previous regression test selection strategy in a major corporation, developing embedded systems. *Method*: We launched a post-hoc case study applying the fix-cache selection method during six iterations of development of a multi-million LOC product. The test case execution was monitored through the test management and defect reporting systems of the company. *Results*: From the observations, we conclude that the fix-cache method is more efficient in four iterations. The difference is statistically significant at $\alpha = 0.05$. *Conclusions*: The new method is significantly more efficient in our case study. The study will be replicated in an environment with better control of the test execution.**

I. Introduction

Regression testing is a resource consuming activity in software development. This is particularly true for iterative development approaches, where features are added to existing software in an iterative fashion. Regression testing is performed to ensure that previously functioning software is not corrupted by the changes. Studies indicate that 80% of testing cost is regression testing and more than 50% of software maintenance cost is related to testing [3].

Several techniques for regression test selection are proposed and evaluated. Engström *et al.* recently reviewed the literature in the field [4] and concluded that most of the proposed regression test selection techniques are not feasible to scale up to testing of large complex real time systems. Industry practice on regression testing is mostly based on experience alone, and not on systematic approaches. There is an urgent need to decrease regression testing cost and increase test efficiency in industry.

A pragmatic approach to regression testing is proposed by Wikstrand *et al.* [12]. The basic idea is to link test cases to source files based on information from the test management and defect reporting systems. Test cases are then prioritized with respect to how fault prone their linked files are, if changed. A cache, as proposed by Kim *et al.* [7], is used to monitor which files are fault-prone and fixed in recent iterations. The *fault prediction* effectiveness of the fix-cache method has been shown to be good [12]. However, the efficiency of the *regression testing* based on these recommendations has not been evaluated earlier.

In this paper we report on the first empirical evaluation of test suite efficiency of the fix-cache method. In an industrial setting we compare the efficiency between the traditional manually selected test suites and the test suites recommended by the fix-cache tool. Our results indicate that the tool based selection generates more efficient test suites.

The paper is structured as follows. In Section II we briefly present the regression test selection method under study as well as related work. Section III presents the design and execution of the evaluation case study. In Section IV we analyze the validity of the study and we discuss the results and future work in Section V.

II. Background and Related Work

The regression test selection algorithm being evaluated in this paper was first described by Wikstrand *et al.* [12]. The algorithm is based on three processes: identifying fault prone source files, linking test cases to source files, and recommending test cases. The company where the study was performed, has a defect report management system, where affected files are recorded when a defect report is closed. This was crucial to the effectivness of the described algorithm. The three processes are described below.

a) Identifying fault prone files: When defect reports are closed, the corresponding updated files are marked as hits in a fix-cache as proposed by Kim *et al.* [7]. As recommended in the original paper, the cache size was fixed at 20% of the total number of files. To maintain the size, files were removed from the cache using a least recently used logic.

b) Linking test cases: Also when defect reports are closed, they are traced back to the originating system test case (if any) and marked correspondingly. The test case is thus linked through the closed defect reports to the files which were changed as a result of the fault that was detected when the test case failed.

c) Recommending test cases: When a new test campaign is about to be conducted, the changes on a file level to the product, compared to the previous test iteration, are obtained from the source management system. If a file is both in the fix-cache and has one or more linked test cases, the linked

test cases are recommended for execution. References to the linking defect reports are given as a rationale to aid the test leader in deciding whether to follow the recommendation.

Wikstrand *et al.* reported on the precision of the fix-cache. The hit rate, i.e. the number of files with fixes which were already in the cache, of the cache on a week-by-week basis was found to be 50-80% [12], less than the 73-95% reported by Kim *et al.* [7], although in the same magnitude.

Sheriff *et al.* [11] published a study on a similar approach, although with a focus on clusters of files that tend to be changed together. They evaluated the test case selection and found that the methodology proposed test cases, additional to those based on pure file changes, in 50 % of the cases. However, it is not clear from the evaluation whether these test cases actually found more faults or not.

Engström *et al.* published a comprehensive systematic review of all empirical evaluations of regression test selection techniques [4]. They conclude that only 4 out of 15 case studies are conducted in a large scale context, i.e. larger than 100 000 LOC, and no more than 1 out of 21 experiments is conducted on large scale artifacts. Software size is not the only criterion for making a study realistic, but the observation calls for more industry evaluations of regression testing methods.

Several studies investigate relationships between fault-proneness and various software metrics, among those lines of code is the most straightforward and most investigated [2]. However, the relation is shown to be logarithmic [8], indicating that smaller classes cause relatively more problems than larger ones. In this study, we only use the characteristic of observed fault proneness as a predictor for future fault proneness.

III. EMPIRICAL EVALUATION

A. Research question and method

The aim of the study was primarily to evaluate the efficiency of the fix-cache approach to regression test selection, compared to the previously used regression test selection strategy in a major corporation, developing embedded systems. We refer to efficiency as the number of found faults per selected test case. Our research question is hence:

- Is the fix-cache regression test selection method more efficient than the previously applied experience-based method?

Methods for empirical evaluations include experiments [13] and case studies [10]. Studies of real industrial size are hard to conduct with the level of control required for a formal experiment. Case studies are less controlled, but offer on the other hand a broader spectrum of data to observe. For our evaluation, we have chosen to conduct a case study, in which the data collection and analysis is mostly post hoc.

B. Case study setting and results

The case under study is a development project at ST-Ericsson in Lund, Sweden. ST-Ericsson develops platform software and hardware designs for embedded mobile devices. The part of the product under study comprises several million lines of code. It is developed at multiple sites across three continents.

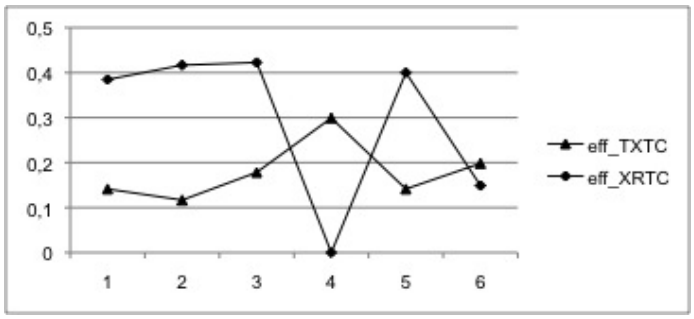

Fig. 1. Fault detection efficiency for each iteration.

TABLE I
NUMBER OF TEST CASES FOR EACH ITERATION. RTC = RECOMMENDED TEST CASES; XRTC = EXECUTED RECOMMENDED TEST CASES; TXTC = TOTAL EXECUTED TEST CASES

Iteration	1	2	3	4	5	6
RTC	27	41	99	1	11	78
XRTC	13	12	71	1	5	47
TXTC	552	480	1301	906	1203	1317

Each week, new increments and fault fixes to a number of the modules in the product are delivered to the main development branch for system and regression testing. In this study, the focus has been on regression test cases from a limited area of system test. The test area in question is representative of the product and tests a cross section of modules and requirements, but we are not able to report any more details about the selected modules for confidentiality reasons.

The fix-cache regression test selection approach was applied during six iterations of regression testing. A list of recommended test cases, based on the method, was delivered to the test department and later followed up by monitoring the test management and defect management databases. Due to lack of control in the case study, not all recommended test cases were executed. We discuss the implications of this in Section III-C. The actual number of test cases selected and executed are presented in Table I, as is the total number of executed test cases, based on the ordinary selection method, which mainly is based on fixed test case priorities and test planning heuristics.

We evaluated the fault detection efficiency, defined as:

$$\text{Eff}_{det} = \frac{\text{\# faults found}}{\text{\# test cases executed}}$$

The efficiency for each of the six iterations is reported in Figure 1. Since only one test case was selected in iteration

TABLE II
NUMBER OF FAILED TESTS FOR EACH ITERATION. XRTC = EXECUTED RECOMMENDED TEST CASES; TXTC = TOTAL EXECUTED TEST CASES

Iteration	1	2	3	4	5	6
XRTC	5	5	30	0	2	7
TXTC	78	56	232	271	170	261

TABLE III
SENSITIVITY ANALYSIS USING EFFICIENCY DATA FOR THE DIFFERENT APPROACHES OVER ITERATIONS 1- 6, COUNTING ITERATION 4 AS AN OUTLIER.

Approach	1	2	3	4	5	6
eff_worst_case	0.19	0.12	0.30	N/A	0.18	0,09
eff_best_case	0.70	0.83	0.59	N/A	0.73	0.49
eff_as_TXTC	0.26	0.20	0.35	N/A	0.26	0.17
eff_as_XRTC	0.38	0.42	0.42	N/A	0.40	0.15
eff_TXTC	0.14	0.12	0.18	0.30	0.14	0.20

4, we consider this iteration being an outlier and it is hence excluded from the subsequent analysis. There are probably factors out of the study control that confuse the picture, since the efficiency of the experience-based method is 0.30 for iteration 4, compared to 0.12-0.18 for the other iterations (see Table III last row).

The underlying data on selected number of test cases and failed tests are reported in Tables I and II respectively. We analyzed the difference between the efficiency of the two approaches using a t-test. There is a significant difference between the two at a 5% significance level ($t = -3.7033, df = 4.629, p = 0.01602$).

C. Sensitivity analysis

Since all the recommended test cases were not executed, as reported in Table I, there is a major threat to the validity of the study that the results are an effect of the properties of the executed set of test cases, rather than the test selection method as such. In order to validate the results, we conducted a sensitivity analysis, calculating theoretical boundaries for the efficiency.

For the sensitivity analysis, the set of Recommended Test Cases are denoted RTC in Figure 2. The Total number of eXecuted Test Cases (TXTC) does not cover all RTC, and hence only the eXecuted share of the Recommended Test Cases (XRTC) subset of RTC is executed. We draw our main conclusions based on XRTC only as we do not know the properties of the set of non-eXecuted share of the Recommended Test Cases (nXRTC).

The worst case, i.e. with the lowest efficiency, would be if the test cases of nXRTC all would pass. The best efficiency case would be if all nXRTC fail, although this is not a realistic case. Two further alternative scenarios are a) if the nXRTC are as efficient as the TXTC subset, and b) if the nXRTC are as efficient as the XRTC subset. In the latter case, the RTC efficiency would be exactly the same as the XRTC efficiency, reported above.

These four alternative scenarios are presented in Figure 3 and the data is tabulated in Table III, using the previously used approach as a reference (eff_TXTC). The worst case is *not* significantly better than the traditional approach ($t = -0.5393, df = 5.246, p = 0.6118$) while the other two approaches are (eff_best_case: $t = -8.3817, df = 4.483, p = 0.0006645$; eff_as_TXTC: $t = -2.7125, df = 5.661, p = 0.0371$).

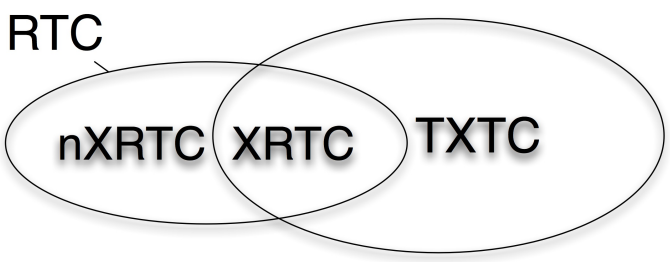

Fig. 2. Diagram relating the sets of test cases to each other. RTC = recommended test cases; nXRTC= non-executed recommended test cases; XRTC = executed recommended test cases; TXTC = total executed test cases

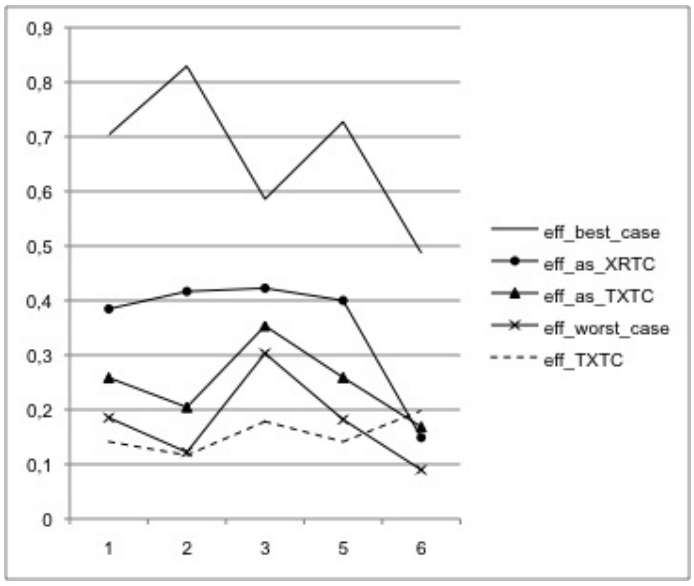

Fig. 3. Sensitivity analysis for iterations 1-3, 5 and 6 - share of test cases detecting defects vs. iterations.

We conclude from the scenario analysis that the efficiency of the fix-cache test case selection method is not due to the incomplete execution of test cases, but the inherent properties of the method itself.

D. Checking assumptions

The fix-cache selection method is based on assumptions concerning *fault churn* and *fault location*. We checked whether these are fulfilled in the studied environment, although with different data sets than the above study.

Fault churn: The fix-cache algorithm is based on the assumption that the faults are not evenly distributed over software modules, and that this distribution is changing over time i.e. there is a *fault churn*. A Pareto-like distribution of faults over modules is statically identified in several studies, e.g. [1], [5], while the dynamic behavior is not studied before, i.e. whether different modules are fault-prone at different occasions.

The assumption was tested by studying post-hoc which modules would have been included in the fix-cache, based on comparing the most fault prone modules in two time periods.

Among modules with at least one fix, the top 20% with the most fixes were selected in each of a three-month period. The share of modules common to the top 20% in the two periods was 66%. A repeated measures ANOVA was conducted on all modules with fixes, with the number of fixes in each of the three-month periods as the dependent variable. The test indicated that the fault distributions were different in the two time periods ($p < .003$), hence the assumption is supported.

Fault location: The other basic assumption is that test cases find faults in the same *fault location*, which sets the upper limit for the method's effectiveness. To test this assumption we analyzed whether test cases, which have failed more than once, lead to fixes in the same modules.

We observed a small number of test cases where fails lead to more than one defect report, causing fixes in the software. 27% of these test cases lead to fixes in the same modules, while the remaining 73% of the test cases lead to fixes in different modules each time. We consider the assumption weakly supported by the studied test cases.

IV. Threats to validity

We analyze threats to the validity of the study and report countermeasures taken to reduce them. The definitions follow Wohlin *et al.* [13].

Conclusion validity is concerned with the statistical analyses underpinning the conclusions. The statistical analyses use the robust t-test and in the checking of assumptions, ANOVA. In the sensitivity analysis, we repeat the t-test for each variant, but towards the same reference. Hence, the error rate problem is not apparent.

Internal validity is about the risk for other factors impacting on the relation between what is manipulated and the outcome. The limitation here is that the original set of test cases as well as the small share of selected test cases may not be representative. However, the sensitivity analysis indicates that the conclusions are robust.

Construct validity is concerned with the alignment between what is measured and what is the underlying construct. The test case efficiency measure is only one view of a good regression test selection procedure. The overall defect detection effectiveness is even more important, i.e. the share of defects detected by different test case selection methods. The available data in this case did not allow us to perform such an analysis. Two assumptions for the method was analyzed in Section III-D and found supported and weakly supported, respectively, although on a small number of test cases.

External validity is related to generalizability of the results. We have no indications that this environment is unique, but the method should of course be evaluated and tailored to other environments before launching it widely. Its underlying assumptions of Pareto distributed faults is verified by other research, e.g. [1], [5], [6] but the dynamic variation over time is not verified in those studies.

V. DISCUSSION AND FUTURE WORK

The fix-cache regression test selection technique is a simple, but apparently efficient technique for test case selection. It makes use of information that already is collected and stored in different databases. Setting it into use involves mainly connecting these databases together.

Our empirical evaluation indicates that the technique is more efficient than the previously used technique. The set of test cases that were selected and executed found significantly more defects per test case than the previously used approach did.

Still, there are many questions remaining open. One major question is whether the defect detection effectiveness is better as well. The technique selected a small set of test cases, so the *number* of faults found is very small compared to the number selected by the manual method.

The size of the cache is a factor that impacts on the number of selected test cases. Future evaluations include varying the cache size, and evaluating the efficiency for various sizes of the cache. They should also include data collection to enable analysis of effectiveness measures such as precision and inclusiveness [9]. Other pragmatic strategies, such as random selection of a given percentage should also be applied as a reference.

Replications in other companies and settings are also encouraged to increase the knowledge of the fix-cache regression test selection method.

References

[1] C. Andersson and P. Runeson. A replicated quantitative analysis of fault distributions in complex software systems. *IEEE Transactions on Software Engineering*, 33(5):273, 2007.

[2] V. R. Basili and B. T. Perricone. Software errors and complexity: An empirical investigation. *Communications of the ACM*, 27(1):42–53, 1984.

[3] P. K. Chittimalli and M. J. Harrold. Recomputing coverage information to assist regression testing. *IEEE Transactions on Software Engineering*, 35(4):452–469, 2009.

[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] N. E. Fenton and N. Ohlsson. Quantitative analysis of faults and failures in a complex software system. *IEEE Transactions on Software Engineering*, 26(8):797–814, 2000.

[6] M. Hamill and K. Goseva-Popstojanova. Common trends in software fault and failure data. *IEEE Transactions on Software Engineering*, 35(4):484–496, 2009.

[7] S. Kim, T. Zimmermann, E. Whitehead, and A. Zeller. Predicting faults from cached history. *29th International Conference on Software Engineering (ICSE'07)*, pages 489–498, 2007.

[8] A. Koru, D. Zhang, K. El Emam, and H. Liu. An investigation into the functional form of the size-defect relationship for software modules. *IEEE Transactions on Software Engineering*, 35(2):293–304, 2009.

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

[10] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009.

[11] M. Sherriff, M. Lake, and L. Williams. Prioritization of regression tests using singular value decomposition with empirical change records. *The 18th IEEE International Symposium on Software Reliability (ISSRE '07)*, pages 81–90, 2007.

[12] G. Wikstrand, R. Feldt, J. Gorantla, W. Zhe, and C. White. Dynamic regression test selection based on a file cache an industrial evaluation. *International Conference on Software Testing Verification and Validation (ICST '09)*, pages 299–302, 2009.

[13] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering: an introduction*. Kluwer, 2000.