ISSN(Online):2320-9801
ISSN(Print): 2320- 9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Special Issue 3, July 2014

Improving Quality of Software Testing Process by Test Case Prioritization

S.K.HariKarthik, Dr.V.Palanisamy

Assistant Professor, Dept. of Information Technology, INFO Institute of Engineering, Coimbatore, India

Principal, INFO Institute of Engineering, Coimbatore, India

ABSTRACT:A standard part of software development involves creating serious of test cases, which tests a feature of the system. Regression testing is an expansive, but important process in a software development scenario. Unfortunately, there may be insufficient resources to allow for the re-execution of all test cases during regression testing. In this situation, test case prioritization techniques aim to improve the effectiveness of regression testing by ordering the test cases so that the most beneficial are executed first. For prioritization of test cases different algorithm are used.

KEYWORDS: Searching techniques, regression testing, Test Case prioritization

I. INTRODUCTION

Software testing is a methodology to find the bugs in the software (or) an activity being part of the software development process aimed at evaluating a software item (system, subsystem, unit etc.) features functionality, performance etc against the given set of system requirements. The testing can be broadly divided into two forms namely i) verification testing ii) validation testing.

Verification testing is the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of the phase. Verification testing is used for requirements verification, design verification and code verification.
Validation testing on the other hand involves execution of final end product. It is the process evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.

In a software development scenario many test cases are executed repeatedly and causes problem in resource constraint environments [7]. In order to avoid this repeatedly executing test cases, prioritization techniques has been applied. Test case prioritization involves selecting a manageable number of tests to use from a large, existing test suite consisting of synthetic test cases or captured operational inputs. A subset of the test suite is selected based on an assessment of how likely it is to reveal any latent defects in the program under test. This assessment involves analysis of profiles of test executions. Various kinds of profiles can be used for this purpose, such as ones reflecting control flow, data flow, input or variable values, object states, event sequences, and timing.

Regression testing- Regression testing is an expensive and frequently executed maintenance process used to revalidate modified software. In earlier days, prioritization technique was not employed in the software development scenarios (especially in regression testing under dynamic environments) as a regular activity. Due to this, two critical limitations appears in regression testing namely, i) it models regression testing as a one-time activity rather than as the continuous process ii) it does not take real world time and resource constraint into considerations.

In the proposed system, prioritization technique was employed to prioritization the repeatedly executing test cases. Test case prioritization involves selecting a manageable number of tests to use from a large, existing test suite

ISSN(Online):2320-9801
ISSN(Print): 2320- 9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Special Issue 3, July 2014

consisting of synthetic test cases or captured operational inputs. A subset of the test suite is selected based on an assessment of how likely it is to reveal any latent defects in the program under test. This assessment involves analysis of profiles of test executions. Various kinds of profiles can be used for this purpose, such as ones reflecting control flow, data flow, input or variable values, object states, event sequences, and timing. The main reasons for prioritization test cases are to test the test case which has high priority. This techniques were employed namely Coverage based and Distribution based technique.

Test case prioritization-

Test case prioritization can address a wide variety of objectives. For example, concerning coverage alone, testers might wish to schedule test cases in order to achieve code coverage at the fastest rate possible in the initial phase of regression testing to reach 100 percent coverage soonest or to ensure that the maximum possible coverage is achieved by some predetermined cut-off point. The ideal order would reveal faults soonest, but this cannot be determined in advance, so coverage often has to serve as the most readily available surrogate. In the Microsoft Developer Network (MSDN) library, the achievement of adequate coverage without wasting time is a primary consideration when conducting regression tests. Furthermore, several testing standards require branch adequate coverage, making the speedy achievement of coverage an important aspect of the regression testing process.

II. COVERAGE ALGORITHM

2.1 Greedy Algorithm

A Greedy Algorithm is a method of “next best” searching algorithm. First it takes the maximum weight of the element, the followed by the element with the second-highest weight, and so on. Greedy search seeks to minimize the estimated cost to reach some goal. It is simple approach, but in some situations where its results are of high quality, it is attractive because it is typically inexpensive both in implementation and execution time. Consider the example of statement coverage for a program containing p statements and a test suite containing k test cases. For the Greedy Algorithm, the statements covered by each test case should be counted first, which can be accomplished in O (p k) time; then, the test cases should be sorted according to the coverage. In the second step, quick sort can be used, thereby increasing the time complexity by O (k log k). Typically, p is greater than n, in which case, the cost of this prioritization is O (p k).

Table 1: Example – Test cases with statements

Test Case	**Statement**							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
A	+	+	+				+	+
B	+	+	+			+	+	+
C	+	+	+	+				
D					+	+	+	+

For the example in Table 1, test case B is selected first since it covers six statements, the maximum covered by a single test case. Test case A, which covers five statements, is selected next. Test cases C and D cover the same number of statements and so the Greedy Algorithm could return either B; A; C; D or B; A; D; C, depending upon the order in which test cases are considered.

2.2 Additional Greedy Algorithm

The “Additional” Greedy Algorithm is also a kind of Greedy Algorithm, but its approaches differ. It combines feedback from previous selections. It iteratively selects the maximum weight element of the problem from that part that has not already been consumed by previously selected elements.

ISSN(Online):2320-9801
ISSN(Print): 2320- 9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Special Issue 3, July 2014

Again, consider statement coverage: The Additional Greedy Algorithm requires coverage information to be updated for each unselected test case following the choice of a test case. Given a program containing p statements and a test suite containing k test cases, selecting a test case and readjusting coverage information has cost O (p k) and this selection and readjustment must be performed O (k) times. Therefore, the cost of the Additional Greedy Algorithm is O (p k^2). After 100 percent coverage has been achieved, there are possible remaining unprioritized test cases that cannot add additional coverage. These remaining test cases could be ordered using any algorithm, the remaining test cases were ordered by reapplying the same algorithm. After 100 percent coverage has been achieved, no further fitness improvement will be possible. For the example in Table 1, test case B is selected first (it covers six statements) leaving statements 4 and 5 uncovered. Test case A is skipped since it covers neither statement 4 nor statement 5. Test cases C and D cover statements 4 and 5, respectively, so each covers only one uncovered statement. Thus, Additional Greedy would return either B; C; D; A or B; D; C; A.

III. EXPERIMENT

Here experiment imported the sample application which is created by using the .cs which is used to find out the number of functions covered, statements covered and number of classes covered. According to that we have to prioritize the test suits. Here we used the Ncover testing tool which is used to provide the graphical representation of the coverage of statements.

Here system have to find out average percentage block coverage to measures the rate at which a prioritized test suite covers the blocks. Then we have to find out the Average Percentage Decision Coverage to measures the rate at which a prioritized test suite covers the decisions (branches). Then we have to find out the Average Percentage Statement Coverage to measures the rate at which a prioritized test suite covers the statements.

In this experiment we have ten test cases and the coverage of the ten test cases is given based on the greedy algorithm.

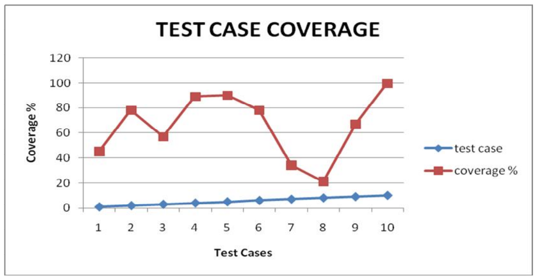

Figure 1: Test Case Coverage

Here the X- axis indicates the test cases and Y-axis indicates the percentage of coverage in the important source program. Here, the test cases 10 has the maximum coverage. So, while filtering, the test case which has the lease coverage will be filtered out.

ISSN(Online):2320-9801
ISSN(Print): 2320- 9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Special Issue 3, July 2014

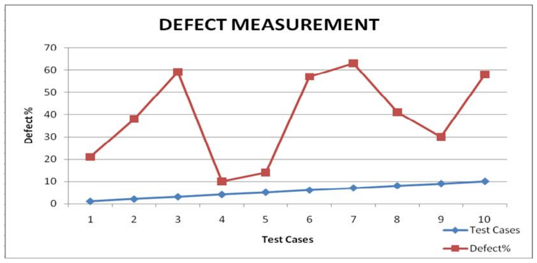

Figure 2: Defect Measurement

Here, the X-axis indicates the test cases and the Y-axis indicates the percentage of defects. The test case 7 has the maximum percentage of defects.

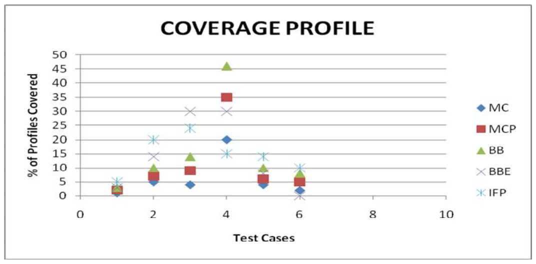

Figure 3: Coverage Profiles

Here X-axis shows the test cases and the Y-axis shows the count of profiles covered. The covered profiles are indicated by method call, method call pairs, Basic Blocks, Basic Block Edges and Information flow pairs.

IV. CONCLUSION AND FUTURE WORK

This paper deals with prioritization with the number of executing test cases by using coverage based. Coverage-based prioritization technique, select test cases to maximize the proportion of program elements of a given type (e.g., statements, branches, conditions, and loop) that are covered (executed). The coverage based technique makes use of greedy algorithm in order to prioritization out the repeatedly executing test cases. So, by prioritization with the number of executing test cases the overall quality of the testing process can be improved.

In the future, we can extended and implemented prioritization technique with distributed techniques. And the efficiency of greedy algorithm and additional greedy can be calculated.

REFERENCES

[1] W. Dickinson, D. Leon, and A. Podgurski, (2001) ‘Finding Failures by Analysis of Execution Profiles,’ Proc. 23rd Int’l Conf. Software Eng., pp. 339-348.
[2] W. Dickinson, D. Leon, and A. Podgurski, (2001) ‘Pursuing Failure: The Distribution of Program Failures in a Profile Space,’ Proc. 10th European Software Eng. Conf. /Ninth ACM SIGSOFT Symp. Foundations of Software Eng., pp. 246-255.
[3] S. Elbaum, A.G. Malishevsky, and G. Rothermel, (2002) ‘Test Case Prioritization: A Family of Empirical Studies,’ IEEE Trans. Software Eng., vol. 28, no. 2, pp. 159-182.
[4] D. Leon, W. Masri, and A. Podgurski, (2005) ‘An Empirical Evaluation of Test Case Filtering Techniques Based on Exercising Complex Information Flows,’ Proc. 27th Int’l Conf. Software Eng., pp. 412-421.
[5] Z. Harman, M. Hierons, (2007) ‘Search Algorithm for Regression Test Case Prioritization,’ IEEE Trans. Software Eng., vol. 33, no. 4, pp. 225-237.
[6] Heimdahl, M. & George, D. (2004), ‘Test-suite reduction for model based tests: effects on test quality and implications for testing’, Automated Software Engineering, 2004 Proceedings. 19th International Conference on, pp. 176-185.
[7] J.M. Kim and A. Porter, (2002) ‘A History-Based Test Prioritization Technique for Regression Testing in Resource Constrained Environments,’ Proc. Int’l Conf. Software Eng., pp. 119-129.