Dynamic Prioritization in Regression Testing

Nilam Kaushik*, Mazeiar Salehie*†, Ladan Tahvildari*, Sen Li‡ and Mark Moore‡
*Department of Electrical and Computer Engineering, University of Waterloo
† Lero - The Irish Software Engineering Research Center, Limerick, Ireland
‡ Research In Motion, Canada

***Abstract*—Although used extensively in industry, regression testing is challenging from both a process management as well as a resource management perspective. In literature, proposed test case prioritization techniques assume a constant pool of test cases with non-changing coverage during the regression testing process, and therefore they work with a fixed, prioritized test suite. However, in practice, test cases and their coverage metrics may change during regression testing due to modifications of software artefacts (e.g. due to bug fixing). For example, modifying obsolete test cases or source code may change the coverage metrics during the process. This may lead to some changes in test case priorities. Dealing with manual tests cases, scheduling test case execution in shared environments and other constraints in practice may cause the same effect. In this paper, we highlight these challenges in industrial regression testing and propose a paradigm called Dynamic Prioritization, which uses in-process events and the most up-to-date test suite to re-order test cases.**

***Keywords*-regression testing, test case prioritization, software testing**

I. Introduction and Motivation

Regression testing is the process of validating modified software to assure that changed parts of software behave as intended and unchanged parts of software have not been adversely affected by the modification [1]. Studies show that regression testing accounts for 80% of the testing costs [2]. Shifts in software development practices towards component based software development and agile development impose constraints on regression testing[3], giving rise to approaches that minimize the cost of regression testing. The three main approaches to reduce the cost of regression testing include test case selection, test suite minimization and test case prioritization.

While test case selection techniques select a subset of existing test cases, test case minimization techniques reduce the test suite size to a minimal subset to maintain the same level of coverage as the original test suite. On the other hand, test case prioritization techniques concern with identifying an ideal order of test cases according to some criteria, such that test cases with higher priority are executed earlier than ones with lower priority. There is empirical evidence indicating that fault detection capabilities of test suites can be severely compromised by minimization [4]. However, test case prioritization does not suffer from such drawbacks as it does not discard test cases, and it can be used in conjunction with other two techniques. A recent survey on regression testing shows an increasing interest in test case prioritization since the late 90s [3]. Due to the apparent advantages of test case prioritization over the other techniques, we focus on prioritization in this research work.

While regression testing has been studied widely in research, putting the proposed techniques into practice has been a challenge. Onoma et al. [5] report that while regression testing is used extensively in industry, re-test all is still a commonly used approach. Engstrom and Runeson point out that there is a clear gap between research and industrial practices in regression testing, as in practice there is no systematic approach for test case selection or prioritization [2]. They also mention that the focus in industry is more on automation to decrease the cost and effort of regression testing. Rooksby et al. [6] argue for the need to investigate and characterize the real-world work of testing.

In literature, proposed test case prioritization techniques often assume a fixed test suite and constant test case coverage metrics(mainly code-based) during regression testing. However, in practice, test cases and their corresponding metrics may change during regression testing due to modifications of software artefacts. For example modifying obsolete test cases or source code (e.g. due to bug fixing) may change the coverage metrics during the process. This may lead to some changes in test case priorities. Although these issues are common to both manual and automated test cases, manual test cases may add other challenges to regression testing. Distributing test cases among testers, executing them in a shared environment and the difficulty in obtaining some coverage metrics impact prioritization. In our previous paper [7], we dealt with regression testing with manual test cases in the absence of code coverage metrics. In this paper, we highlight broader challenges in industrial regression testing, with a focus on test case prioritization. We propose a paradigm called Dynamic Prioritization that addresses a main challenge in this context.

II. Challenges in Industrial Regression testing

We categorize the challenges in industrial regression testing into three main groups- environment and resource related, metrics related, and challenges related to in-situ changes during testing. We elaborate on these categories in this section.

A. In-situ changes

The regression test suite consists of those test cases that are developed for new features during the release in addition to those that already exist for regression features. Over the course of the development cycle, test cases are modified to accommodate requirement changes and maintain a certain level of test adequacy for regression features. Over time, some regression test cases may become obsolete, requiring revalidation. As test case revalidation is a largely manual activity, it is atypical and cumbersome for testers to revisit all the test cases in the regression pool and revalidate them. Instead, testers resort to revalidating test cases dynamically at the time of execution. Some other common problems include the presence of bad test cases and duplicate test cases [8]. Thus, the pool of test cases continuously changes over the development cycle. However, test case prioritization techniques proposed in literature normally assume a constant pool of test cases and a constant environment.

Furthermore, it is a common practice for project managers to facilitate a defect triage meeting in order to examine open defects and to prioritize and assign defects to concerned teams for further action. The triage team validates the severity of defects and assigns priorities. Due to limited resources and time, some defects may be deferred to another release. Triage is also a ground to evaluate the impact of a defect and its resolution on the testing and development schedule. From a testing standpoint, triage determines, to a certain extent, which test cases are to be run in order to minimize the impact of defect resolution on other components. An effective triage makes the best possible use of time and resources. The role of triage on test case prioritization has also not been taken into consideration in literature.

B. Metrics related challenges

This set of challenges pertains to the difficulty in obtaining metrics that are key to prioritization techniques.

1) Difficulty in obtaining code coverage: Most proposed regression testing techniques are code based. While code-based testing can be effective at a unit test level, obtaining code coverage can be a challenge for the system-level testing of large distributed systems, particularly when some test cases are manual and requirements-based. Moreover, code-based regression testing techniques require testers to understand the code to some degree. This can be a time consuming exercise [9]. Programming language dependency may also add to the complexity of code-coverage measurement as large software components incorporate more than one programming language.

2) Ambiguity in requirement coverage: Srikanth et. al propose a requirements-based test case prioritization scheme using requirement complexity, requirement volatility and customer-assigned requirement priority [10]. The underlying assumption of such an approach is that requirements are complete and formalized, which is often not the case in real practice. In industry, natural language is commonly used to capture product requirements. Due to the flexibility in the use of natural language, requirements can be ambiguous and prone to easy misinterpretation. An accurate interpretation of a requirement depends on the writer and reader's use of common terminology for the same concepts. It is not uncommon to come across situations where differences in interpretation of requirements can lead to end deliverables that do not behave as expected. Furthermore, it is difficult to modularize requirements as several smaller requirements may be grouped into a single all-encompassing requirement. This in effect makes the task of evaluating requirement complexity arduous.

C. Environment and resource related challenges

These challenges pertain to constraints within the testing environment and resources.

1) Testing execution and environment constraints: Software testing may involve manual as well as automated test cases. In the case of manual testing, the testing environments may be shared by multiple testers, making test case execution dependent on the availability of resources. Existing techniques in literature assume that test cases can be executed in any given order without affecting the execution cost. This is not very practical as test cases may have dependencies among them or clusters of test cases may be managed by different testers. Instead, it logistically makes sense to group test cases that have similar pre-conditions or are related to the same features, thereby potentially saving on the context-switch cost and preserving dependency relations among test cases as well. Adding to these issues is the human factor involving the expertise of testers and their availability, which can impact the distribution of test cases among testers. While grouping and distributing test cases maybe easy to perform manually by experienced testers, a prioritization technique has to explicitly take the aforementioned constraints into consideration.

2) Tools: In industry, tools are incorporated to manage various artefacts of the development process such as defects, test cases, requirements, code, etc. Although neglected by research methods, tools are an integral aspect of the software development process. Like processes and resources, tools need to be selected with due diligence to match the needs of the projects in the organization. Tools should have the potential for adaptability and extensibility. As organizations evolve, existing tools are replaced by new tools to increase productivity, performance and scalability. Seamless integration across tools is crucial in obtaining the data necessary to build traceability links between artefacts, particularly the ones that are essential for regression testing. Good tools can enforce users to populate certain fields, leaving little room for incomplete data. Research methods work under the premise that any data is readily available and accurate, which is contrary to real practice.

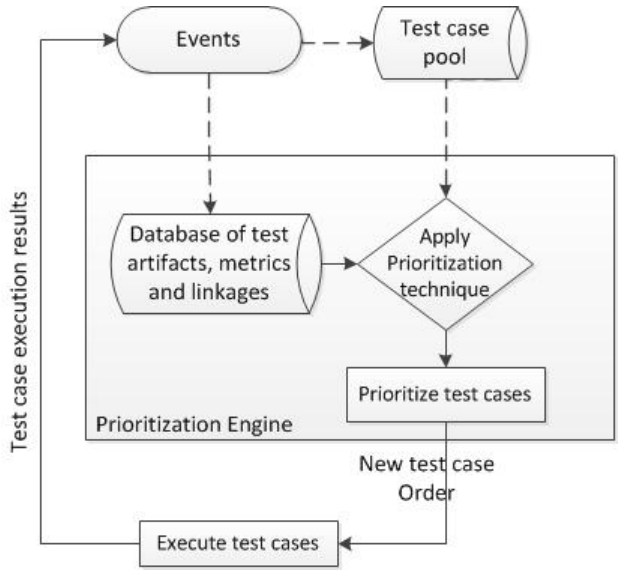

Figure 1: Process flow of Dynamic Prioritization in Regression testing

Artifact	Attribute	Related Metrics and linkage
Test case	test case creation data execution history execution status	Test case to Test case linkage Test case execution history Test case priority Test case dependency
Defect	priority severity defect status	Defect and test case linkage Defect and code linkage Severity of bugs Number of revealed bugs
Requirement	priority customer importance Requirement type	Requirement and defect linkage Requirement complexity Requirement change Requirement importance Requirement coverage per test case Additional requirement coverage Requirement dependency
Code	defects fixed feature covered	Code and test case linkage Code complexity Code change Total code coverage per test case Additional code coverage per test case

Table I: Testing artefacts and their respective attributes, metrics and linkage

III. Dynamic Prioritization in Regression testing

While most of the challenges discussed above are known, the issues posed by the in-situ changes during the testing process have not been addressed previously, to the extent of our knowledge. In order to overcome this challenge, we propose a paradigm called Dynamic Prioritization which involves changing the order of test cases during the testing process. Since the test case pool changes through the development cycle, the list of prioritized test cases would change as well. In dynamic prioritization, we take advantage of internal feedback such as triage and in-process events to prioritize test cases. Figure 1 illustrates a high level process flow of dynamic prioritization in regression testing.

A. High level process flow

1) Database of test artefacts, metrics and linkages: The artefacts relevant to the software testing process are listed in Table 1. Each artefact has some metrics that can be collected and used for prioritization, including: i) test case metrics include test case execution history, test case complexity and dependency, ii) bug metrics include severity and number of revealed defects, iii) requirements metrics include requirement complexity, requirement change, requirement coverage, and iv) code metrics include code complexity e.g. McCabe's Cyclomatic complexity, code change and total code coverage. Since test suite optimization techniques are data-intensive, rich data and traceability links between these artefacts are key for a prioritization technique to be effective.

2) Prioritization technique: Various prioritization techniques have been proposed in literature, the major ones include code-based techniques such as [11], requirements-based techniques such as [10] and [12], cost effective-based techniques such as [13] and chronographic history-based techniques such as [14]. For the purposes of this paper we assume that a suitable prioritization technique for the domain at hand is known. Availability of test artefacts and linkages data notwithstanding, several prioritization techniques may be combined to form one amalgamated technique.

3) Events: An event in this process is any change that can affect the prioritization metrics discussed above. Events are incidents such as requirement change/removal, code change, defect fix, and test case addition/removal/update. A frequent event that makes sense to use for dynamic prioritization is the occurrence of a new software build during regression testing. Such an event can provide information about code changes and defects.

4) Test case pool: As discussed in section II A, the pool of test cases is dynamic as it is continuously updated during the testing process. The pool of test cases is an input to the Prioritization Engine.

5) Prioritization Engine: The prioritization engine comprises of the the database of artefacts, linkages and metrics, and the prioritization technique. It is triggered on an event notification and applies the prioritization technique on the latest pool of test cases, outputting a list of prioritized test cases.

B. An example scenario

We demonstrate the idea of Dynamic Prioritization with a sample scenario. Assume we have the following setting:
Initial test suite: A, B, C, D, E, F, G, H, I
Initial test case order: A, B, C, D, E, F, G, H, I
Test cases executed so far: A, B, C

Then these two events occur:

Event 1: deletion of obsolete test case F
Event 2: a code change (bug-fix)

Consequently these changes will be applied to the test suite and the database in the prioritization engine will: i) Remove all linkages associated with test case F due to Event 1, and ii) Update linkages and metrics related to the code change (causes update on coverage metrics of test cases E and I) due to Event 2. As a result, we will have an updated setting:

Updated test suite: D, E, G, H, I
New test case order: I, E, D, G, H

It is evident from this example that the latest test suite includes those test cases from the old prioritized test case order that were not executed (test case D, E, G, H and I) and excludes any obsolete test cases (test case F) and test cases that were already executed (test case A, B and C). Moreover, the new prioritized test case order re-orders the new test case pool based on the code change in Event 2, through which a linkage to test case E and I is established.

IV. Conclusion and future work

Most prioritization techniques proposed in literature assume a fixed pool of test cases and the availability of coverage metrics for all test cases. However, in industrial projects test cases and other system artefacts(mainly source code) may change, while some metrics may not be easily available (e.g., code coverage), and resource/tool constraints may alter during regression testing process. These issues can make the initial test case priorities useless. We have categorized these challenges into three classes: environment and resource related, metrics related, and challenges related to in-situ changes.

In this paper we address the challenge posed by in-situ changes during the testing process, the less addressed problem in industrial regression testing. We introduce the idea of Dynamic Prioritization in regression testing which uses in-process events to re-order test cases. Dynamic Prioritization uses the most up-to-date pool of test cases and generates a new test case order based on in-process events.

Designing and implementing a prototype of Dynamic Prioritization in a pilot project is top priority in our future work list. However, we need to also address the other challenges, even partially, to make sure we have rich enough data and to ensure we deal with tools and environment constraints properly. We have already started working on those issues [7], and we plan to put together what we learned in employing Dynamic Prioritization.

References

[1] M. J. Harold, J. A. Jone, T. Li, and D. Liang, "Regression test selection for java software," in *Proc. of the ACM Conference on OO Programming, Systems, Languages, and Applications*, 2001.

[2] E. Engstrm and P. Runeson, "A qualitative survey of regression testing practices," in *Proc. of the International Conference on Product-Focused Software Process Improvement*, 2010, pp. 3–16.

[3] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," in *Software Testing, Verification and Reliability*, 2010. [Online]. Available: http://dx.doi.org/10.1002/stvr.430

[4] G. Rothermel, M. J. Harrold, J. Ostrin, and C. Hong, "An empirical study of the effects of minimization on the fault detection capabilities of test suites," in *Proc. of the International Conference on Software Maintenance*, 1998, pp. 34–43.

[5] K. Onoma, W.-T. Tsai, M. Poonawala, and H. Suganuma, "Regression testing in an industrial environment," in *Proc. of Communications of the ACM*, vol. 41, 1998, pp. 81–86.

[6] J. Rooksby, M. Rouncefield, and I. Sommerville, "Testing in the wild: The social and organisational dimensions of real world practice," in *Computer Supported Cooperative Work (CSCW)*, vol. 18(5), 2009, pp. 559–580.

[7] M. Salehie, S. Li, L. Tahvildari, R. Dara, S. Li, and M. Moore, "Prioritizing requirements-based regression test cases: A goal-driven practice," in *Industrial Track of European Conference on Software Maintenance and Reengineering (CSMR)*, 2011, p. TBA.

[8] C. Jones, "Software quality in 2008: A survey of the state of the art," White Paper, Software Productivity Research, 2008.

[9] Y. Chen, R. Probert, and D. P. Sims, "Specification-based regression test selection with risk analysis," in *Proc. of the Conference of the Centre for Advanced Studies on Collaborative research*, 2002.

[10] H. Srikanth, "Requirements-based test case prioritization," in *Doctoral Symposium in International Conference of Software Engineering*, St. Louis, MO, 2005.

[11] D. Leon and A. Podgurski, "A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases," in *Proc. of the International Symp. Software Reliability Eng*, 2003, pp. 442–453.

[12] X. Zhang, C. Nie, B. Xu, and B. Qu, "Test case prioritization based on varying testing requirement priorities and test case costs," in *Proc. of the Seventh International Conference on Quality Software*, 2007.

[13] A. G. Malishevsky, J. R. Ruthruff, G. Rothermel, and S. Elbaum, "Cost-cognizant test case prioritization," in *Technical Report TRUNL-CSE-2006-0004, Department of Computer Science and Engineering, University of Nebraska*, 2006.

[14] J. Kim and A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments," in *Proc. of the International Conference on Software Engineering (ICSE)*, 2002, p. 119129.