Performance Evaluation of Cost-cognizant Test Case Prioritization

PrakashSrivastava
Department of Computer Science & Engineering
Invertis University,
Bareilly, India
prakash2418@gmail.com

***Abstract*-**Software testing is indispensable for all software development. In software development practice, testing accounts for as much has 50% of total development efforts. Regression testing has been used to support software testing activities and assure the acquirement of appropriate quality through several versions of a software program. Regression testing, however, is too expensive because it requires many test case executions, and the number of test cases increases sharply as the software evolves. Consequently, this leads to the evolution of Test Case Prioritization which helps in minimising the Regression test suite reduction so that the effectiveness of Regression testing enhances. Cost-cognizant test case prioritization incorporates test costs and fault severities into test case prioritization as important factors. As a result of the proposed approach, software testers who perform regression testing are able to prioritize their test cases so that their effectiveness can be improved.

***Key words*-**APFD, regression, severity

I. INTRODUCTION

Software testing is the process of validation and verification of the software product. According to Myers "Software Testing is the process of executing a program with the intent of finding errors". Effective software testing will contribute to the delivery of reliable and quality oriented software product, more satisfied users, lower maintenance cost, and more accurate and reliable result. However, ineffective testing will lead to the opposite results; low quality products, unhappy users, increased maintenance costs, unreliable and inaccurate results. Hence, software testing is a necessary and important activity of software development process.

The importance of testing can be understood by the fact that "around 35% of the elapsed time and over 50% of the total cost are expending in testing programs". Software is expected to work, meeting customer's changing demands, first time and every time, consistently and predictably. Earlier software systems were used for back-office and non-critical operations of organizations.

A software product, once developed, has a long life and evolves through numerous additions and modifications based on its faults, changes of user requirements, changes of environments, and so forth. With the evolution of a software product, assuring its quality is becoming more difficult because of numerous release versions. It is becoming much harder to manage the software itself. On the other hand, users hope that a new software version has better quality than before. However, sometimes the quality of software becomes worse than before because the added or modified features create

additional faults into the existing product as well as the newly modified version.

Regression Testing is a maintenance activity that attempts to validate modified softwareandensure that modifications are correct and have not inadvertently affected the software.Thepurpose of regression testing is to ensure that changes made to software, such as adding new features or modifying existing features, have not adversely affected the existing features of the software. Regression testing is usually performed by running some, or all, of the test cases created to test modifications in previous versions of the software.

A. Techniques of Regression Testing

The various techniques of regression testing are as follows:

A.1 Retest All

The simplest regression testing strategy is to rerun all existing test cases. This strategy is easy to implement, but can be unnecessarily expensive, especially when changes affect only a small part of the system. This retest-all approach may consume excessive time and resources. It does not require any test selection process, the retest-all approach over time becomes less and less affordable for complex systems.

A.2 Regression Test Selection (RTS)

Regression test selection techniques select a subset of the existing test suite for execution, depending on factors such as the changes made to the code and the execution behaviour of tests.With this approach only a subset of the test cases contained in a test suite are selected and rerun. Reducing the number of test cases rerun reduces regression testing costs, but may also cause fault-revealing test cases to be omitted. Since, in general, optimal test selection is impossible, the cost-benefit tradeoffs of RTS techniques are a central concern of regression testing research and practice. Regression test selection techniques can have substantial costs, and can discard test cases that could reveal faults, possibly reducing fault detection effectiveness.

A.3 Test Suite Reduction

Test suite reduction techniques permanently reduce the test suite by identifying and discarding redundant tests. Test suite reduction techniques address this problem by using information about P and T to permanently remove redundant test cases from T, so that subsequent reuse of T can be more efficient. Reduction thus differs from selection in that the latter does not permanently remove test cases from T, but simply “screens” those test cases for use on a specific version P′ of P, retaining unused test cases for use later.

A.4 Test Case Prioritization

To reduce the cost of regression testing, software testers may prioritize their test cases so that those which are more important, by some measure, are run earlier in the regression testing process. One potential goal of test case prioritization techniques is to increase a test suite’s rate of fault detection (how quickly, in a run of its test cases, that test suite can detect faults). Test case prioritization provides a way to run test cases with the highest priority earliest according to some criterion earliest, and can yield meaningful benefits, such as providing earlier feedback to testers and earlier detection of faults. Test case prioritization can significantly improve the rate of fault detection.

B.Cost-cognizant test casePrioritization

The goal of this paper is to present a survey of the main regression-testing techniques proposed to date, so as to better understand the state of the research and the state of the practice in regression testing, along with a discussion of the current trends in both academia and industry.

Since test case prioritization was introduced, there has been an important weakness in the technique; there has been no consideration of test costs and fault severities. For this reason, test case prioritization techniques often produce no appropriate test orders in practice [3]. Cost-cognizant test case prioritization incorporates test costs and fault severities into test case prioritization [1, 3]. In short, cost-cognizant test case prioritization considers the test cost and fault severity of each test case as important factors, and the test cost and fault severity are used for prioritizing test cases on the existing test case prioritization algorithms.

The purpose of this prioritization is to increase the likelihood that if the test cases are used for regression testing in the given order, they will more closely meet some objective than they would if they were executed in some other order. However, this cost-cognizant test case prioritization technique reveals a problem; the specific way to estimate cost and fault severity is not clarified even though such estimations are needed.

B.1 Historical Value-Based Approach

This approach is based on the use of historical information, toestimate the current cost and fault severity for cost cognizant test case prioritization to determine the priority of given test cases. By using the historical information of the costs of the test cases and the fault severities of detected defects in a test suite, the historical value of the test cases is calculated and used for the basis of test case prioritization. Additionally, the historical value can be combined with not only a cost-cognizant test case prioritization technique, but also several existing test case prioritization techniques such as a coverage-based test case prioritization technique. Namely, the historical value is calculated from the previous test costs and fault severities of detected defects in a test suite. Then, the historical value is used for the factor that affects the prioritization of test cases in a given test suite. Effectiveness of this approach can be quantified in terms of Average Percentage of Faults Detected (APFD) in a particular test suite.

B.1.1 Approach and Overview

- To conduct regression testing for P′, a test suite is composed of the test cases from the

test case repository.

- The cost of a test case and fault severity of the detected defects, which are the results from the execution of a test case, are stored in the historical information repository.
- When the prioritization is required, the historical value model uses the stored historical information, the test costs of the test cases and the fault severities of the detected defects, and calculates the historical value.
- The calculated historical value is used for the criterion of prioritizing test cases in a test suite. The figure 2.1 shows the overall description of the Historical value based approach for test case prioritization.

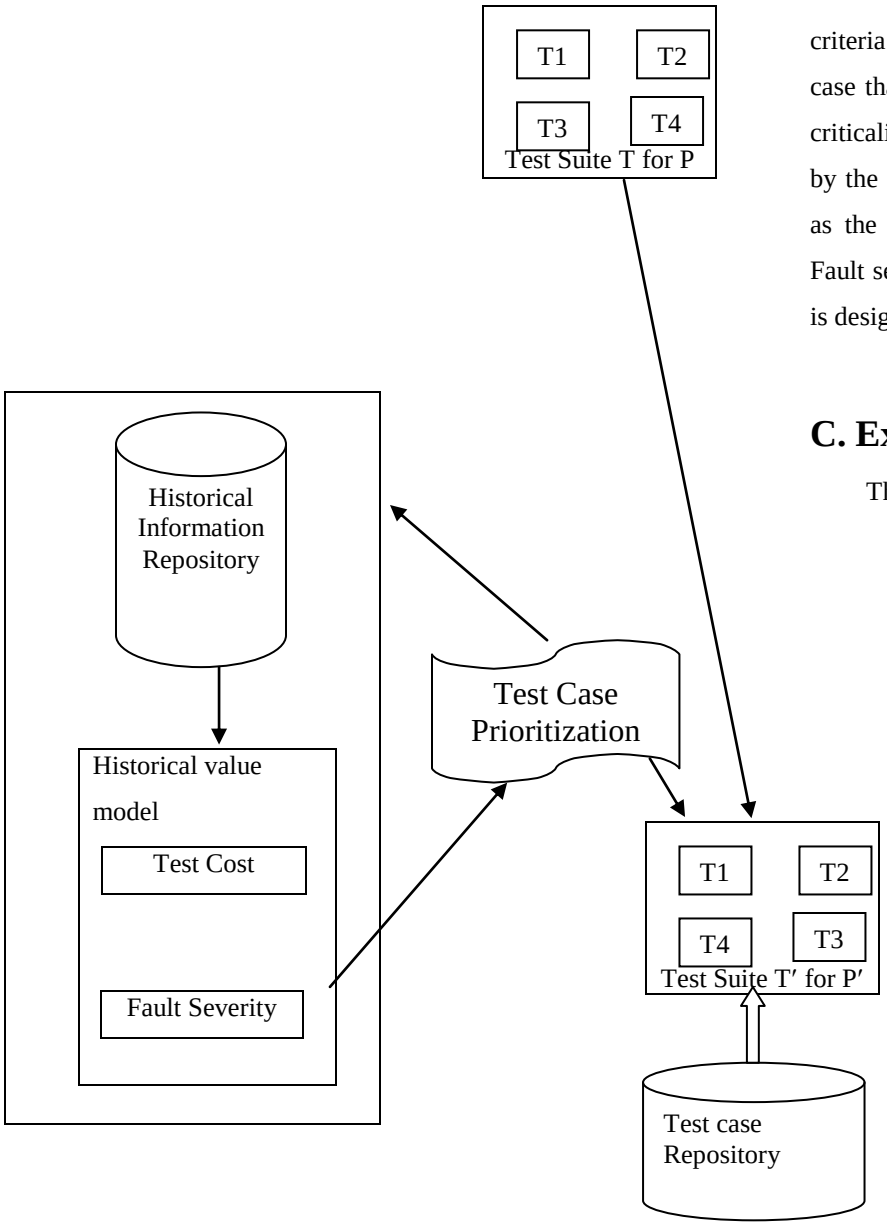

Figure 1: Overview of Historical value based approach

B.1.2 Test Cost and Fault Severity

Test costs are greatly diversified in software testing. Depending on the criteria, a test cost can be refined through several factors such as machine time, human time, test case execution time, monetary value of the test execution, and so forth [1]. Similarly, fault severity can also be refined by depending upon such criteria as test criticality (the criticality of the test case that detects a fault) and function criticality (the criticality of the function in the code that is covered by the test case).In our approach, test cost is refined as the test case execution time of a test case[1,3]. Fault severity is refined to test case criticality, which is designated to each test case by software testers.

C. Experiment Environment

The experiment environment will consist of:

- JUnit testing framework to provide for assessing test case prioritization techniques using hand-seeded faults, test case execution is also performed on the JUnitenvironment [15].JUnit is a simple, open source framework to write and run repeatable tests. Approximately thousands of test cases are contained in a Test suite of Junit under various Directories like Experimental, Manipulation, Assertion, Description etc. These test cases are executed in a Junit testing environment to exhibit following information:Success and failure of Test cases, Test case execution time, which is also considered as the Cost Severity. For the target of the testing objects, whose faults are seeded by hand, and tools came from Software-artifact Infrastructure Repository[16].
- For the execution of the experiment Microsoft Dot Net framework is used in which crystal reportisusedfor the display of graphs and results.Microsoft visual studio is a software tool for automating software build processes. It

is similar to netbeans but is implemented using the VB.Net

The figure 2 describes the overall structure of the experimental environment.

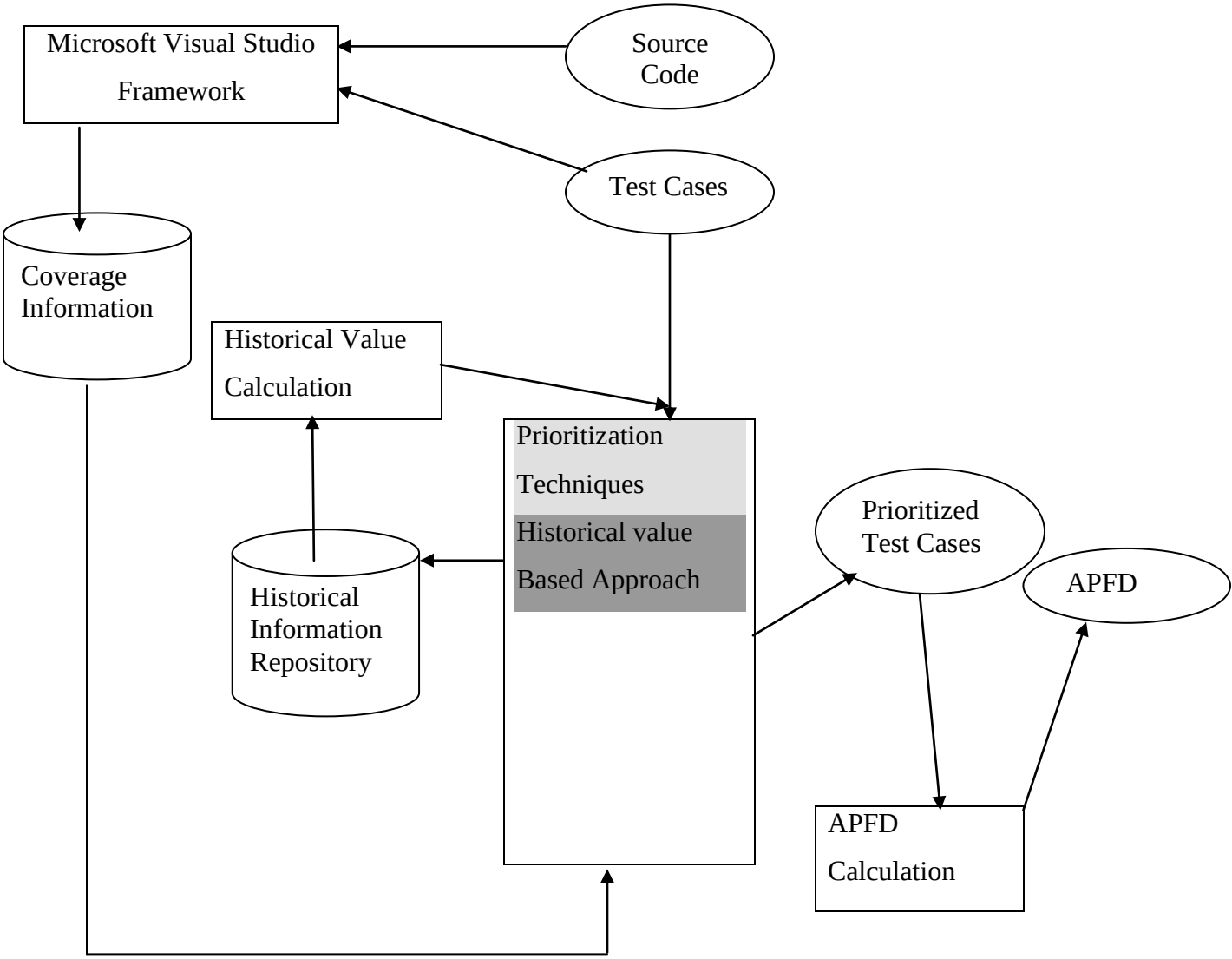

Figure 2: Overview of the experiment environment

language, requires the windows operating system platform and is best suited to build projects.

D.Results and Discussion

D.1 Implementation Parameters

The salient parameters through which we analyze the performance of Cost Cognizant Test Case Prioritization:

- Execution Time
- Cost Severity
- Fault Severity
- Average Percentage of Faults Detected(APFD)

The performance of Cost Cognizant Test Case Prioritization is analyzed through Junit testing framework for execution time & cost severity. The fault severity is analyzed on the basis of Software

artifact Infrastructure repository whose faults are integrated through our experiment environment. APFD performance is analyzed through crystal reports and graphs. Given below is a instance of faults detected in a particular test suite.

TABLE I: Fault Matrix

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
F1	*					*				
F2				*			*	*	*	
F3		*			*	*				*
F4							*			
F5		*						*	*	
F6				*						
F7				*	*					
F8		*	*							
F9						*				
F10	*									*
No. of Faults	2	3	1	3	2	3	2	2	2	2
Time in Min	5	7	11	4	10	12	6	15	8	9

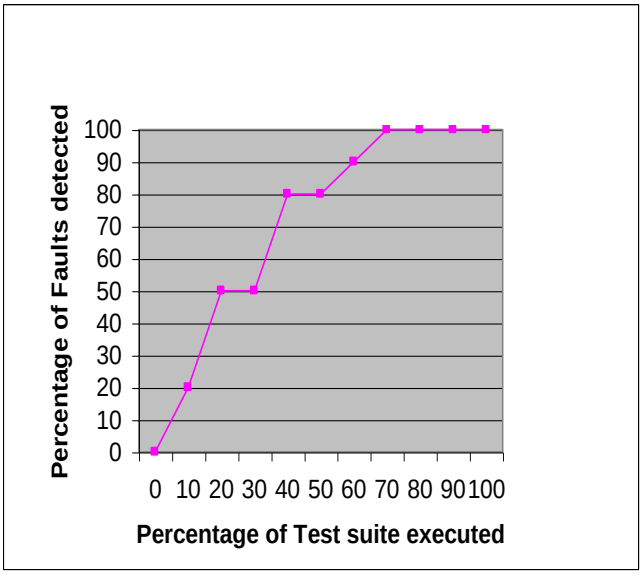

Figure 3: APFD for non prioritized test suite

Non Prioritized test sequence in the order of

T1	T2	T3	T4	T5	T6	T7	T8	T9	T10

Figure 3 illustrates that following the Non Prioritized order of test sequence percentage of faults are exposed in a large execution time of test cases. The number of Test cases are also more to be executed to identify faults.

Figure 4 illustrates that following the Prioritized order of test sequence percentage of faults are exposed in a less execution time of test cases. The number of test cases is less as compared to non prioritized order of test cases. It shows that Historical approach is efficient to find more number of faults detected in lesser number of test case execution comprising of various test cases in a particular test suite. Hence the result shows that it will improve the effectiveness of software tester who will perform regression testing in less time with less effort.

Prioritized test sequence in the order of

T4	T2	T1	T7	T6	T9	T10	T5	T8	T3

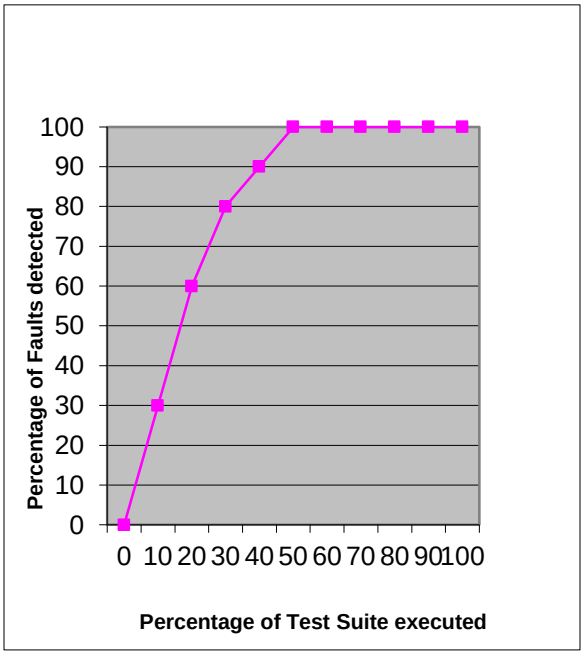

Figure 4: APFD for prioritized test suite

E. Conclusion

The test cases are executed in a junit testing environment to know about the execution time, success and failure of test cases, type of error identification. The historical value module is also made to calculate historical value for the test cases. The .NET framework is used for the target of the experiment which is comprised of several modules. The junit information is integrated successfully with the help of java class path settings. The performance of cost cognizant test case prioritization using historical value based approach is analyzed through crystal report graphs and fault matrix in terms of Average Percentage of Fault detected (APFD). The results of the experiment shows the performance of cost cognizant test case in terms of (APFD) and software testers who perform regression testing are able to prioritize their test cases so that their effectiveness can be improved in terms of their effort & accuracy. The major contributions of this research work are the following two points. First, it provides a way to estimate the cost and fault severity of the current test case by using Junit testing framework & historical information. Second, the proposed approach can complement other test case prioritization techniques because it can be combined with other test case prioritizations.

REFERENCES

[1] Sebastian Elbaum, Alexey Malishevsky, and Gregg Rothermel, "Incorporating Varying Test Costs and Fault Severities into Test Case Prioritization", Proceedings of the International Conference on Software Engineering, vol. 3, pp. 329-338, May 2001.

[2] Institute of Electrical and Electronics Engineers (IEEE) IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries New York, NY: 1990.

[3] Alexey G. Malishevsky, Joseph R. Ruthruff, Gregg Rothermel, and Sebastian Elbaum, "Cost-cognizant Test Case Prioritization", Technical Report TR-UNL-CSE-2006-0004, University of Nebraska-Lincoln, March 2006.

[4] Gregg Rothermel and Mary Jean Harrold "Empirical Studies of a Safe Regression Test Selection Technique", IEEE transactions on Software Engineering, vol. 24, pp. 401-419, June 1998.

[5] Zheng Li, Mark Harman, and Robert M. Hierons "Search Algorithms for Regression Test Case Prioritization" IEEE transactions on Software Engineering , vol. 33, pp. 225-237, April 2007.

[6] AnjaneyuluPasala, L.H Yannick, Lew Yaw Fung, FadyAkladios, AppalaRaju G, Ravi P Gorthi "Selection of Regression Test Suite to Validate Software Applications upon Deployment of Upgrades" 19th Australian Conference on Software Engineering vol. 26, pp. 130-138, March 2008.

[7] T.M.S.UmmuSalima, A.Askarunisha, N.Ramaraj "Enhancing The Efficiency of Regression Testing Through Intelligent Agents" International Conference on Computational Intelligence and Multimedia Applications, vol 1, pp. 103-108, Dec. 2007.

[8] Mark Sherriff, Mike Lake, and Laurie Williams "Prioritization of Regression Tests using Singular Value Decomposition with Empirical Change Records", 18th IEEE International Symposium on Software Reliability, pp. 81-90, Nov. 2007.

[9] Hao Chen, BeijiZou, NaizhengBian, Lili Pan "A Reusable Component-Based Library for GUI Regression Testing", First International Workshop on Knowledge Discovery and Data Mining, pp. 326-329, Jan 2008.

[10] Mary Jean Harrold and Alessandro Orso "Retesting Software During Development and Maintenance", Frontiers of Software Maintenance, pp. 99-108, September 2008.

[11] Xiao Qu, Myra B. Cohen, Katherine M. Woolf "Combinatorial Interaction Regression Testing: A Study of Test Case Generation and Prioritization", IEEE International Conference on Software Maintenance, pp. 255-264, Oct. 2007.

[12] Henry Muccini, Via Vetoio, "Using Model Differencing for Architecture-level Regression Testing", 33rd EUROMICRO Conference on Software Engineering and Advanced Applications , pp. 59-66, Aug. 2007.

[13] Gregg Rothermel, Roland H. Untch, Chengyun Chu, and Mary Jean Harrold, "Prioritizing Test Cases for Regression Testing", IEEE Transactions on Software Engineering, vol. 27, pp. 929-948, October 2001.

[14] Hyuncheol Park, HoyeonRyu, JongmoonBaik "Historical Value-Based Approach for Cost-cognizant Test Case Prioritization to Improve the Effectiveness of Regression Testing", Second International Conference on Secure System Integration and Reliability Improvement pp. 39-46, July 2008.

[15] JUnit Testing Framework http://www.junit.org

[16] Software-artifact Infrastructure Repository, http://sir.unl.edu

[17] Alex Keener, Sofya – dynamic program analysis for Java software, http://sofya.unl.edu