Prioritizing Requirements-Based Regression Test Cases: A Goal-Driven Practice

Mazeiar Salehie, Sen Li, Ladan Tahvildari
Software Technologies Applied Research (STAR) Lab
University of Waterloo
Waterloo, Canada
{*msalehie,s35li,ltahvild*}*@uwaterloo.ca*

Rozita Dara, Shimin Li, Mark Moore
BIS-E Software Verification & Validation Department
Research In Motion (RIM)
Waterloo, Canada
{*rdara,shili,mamoore2*}*@rim.com*

***Abstract*—Any changes for maintenance or evolution purposes may break existing working features, or may violate the requirements established in the previous software releases. Regression testing is essential to avoid these problems, but it may be ended up with executing many time-consuming test cases. This paper tries to address prioritizing requirements-based regression test cases. To this end, system-level testing is focused on two practical issues in industrial environments: i) addressing multiple goals regarding quality, cost and effort in a project, and ii) using non-code metrics due to the lack of detailed code metrics in some situations. This paper reports a goal-driven practice at Research In Motion (RIM) towards prioritizing requirements-based test cases regarding these issues. Goal-Question-Metric (GQM) is adopted in identifying metrics for prioritization. Two sample goals are discussed to demonstrate the approach: *detecting bugs earlier* and *maintaining testing effort*. We use two releases of a prototype Web-based email client to conduct a set of experiments based on the two mentioned goals. Finally, we discuss lessons learned from applying the goal-driven approach and experiments, and we propose few directions for future research.**

***Keywords*-Test case prioritization; Software regression testing; Requirements-based test cases; Goal-driven approaches**

I. Introduction

Regression testing is performed to avoid unwanted side-effects of changes in a new release. Onoma *et al.* list several scenarios where regression testing is essential [1], such as developing a product family, maintaining large programs over a long period of time, and evolving a rapidly-changing product. This paper focuses on the system-level requirements-based testing and tries to address some challenges related to regression testing in this phase.

This research work is the result of collaboration between the University of Waterloo STAR Lab and the RIM Software V&V group for BlackBerry Internet services. Two practical issues in test case prioritization (TCP) for regression testing motivated us to do this research. First, TCP for regression features and requirements can be performed to achieve different goals, such as reducing effort and maintaining the current product quality. Although some researchers noted the key role of goals (e.g., [2]), goal-driven TCP has not been systematically addressed yet. We use GQM for identifying non-code metrics that can help us achieving regression testing goals.

Second, in some practical situations, code-based metrics (e.g., the code coverage for each single test case) are not available or is costly to collect. One of the reasons is that functional test cases are not always fully-automated due to time constraints or other management issues. Also, test cases may be run on shared distributed environments by a number of testers simultaneously and collecting code coverage data may not be straightforward for each test case.

We target two research questions: *RQ1*: how goals can direct us to identify appropriate metric(s) for prioritization in requirements-based regression testing? *RQ2*: how non-code information can help us prioritize test cases? We conduct a set of experiments on an email client to practice GQM in identifying non-code metrics for TCP, and to investigate the usefulness of the metrics.

II. Related Work

Several test suite optimization approaches have been discussed for regression testing improvement. These approaches generally include [3]: i) selection, ii) minimization, and iii) prioritization. The existing research efforts indicate that prioritization is the safest approach in terms of the number of escaped bugs from the testing process. If it is possible to define a test adequacy criteria to be evaluated during the testing process, we will be able to stop testing after executing a subset of test cases (converting prioritization to selection). Goals in regression testing can be of different kinds. For example, Rothermel *et al.* noted revealing bugs or high-risk bugs earlier as potential goals in testing [2]. Other goals related to cost and effort can also be articulated in a project.

Although regression testing has drawn a considerable amount of attention from researchers and practitioners, to the best of our knowledge, there are only few works that discuss selection techniques for regression testing based on non-code metrics. For example, a selection technique is discussed by Chen *et al.* which is based on the modification of a system's activity diagram, a notation of the UML [4]. The ideas proposed in these papers are interesting and innovative, but all of them are selection techniques and are not requirements-based. To the best of our knowledge, prioritization techniques for requirements-based regression test cases have not been extensively addressed yet.

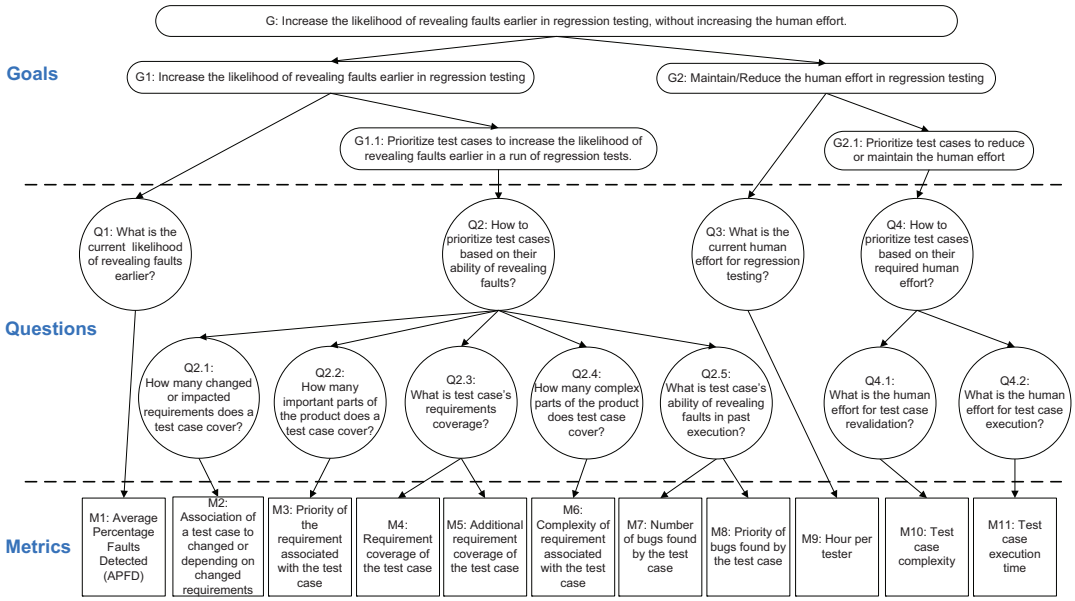

Figure 1. GQM Model for the Example Goals

Several research works suggested related ideas to the research theme of this paper. Onoma *et al.* raised some issues about regression testing in an industrial environment, such as regression testing process in practice and regression testing process cost analysis [1]. These issues are insightful and support our approach to take test execution cost into account in TCP. In another work, Qu *et al.* presented a set of algorithms that can dynamically assign priority to each test case based on the set of previously found bugs and execution time [5]. Moreover, Srikanth *et al.* tried to prioritize requirements-based test cases using a utility value computed by four metrics: customer-assigned priority, requirements complexity, requirements volatility, and fault proneness of the requirements [6]. None of these works proposed a systematic way to identify non-code metrics regarding their impacts on project goals.

III. Using Goals in Identifying Metrics for TCP

A three-level model is defined by GQM for metrics derivation [7], including goals, questions, and metrics. The goal and metric levels match to what is needed for TCP. However, questions in GQM are posed for evaluating goals not achieving them. The flexibility of the question level lets us adding a hierarchy of questions. By considering some testing heuristics, we can come up with a set of questions regarding prioritization. For example, testing more complex or frequently-changing artifacts can be considered. We also thought about adding prioritization goals to the first level. But, this led to augmenting insignificant goals for managers.

To show how we use GQM in identifying prioritization metrics, we present an example. As shown in Figure 1, the goal "*increase the likelihood of revealing faults earlier in regression testing, without increasing the human effort*" is set for regression testing. This high-level goal can be broken into two sub-goals. The first one (G1) is to increase the likelihood of finding bugs earlier than the current situation. The other one (G2) is to reduce or maintain the current testing effort. Questions Q1 and Q3 are designed to derive the measurement metrics for evaluating G1 and G2. Average Percentage Faults Detected (APFD) (M1) is a well-known metric used to measure the rate of fault detection [2], and tester per hour (or hour per tester) (M9) is a common metric for human effort. Moreover, to attain G1 and G2, we can apply prioritization techniques to specify test case execution order. Two sub-sub-goals (G1.1 and G2.1) are added respectively to G1 and G2 for this purpose. Two questions (Q2 and Q4) are posed to specify what we need to employ prioritization.

For Q2, five sub-questions are raised to characterize G1.1 from different perspectives in the regression testing:

- Q2.1: A regression test case may fail when existing features or requirements change, or when new features impact the existing ones. For non-code metrics, the question is related to changes in requirements and feature dependencies (M2).
- Q2.2: Commonly-used features are usually modified and updated, and the code corresponding to those features might probably be defective. The requirements for the important features are normally assigned higher priority (M3).
- Q2.3: A test case with higher requirements coverage (M4) might increase the likelihood of revealing more faults. The additional requirement coverage (M5) is also important. In this case, a test case with the highest requirements coverage is selected and, then, the rest of test cases will be ranked according to the additional requirements each one covers.

- Q2.4: A common testing heuristic is that more complex code is more likely to contain bugs. Thus, we can execute test cases that cover more complex requirements earlier. For each requirements specification, we may use a complexity measure (M6) as how difficult developers can implement it. M6 can be estimated by developers and it may be combined with the number of function points, constraints or preconditions in each specification.
- Q2.5: A test case ability to reveal faults in the past may reflect its ability to reveal faults in the future [5]. There could be two metrics relating to the bug history of a test case: the number of found bugs (M7) and their priority (M8).

To attain G2.1, test cases that require less effort may be executed earlier. The tester effort for regression testing is captured by two questions Q4.1 and Q4.2 based on two aspects: the effort to revalidate test cases and the effort to execute test cases [1]. Generally, the revalidation effort highly depends on the test case complexity (M10), either assigned by a tester or based on test case analysis. The test case execution time (M11) includes both the execution time, and the comparison time with the expected results.

IV. Conducted Experiments

A. Experiment Setup

Due to limitations on publishing RIM confidential data, we implemented a prototype Java EE Web-based email client as the system under test. Two consecutive versions of this application were selected for conducting experiments. The second version contained some changes to fix several bugs and to introduce several enhancements to the attachment service. We generated functional system test cases based on requirements using the Selenium framework (available at http://seleniumhq.org). Some of the test cases were semi-automated.

The goals G1 and G2 in Figure 1 are considered for the experiments. Table I lists the treatments used for TCP. The first six treatments are towards achieving G1 without considering cost. Optimal (T1) is the control treatment that is the best possible test case order to find bugs early. This order is found manually by the tester. Treatments T7 to T11 are cost-aware, and the Optimal* (T7) is the optimal cost-aware order. For these treatments, there may be similar values for clusters of test cases (e.g., those with similar requirements coverage). Therefore, while there are *best* and *worst* cases for each treatment, only worst cases are considered in experiments. As shown in Table I, a subset of applicable metrics derived in Figure 1 was selected to achieve G1 and G2. For treatments with more than one metric, multi-level prioritization was applied. In these cases, the treatments were named based on the prioritization order. For example in T5, test cases are prioritized first by ChangeDep value and then by Imp. Since there were a few semi-automated test cases, only one tester executed the entire test suite. In addition, the cost metric in the experiments equals to the test execution time.

Table I
Treatments Used in Experiments (Metrics from Figure 1)

Treatment	Metric(s)
Optimal (T1)	-
ChangeDep (T2)	Req. change/dependency (M2)
Imp (T3)	Req. importance (M3)
Comp (T4)	Req. complexity (M6)
ChangeDep-Imp (T5)	Req. change/dependency (M2) & req. importance (M3)
ChangeDep-Comp (T6)	Req. change/dependency (M2) & req. complexity (M6)
Optimal* (T7)	-
ChangeDep-Cost (T8)	Req. change/dependency (M2) & test case execution time (M11)
ChangeDep-Imp-Cost (T9)	Req. changes/dependency (M2), req. importance (M3) and test case execution time (M11)
ChangeDep-Comp-Cost (T10)	Req. change/dependency (M2), req. complexity (M6) & test case execution time (M11)
ChangeDep-Imp-Comp-Cost (T11)	Req. changes/dependency (M2), req. importance (M3), req. complexity (M6) and test case execution time (M11)

B. Obtained Results

The first set of experiments considered only G1. We measured the detected fault percentage in terms of the executed test suite percentage for treatments T1 to T6. T2, T5 and T6 are closer to the optimal ordering (T1) than the others. The APFD, calculated based on the area under the curve, is larger for ChangeDep-Imp (T5), that means T5 satisfies G1 more than T2 and T6.

Some test cases were semi-automated and several preconditions were needed to be satisfied before running them. The total time was considered as cost for those cases. To investigate the cost effect, we ran the second set of experiments with cost-awareness based on T7 to T11. Among the first six treatments (no cost-awareness), T1 is the best and ChangeDep-Imp is the second best in terms of the cost. Therefore, ChageDep-Imp still outperforms the other treatments while its accumulated cost is 56.2%. Overall, between all eleven treatments after Optimal and Optimal*, ChangeDep-Imp-Cost (T9) is the best considering G2. The accumulative cost for T9 is 50%, which is better than ChangeDep-Imp. Now, the question is that which one has better APFD.

Figure 2 shows the detected fault percentage for all the treatments. Interestingly, ChangeDep-Cost (T8) has the best APFD after Optimal and Optimal*, while it has 56.2% accumulative cost for finding all the bugs. The next treatment in Figure 2 is ChangeDep-Imp-Cost (T9) that was the best in terms of cost. By calculating APFD, we observe that after Optimal and Optimal* by 80.77%, T8 has the best APFD value of 75.64%. This is about 1.4% better than T9, while T9 has about 6% advantage in cost. If G1 and G2 are both equally important for us, T9 is the best one at achieving both goals in regression testing of the email client.

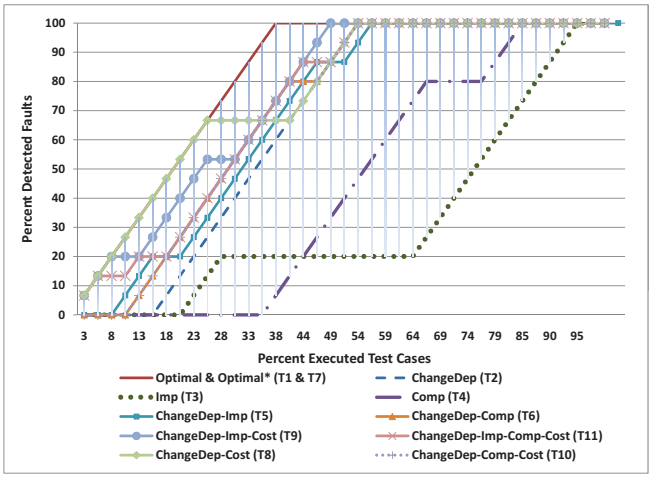

Figure 2. Detected Fault Percentage (Worst Case) by Considering Cost

V. Lessons Learned and Future Works

This paper tries to address two motivating research questions in TCP. Regarding RQ1, the adopted goal-driven approach helped us systematically look for metrics positively impacting the designated goals. While GQM is normally used in quality measurement to identify appropriate metrics, we employed it to identify metrics for attaining goals in regression testing. This would be particularly helpful in case of having several goals in industrial projects. In the experiments, the specified goals helped us think about different aspects of each goal. Moreover, we involved the question of how to achieve goals via prioritization. We used some implicit testing heuristics to formulate questions for this purpose. Although for this small example writing questions were not difficult, we believe the heuristics should be explicitly defined in a catalog to facilitate formulating questions. For example, heuristics referring the relation between complexity, volatility, previously defective artifacts, and fault-proneness are helpful in writing questions for G1.1.

The second question, RQ2, deals with how prioritization is possible when code-based data is not available. The obtained results suggest that non-code metrics are able to generate promising results. This effect was observed in both with and without cost-awareness. However, we faced some challenges in collecting non-code data. For requirements, we had to deal with documents written in natural language text. We used some text mining tools to investigate these documents and we also consulted with the developers to have an estimate of requirements-based metrics. But, the gathered data still seems less reliable than code-based data.

We were not surprised to observe treatments including change-based metrics outperformed the other ones. Rothermel *et al.* [2] pointed out that the majority of faults are covered by modification-traversing test cases based on code coverage. The experiments endorsed this effect for non-code data and ChangeDep was the most impacting metric. However, the point is that analyzing requirements and feature dependency is not always straightforward.

We used multi-level prioritization, but it is also possible to apply a weighted sum of metrics, as discussed in [6]. After consulting with a few testers at RIM, we came to the conclusion that a multi-level approach is more understandable for practitioners, even though the latter method might lead to better results. Another plus point for the first method is that if we do not have complete measures for some metrics, we can still use them in some clusters of test cases. For example, assume that we have a set of test cases with equal requirements coverage and have Imp values for this set, while for the others data is not complete. We can still use Imp to prioritize test cases inside this set, while in the weighted sum method, we might need different weights for each set.

Several directions can be followed in this research. We can add more non-code metrics for TCP, e.g. based on bug history. In TCP with code-based data, a higher fault-proneness probability can be assigned to modules with history of defects. We may also be able to use a similar strategy based on links between bugs and requirements. Another potential direction, seems promising, is to combine code and non-code metrics. At least for some test cases, especially automated ones, we may have the code data. Because during testing the code can be changed for bug fixing, we might need to re-prioritize test cases. In this case, we need a dynamic TCP that cannot be evaluated with a single final APFD or cost value.

References

[1] A. Onoma, W. Tsai, M. Poonawala, and H. Suganuma, "Regression testing in an industrial environment," *Communications of the ACM*, vol. 41, no. 5, pp. 81–86, 1998.

[2] G. Rothermel, R. Untch, C. Chu, and M. Harrold, "Prioritizing test cases for regression testing," *IEEE Trans. on Software Engineering*, vol. 25, no. 5, pp. 929–948, 2001.

[3] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software Testing, Verification and Reliability*, vol. 20, no. 2, 2010.

[4] Y. Chen, R. Probert, and D. Sims, "Specification-based regression test selection with risk analysis," in *Proc. of the Conf. of the IBM CAS (CASCON)*, 2002, pp. 1–14.

[5] B. Qu, C. Nie, B. Xu, and X. Zhang, "Test case prioritization for black box testing," in *Proc. of the Int. Computer Software and Applications Conf.*, 2007, pp. 465–474.

[6] H. Srikanth, L. Williams, and J. Osborne, "System test case prioritization of new and regression test cases," in *Proc. of the Int. Symp. on Empirical Software Eng.*, 2005, pp. 10–19.

[7] V. Basili, G. Caldiera, and H. Rombach, "The goal question metric approach," *Encyclopedia of Software Engineering*, vol. 1, pp. 528–537, 1994.