



Reusability and Test Case Prioritization Approach in Regression Testing

Jaspreet Singh¹, Shivani Sharma²

¹Assistant Professor, Department of Computer Science & Engineering
Chandigarh University, Gharuan, Punjab, India

²Research Scholar, Master of Engineering,
Department of Computer Science & Engineering
Chandigarh University, Gharuan, Punjab, India

Abstract: Software engineering deals with the development of software and different types of web applications. Due to changes in the customer requirements, product modifications have to be done in the web application so that an organization can interact with more customers. In order to validate the modifications, regression testing has to be performed again and again. The test cases of the modified versions have to be re-generated and testing has to be performed for each module. This process involves maximum time and effort for the generation of test cases. In the proposed work, variable and constraints have been reused from the previous version of the web application. These constraints and variable that has been reused provides less effort and time for the modification of test cases. The test cases can be generated only for the modifications on the basis of reusable constraints and variables. The testing process involves an algorithm that is used to compute the reusable constraints and variables for processing of the web application. In addition to it, prioritization of the test cases has been done on the basis of different coverage area of the web application.

Keywords: Prioritization, Reuse, Regression Testing, Test Case

I. Introduction

Software engineering is an engineering branch associated with the concept of software development process. It guides the programmer how to access the requirements from end users, how to design the algorithm before writing the code, test the program and prepare its documentations [1]. IEEE defines software Engineering: "It is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software"[2]. Software may be developed for a particular customer or may be developed for a general market. Software testing is one of the major and primary technique for achieving high quality software. It is the last phase of the product before delivery to the customer. Software testing is an important part which tells whether the product is efficient, error-free, works properly and according to the requirements of the customer or not[3]. Fig1.1 shows software testing. There are different types of testing.

- a) White-box testing: It is also known as Glass testing and structural testing. In white-box testing, code is visible. Testing team has knowledge about internal mechanism of the components.
- b) Black-box testing: It is also known as functional testing. It is a testing which is based upon the input & expected output and having no knowledge of the internal code [4].
- c) Unit testing is a type of white-box testing. It is a testing for low-level design code. It is done within a class and starts from a individual module[5].
- d) Integration testing is done when two or more modules are combined together into a larger module[6].
- e) System testing verifies the functionality of the module after integration. It is done at the interfaces of the both the structure and component module. This type of testing is done in distributed or client/server modules. System Testing is also known as end-to-end testing. It tests the complete environment of the applications[4].
- f) Load Testing is done to determine whether the system is able to handle given number of users or not. It is done only to check the performance of the system as it is working well or not [7].
- g) Regression testing is different from unit testing, integration testing or system testing. It is related to bug fixing and breaking other things that the developer is not concentrating on. Sometimes bug fixes do not fix the bug. It also discovers fault localization[8].

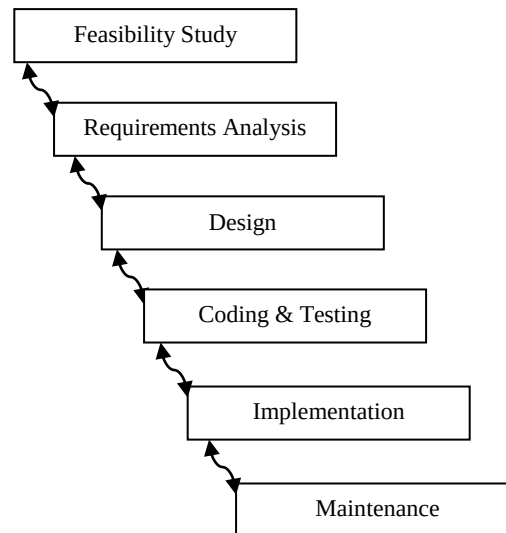


Fig.1.1 Software Testing

II. Regression Testing

Regression Testing is a type of testing that focuses on retesting after changes have been made. When the development team fixes bug then testing the other features of the application might be affected due to the bug fixing. It is known as regression testing[8]. It will increase the productivity and potency of software package and quality assurance of the applications. This type of testing is required when the system being tested is an up gradation of an already existing system to fix some bugs, enhance functionality or performance etc. Regression testing involves running of an old test suite after each change to the system or after each bug fix to ensure that no new bug has been introduced due to the change or the bug fix. If only a few statements are changed, then the entire test suite need not to be run, only those test cases that test the functions that are likely to be affected by the change need to be run[8]. For example, there are three modules like Admin, Personal Information and Employment. Suppose bug occurs in Admin module i.e. an existing user's account is not accessible with valid login-id and password, so this may be bug. Testing team will report the bug to the development team. Development team will fix the bug and hand over to testing team for testing. Testing team checks that the bug fixing does not affect the remaining functionality of same module or other modules[9].

A. Need and Importance of Regression Testing

- Regression testing is used for testing the correctness of a program.
- It is a quality measure which ensures that new code is consistent with the old code & unmodified code is not affected.
- It is important if there is frequently change or improvements in an existing application.
- The test case used for the regression testing can be reused to save time and effort required to develop new test cases.
- It reduces the product release risk by covering almost all defects in early stages of the development rather than finding and fixing them at the end of the release cycle.

B. Regression Testing Methodologies

Regression testing involves four methodologies. These methods are[10]:

1) Retest all

In this technique, all the tests in the existing test suite must be re-executed. The test cases which do not apply to the modified version of program are discarded and the remaining set of test cases is used to test the modified program. It is a very expensive process as it consumes a huge amount of time and resources due to execution of unnecessary tests.

2) Regression Test Selection

It is less expensive than retest all. It utilizes the information about the program, modified program and test cases to select a subset of test cases for testing. It selects only those test cases which are required.

3) Test Suite Reduction

It removes redundant test cases in addition to test selection. It is better than previous technique as it reduces the cost of validation, execution and managing of test suites over future releases of software. This technique has a limitation that it may reduce the fault detection capability with the reduction of test suite size.

4) Prioritization of test cases

In this technique, each test case is assigned a priority. Priority is set according to some criterion and the test cases which have highest priority is scheduled first. Criteria for test case selection may be that the test case which has faster code coverage gets highest priority. Another criterion may be the rate of fault detection. Advantage of this technique is that it does not discard or permanently remove the test cases from test suite.

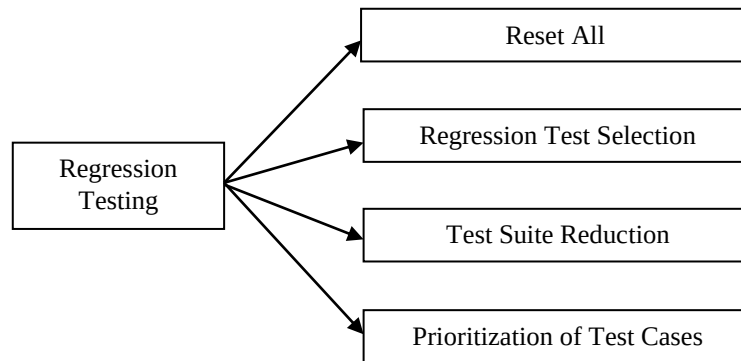


Fig.1.2 Regression Testing Methodologies

III. Test Case Prioritization

Test cases acts like a starting point of the test execution. It represents a set of conditions or variables in which a tester will determine whether a system under test satisfies requirements or works correctly. Test cases are generally written by a team member who understands the functionality or technology being tested. Test case must be submitted for peer review.

A. Prioritization Techniques

Test case prioritization techniques provide a way to schedule and run test cases which have the highest priority in order to provide earlier detection of faults. Each test case is assigned a priority. There are various prioritization techniques.

1) Customer Requirement-Based Prioritization

This technique prioritizes test cases based on the requirement documents. It uses various factors to weight (or rank) the test cases. The factors may be according to customer-assigned priority(CP), requirement complexity (RC) and requirement volatility (RV). The scaling factor can vary from 1 to 10. Highest factor values indicates need for prioritization of test case related to that requirement. If requirements have high complexity then it leads to maximum number of faults[11].

2) Code Coverage Technique

It specifies the quantity of source code that has been exercised during testing[12].

- It specifies those areas of a program which are not exercised by a set of test cases.
- Additional test cases may be generated to increase coverage.
- It determines a quantitative measure of code coverage, which is an indirect measure of quality.
- It identifies redundant test cases that do not increase coverage.

3) Fault Severity Technique

Test case prioritization depends upon the severity of the faults. The order of execution of test case depends on the size of the test suite and how much time each test case takes to run. The use of an effective prioritization technique can help testers to obtain an increased rate of fault detection[13]. The rate of fault detection is considered good if

- It disclose faults earlier that have high risk
- It reveal faults related to critical code sections earlier
- Provides confidence in the system's reliability earlier

4) History Based Technique

This technique prioritizes test cases based on the historical execution knowledge. For a test case, using historical information of each prioritization, increase or decrease its likelihood in the current test session. A probability value is assigned to the history and the value of the current session is calculated based on the number

of the fault detection (or coverage). Finally, the calculated value of a test case is equal to the sum of the current number multiplied by a probability and the historical value multiplied by another probability[14][15].

5) *Cost Effective-based Technique*

This method is used to prioritize test cases on the basis of various costs such as cost of analysis, cost of validation and cost of prioritization etc. The cost of a test case can be related to the resources required for execution and validation. To minimize the costs associated with testing and software failures, the goal of testing must be to uncover as many defects as possible with as little testing as possible i.e. write test cases that have a high likelihood of uncovering the faults that are most likely to be observed as a cause of failure in normal use. The cost-benefits analysis should be considered first before they are applied to large programs[16].

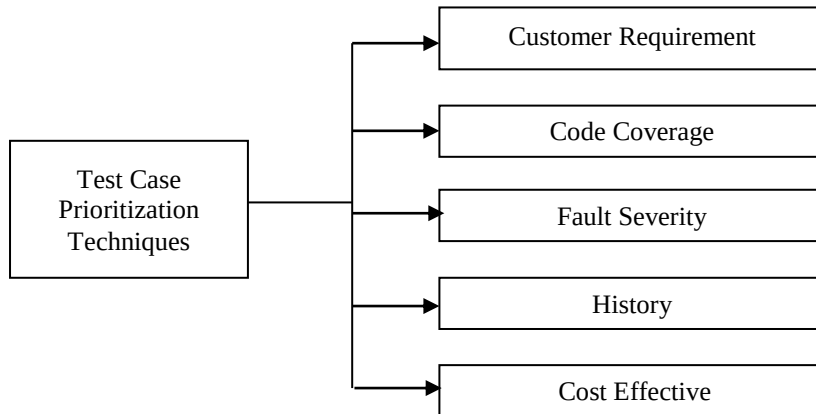


Fig. 1.3 Test Case Prioritization Techniques

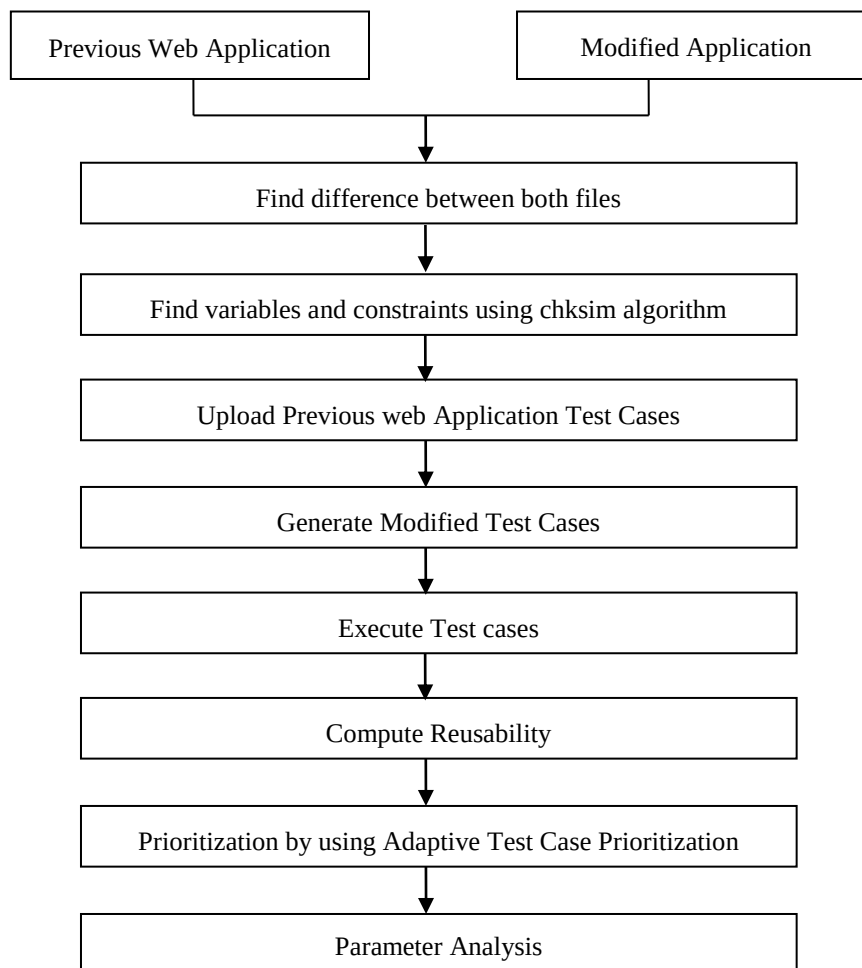


Fig. 1.4 Proposed Methodology

IV. Proposed Methodology

A company that provides web applications undergoes different security attacks and feature changes in their products according to the demands of users. Due to changes in the various fields of web applications, the testing has to be performed again and again. The main problem occurred in reusability of test cases is that it does not consider optimization of test suites developed automatically. The test cases that have been developed using reusability constraints must act as priority based test cases for regression testing. Fig.1.4. represents the proposed methodology. It consists of following steps:

Step 1: Take the Source Code from a web application like an online hotel booking web application and perform some modification in the source code.

Step 2: Find out the differences between the source code and modified code.

Step 3: After finding the difference, extract the variables and constraints from both files by using an algorithm (Chksim). Chksim algorithm matches each variable from previous test version and modified version. This algorithm provides the list of the variables that can be reused [17].

Step 4: Upload the Previous Test Cases.

Step 5: It will automatically reuse the test cases and generate the modified test cases.

Step 6: Execute the test cases that include the total number of test cases, number of test case failure, total reuse test cases, Runtime of test cases.

Step 7: Finally the Reusability is concluded. It includes the number of variables, constraints and show the Percentage of Reusability.

Step 8: After computing the reusability, prioritize test cases by using adaptive test case prioritization approach.

Step 9: Perform analyses of the parameters like reusability, execution runtime of test cases and execution runtime of prioritization.

V. Implementation

Implementation is done using Java. Java is an object oriented programming language. It supports portability and platform independent features i.e. Java programs can run on any browser and operating system.

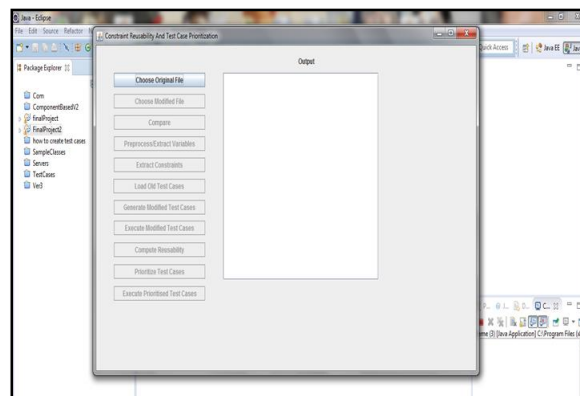


Fig. 1.5 Web application for Reusability

Figure 1.5 represents the application that has been developed for the computation of reusable variables and constraints between previous and modified version of the web application.

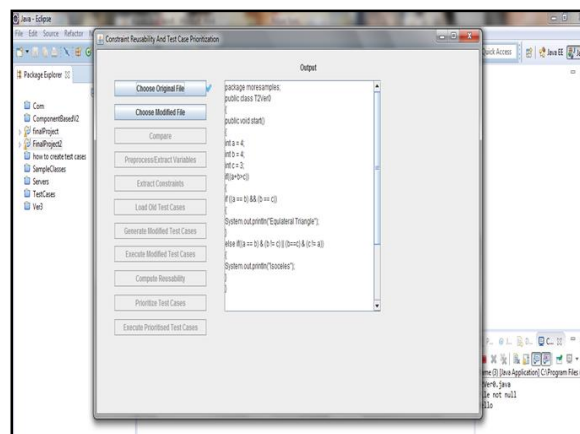


Fig. 1.6 Uploading of an original file

Figure1.6 represents an original file containing the source code from the web application. Files have been loaded from previous and modified version of the web application. These files have been scanned word to word so that each single variable can be computed for further generation of the test cases.

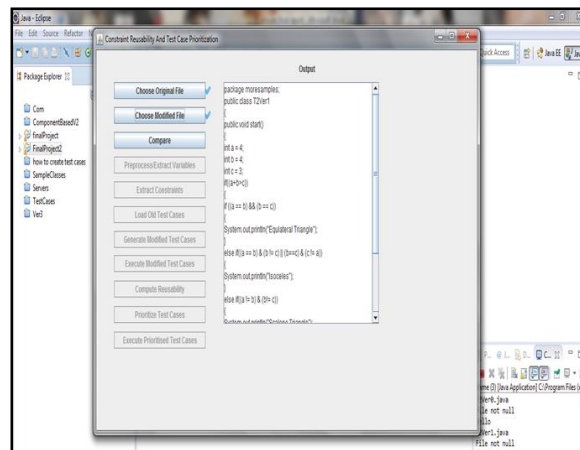


Fig. 1.7 Uploading of modified file

Figure1.7 represents the modified file that contains the source code with some changes in the web application.

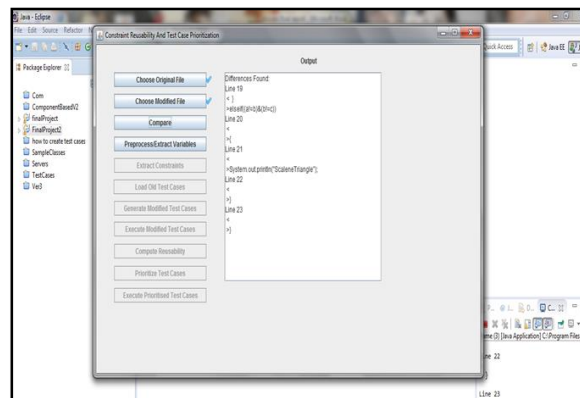


Fig.1.8 Extraction of difference between version files

Figure1.8 represents the difference that has been extracted by comparing different variables from the previous version file and modified version of file. The characters, symbols and variables have been scanned and compared. The variable list is generated that can be reused from these files. These variables can generate test cases for the modification region that was not available in the previous versions.

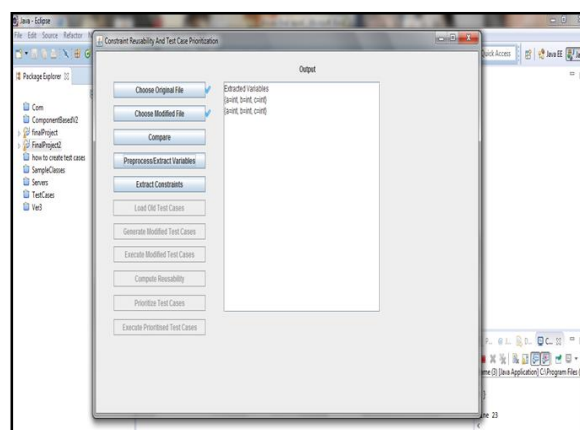


Fig.1.9 Extraction of Variables

Figure1.9 represents the variables from both original file and modified file which can be reused

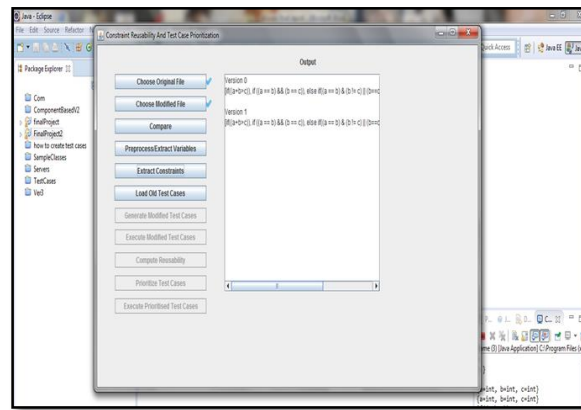


Fig.1.10 Extraction of constraints between versions files

Figure1.10 represents constraints and operands that can be used for the generation of test cases for the modification region that was not available in the previous versions.

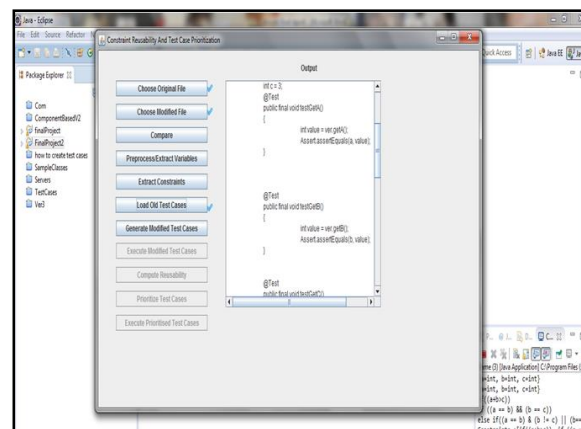


Fig.1.11 Loading of previous Test cases

Figure1.11specify that after computation of reusable variables and constraints, test cases for previous version have been loaded. The test cases for the modified version are available on the basis of difference computed in two different files of the versions. The test cases of the modified version have been loaded for prioritization process using test case prioritization approach and executed on the modified version for testing.

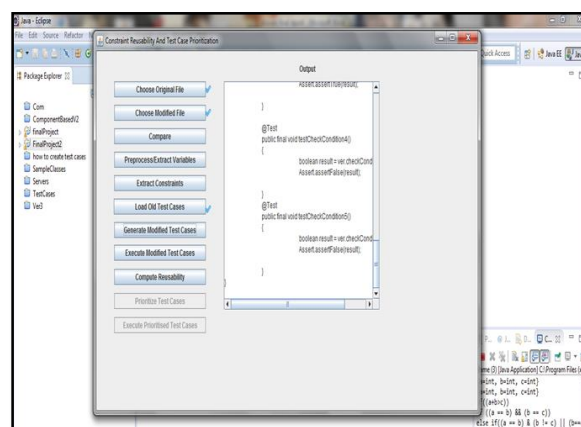


Fig. 1.12 Automatically generated test cases

Figure1.12 represents the test cases of modified version that is automatically generated from the previously uploaded test cases.

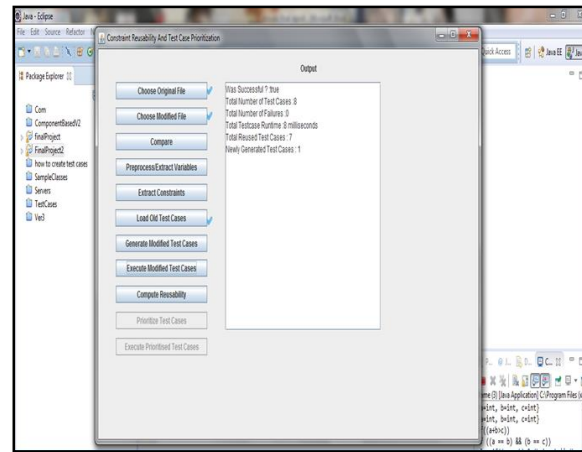


Fig.1.13 Execution of modified test cases

Figure1.13 shows the execution of modified Test Cases that shows total number of Test Cases, Failure of Test Cases, Execution time of test cases, Total reused Test Cases and Generated Test Cases.

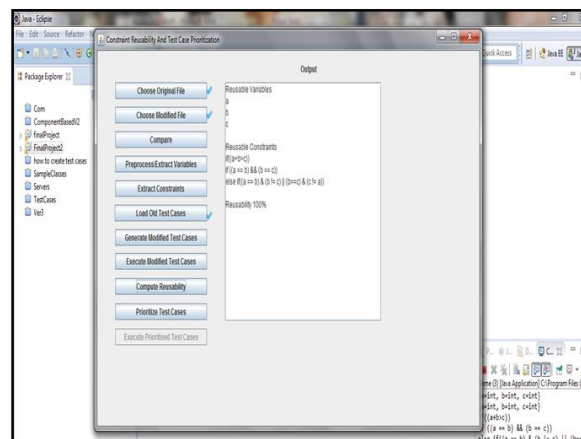


Fig.1.14 Computation of reusability

Figure1.14 represents computation of the reusability of the variables that can be reused for the previous and modified versions. The reusability is computed for various files of previous and modified versions. Maximum the reusability in the variables, Minimum efforts will be required for application development.

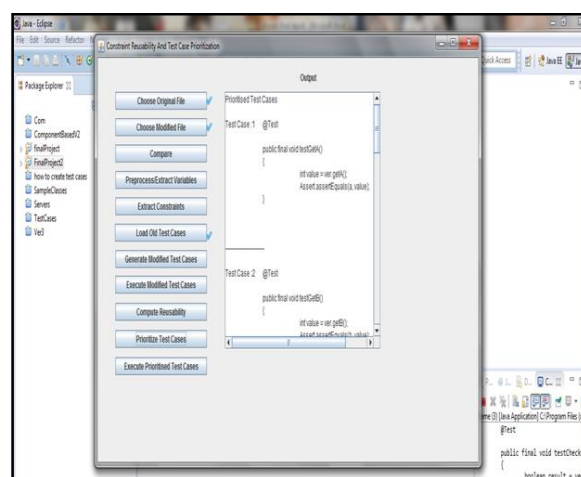


Fig.1.15 Test Case Prioritization

Figure 1.15 specify the test cases priorities to execute in an order. The test cases which have more impact on the system are assigned higher priority so that they will run earlier and consume less time.

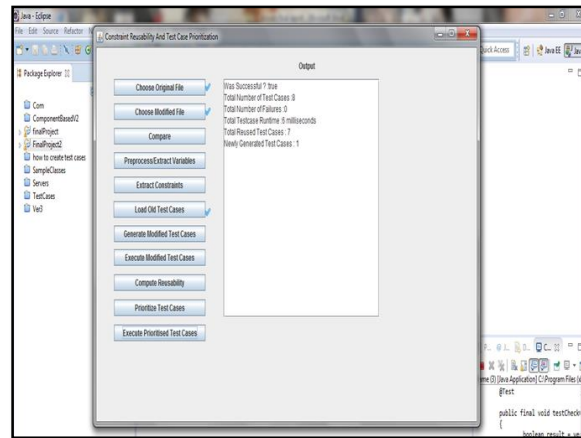


Fig. 1.16 Execution of Test Cases

Figure 1.16 shows execution of test cases that includes the total number of test cases, Number of Failures, Test cases Runtime, Total reused test cases and newly generated test cases.

VI. Results And Discussion

The following tables represent constraint reusability, test case reusability and efficiency of test cases before and after prioritization.

Table 1: Input values and Constraints Reusability

Version Pair	0 & 1	0 & 1	0 & 1
Total Paths	8	6	10
Total Inputs	4	3	3
Reusable Inputs	3	3	2
Regression Path	11(1)	2(19,21)	3(13,14,17)
Reusability	50%	100%	33%

Table 1 represents the variable and the constraints that have been reused in the modified version from previous version of file. It specifies total paths, total inputs, reusable inputs, regression path followed during testing & reusability percentage.

Table 2: Test Case Reusability

Version Pair	0 & 1	0 & 1	0 & 1
Total Test Cases	5	8	3
Total Reusable Test Cases	5	7	2
Test Case Reusability	100%	88%	66%

Table 2 represents the values for reusability of the test cases that have been reused in the modified version from previous version. The test cases that have been loaded from previous version depends upon the difference computed between both previous and modified file.

Table 3: Efficiency of Test Cases before and after Prioritization

Version Pair	0 & 1	0 & 1	0 & 1
Total Test Case Runtime	5 millisecond	8 millisecond	3 millisecond
Total Test Case Runtime	4 millisecond	6 millisecond	2 millisecond
Efficiency after Prioritization	80%	75%	66%

Table 3 represents the time taken by the system for the execution of test cases before and after optimization process of prioritization.

VII. Conclusion

In this paper, a technique is presented which identifies reusable constraint values for regression test cases by analyzing definitions and uses of the variables for two consecutive versions. It is concluded that a large number of constraint input values may be reused from the previous version's test cases. It reduces a significant amount of effort for resolving constraint values for those variables when testing is performed on new versions of the application. Further, the constraints and actual values for variables may be reusable across several versions as long as the definition and use relationships of the variables hold across different versions. Test cases are also prioritized to cover maximum area of the web application that has been set as maximum priority. The reusability of the variable and constraints decrease time, cost and efforts to develop the new test cases as the applications evolve over time.

VIII. Future Scope

While the experiment results are promising, the approach uses a relatively small number of two consecutive versions of an application. It has to be evaluated using widely used open source web applications with three versions or industrial-size applications. The prioritization can also be done by using genetic algorithm or artificial intelligence approaches. This can be helpful for optimizing the test suits on the basis of fitness value.

IX. References

- [1] Sommerville, Ian "Software-Engineering", Wesley, 9th Edition-2011.
- [2] Pressman, Roger S., "Software engineering: A practitioner's approach", McGraw-Hill Companies, 5th edition, 2005.
- [3] Isha Sharma and Jasleen Kaur, "International Journal of Computer Science and Mobile Computing", Vol.3 Issue.7, July- 2014, pp. 607-614.
- [4] Mangal, B. S., "Analyzing Test Case Selection using Proposed Hybrid Technique based on BCO and Genetic Algorithm and a Comparison with ACO", IJCA, 2012.
- [5] Dr. Arvinder Kaur, Shivangi Goyal, "A Bee Colony Optimization Algorithm for Fault Coverage Based Regression Test Suite Prioritization", International Journal of Advanced Science and Technology, Issue 29, 2010.
- [6] Corina S, Pasareanu, Willem Visser, "A survey of new trends in symbolic execution for software testing and analysis", pp 339- 353, 2011.
- [7] Swarnendu Biswas and Rajib Mall, "Regression Test Selection Techniques: A Survey", Informatica 35, 289–321, April 2011.
- [8] Mall, Rajib, "Fundamentals of Software engineering", PHI Learning Pvt. Ltd., 2014.
- [9] <http://www.onestoptesting.com/regression-testing/types.asp>
- [10] P.Ranjan Srivastava, "Test Case Prioritization", Journal of Theoretical and Applied Information Technology, pp.178-181, 2005.
- [11] G.Rothermel, R.Untch, C.Chu and M. Harrold, "Test Case Prioritization", IEEE Transactions on Software Engineering, pp. 929-948, vol. 27, 2001.
- [12] Hyunsook Do and Gregg Rothermel, "A Controlled Experiment Assessing Test Case Prioritization Techniques via Mutation Faults", IEEE International Conference proceedings on Software Maintenance, pages 411-420, 2005.
- [13] M.Thillaikarasi and Dr.K.Seetharaman "Effectiveness Of Test Case Prioritization Techniques Based On Regression Testing", International Journal of Software Engineering & Applications (IJSEA), pp.113-123, Vol.5, No.6, November 2014.
- [14] Jung-Min Kim and A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments", ICSE '02: Proceedings of the 24th International Conference on Software Engineering, pages 119–129, New York, NY, USA, 2002, ACM Press.
- [15] Chu-Ti Lin "History-Based Test Case Prioritization with Software Version Awareness" IEEE Conf. on Engineering of Complex Computer Systems (ICECCS), pp 171 – 172, 2013.
- [16] Alexey G. Malishevsky, Gregg Rothermel and Sebastian Elbaum, "Modeling the Cost-Benefits Tradeoffs for Regression Testing Techniques", Proceedings of the International Conference on Software Maintenance (ICSM'02), 2002.
- [17] Md. Hossain, Hyunsook Do, Ravi Eda, "Regression Testing for Web Applications Using Reusable Constraint Values", IEEE International Conference on Software Testing, Verification, and Validation Workshops, pp.312-321, 2014.
- [18] Engstrom, E. "Improving Regression Testing Transparency and Efficiency with History-Based Prioritization-An Industrial Case Study", IEEE 4th International Conference on software testing verification and validation (ICST), PP. 367-376, 2011.
- [19] Lei Xu "A framework for Web applications tests", ISSN 0-7695-2140-1, pp. 300 – 305, IEEE, 2014.
- [20] Dan Hao "Adaptive Test-Case Prioritization Guided by Output Inspection" International conf. on Computer Software and Applications Conference (COMPSAC), pp 169 – 179, 2013.
- [21] Chen, L., Wang, Z., Xu, L., Lu, H., & Xu, B. "Test case prioritization for web service regression testing", Service Oriented System Engineering (SOSE), Fifth IEEE International Symposium (pp. 173-178). 2010.
- [22] Chaturvedi, Animesh and Atul Gupta. "A tool supported approach to perform efficient regression testing of web services", Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), IEEE 7th International Symposium, pp. 50-55, 2013.