ADAPTIVE REGRESSION TESTING STRATEGY:

AN EMPIRICAL STUDY

A Thesis
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Md. Junaid Arafeen

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Program:
Computer Science

July 2012

Fargo, North Dakota

North Dakota State University
Graduate School

Title
Adaptive Regression Testing Strategy:

An Empirical Study

By

Md. Junaid Arafeen

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Hyunsook Do
Chair

Dr. Kenneth Magel

Dr. Saeed Salem

Dr. Zakaria Mahmud

Approved by Department Chair:

7/3/2012	Dr. Brian M. Slator
Date	Department Chair

ABSTRACT

When software systems evolve, different amounts of code modifications can be involved in different versions. These factors can affect the costs and benefits of regression testing techniques, and thus, there may be no single regression testing technique that is the most cost-effective technique to use on every version. To date, many regression testing techniques have been proposed, but no research has been done on the problem of helping practitioners systematically choose appropriate techniques on new versions as systems evolve. To address this problem, we propose adaptive regression testing (ART) strategies that attempt to identify the regression testing techniques that will be the most cost-effective for each regression testing session considering organization's situations and testing environment. To assess our approach, we conducted an experiment focusing on test case prioritization techniques. Our results show that prioritization techniques selected by our approach can be more cost-effective than those used by the control approaches.

ACKNOWLEDGMENTS

I would like to thank the advisory committee members and professors at North Dakota State University (NDSU), Dr. Hyunsook Do, Dr. Kenneth Magel, Dr. Saeed Salem and Dr. Zakaria Mahmud for their valuable support and guidance. As my advisory committee chair, Dr. Hyunsook Do always made time in providing technical expertise and guidance throughout the process. This would not have been possible without her constant support.

I am also thankful to the National Science Foundation for funding this research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

Regression testing is an important and necessary activity that can maintain the quality of modified software systems. To date, many regression testing techniques have been proposed. For instance, regression test selection techniques (e.g., [1], [2], [3]) reduce testing costs by selecting test cases that are necessary to test a modified program. Test case prioritization techniques (e.g., [4], [5], [6]) reorder test cases, scheduling test cases with the highest priority according to some criterion earlier in the testing process to yield benefits such as providing earlier feedback to testers and earlier fault detection.

While this research has made considerable progress in regression testing areas, one important problem has been overlooked. As systems evolve, the types of maintenance activities that are applied to them change. Differences between versions can involve different amounts and types of code modifications, and these changes can affect the costs and benefits of regression testing techniques in different ways. Thus, there may be no single regression testing technique that is the most cost-effective technique to use on every version. For instance, as we observed from our study, prioritization technique, that works best, changes across versions.

We propose to address this lack by creating and empirically studying *adaptive regression testing* (ART) strategies. ART strategies are approaches that operate across system lifetimes, and attempt to identify the regression testing techniques that will be the most cost-effective for each regression testing session. ART strategies evaluate regression testing techniques in terms of decision criteria such as cost and benefit factors and choose the best alternative among techniques considering organization’s situations and feedback from prior regression testing sessions.

The problem of performing such evaluations is known as the “multiple criteria decision making” (MCDM) problem, and MCDM approaches have been used in many science, engineering, and business areas that involve complex decision problems, such as technology investment, resource allocation, and layout design [7], [8]. To date, many MCDM approaches have been proposed including the Weighted Sum Model (WSM), the Weighted Product Model (WPM), the Analytical Hierarchy Process (AHP), and other variants. Among these MCDM methods, AHP has been one of the more popular methods, having been used by researchers and practitioners in various areas including software engineering [9], [10], [11].

Therefore, in this research, as an initial approach to creating ART strategies, we investigated AHP method [7] to see whether AHP can be effective for selecting appropriate regression testing techniques across system lifetime, particularly focusing on test case prioritization techniques. To do this, we have designed and conducted a controlled experiment using several Java programs with multiple versions considering several selection strategies. The results of our experiment show that the prioritization techniques selected by AHP can be more cost-effective than those used by the control approaches.

In the next section, we describe background information and related work relevant to prioritization techniques and regression testing strategies. Chapter 3 describes our proposed approach, ART strategy. Chapter 4 presents our experiment setup, Chapter 5 presents results and analysis, and Chapter 6 address threats to validity. Chapter 7 discusses our results, and Chapter 8 presents conclusions and future work.

CHAPTER 2. BACKGROUND AND RELATED WORK

Regression testing attempts to validate modified programs to see whether changes have produced unintended effects. Depending on various factors, such as the size and complexity of the program and its test suite, regression testing process can be very expensive. Thus, many researchers have proposed numerous cost-effective regression testing techniques including regression test selection, test suite reduction/minimization, and test case prioritization, but here, we limit to our discussion to test case prioritization, which is directly related to our work.

Test case prioritization techniques (e.g., [6], [12]) reorder test cases in order to increase the chance of early fault detection using various types of information available from software artifacts, such as the coverage of code achieved by tests, code change information, or code complexity. For example, one technique, *total block coverage prioritization*, simply sorts the test cases in the order of the number of blocks they cover. One variation of this technique, *additional block coverage prioritization* iteratively selects a test case that yields the greatest block coverage, then adjusts the coverage information for the remaining test cases to indicate their coverage of blocks not yet covered, and then repeats this process until all blocks coverable by at least one test case have been covered.

To date, numerous test case prioritization techniques have been proposed, and a recent paper by Yoo and Harman [13] provides a comprehensive overview of these techniques. While the goal of the proposed techniques is to improve the effectiveness of regression testing, to be useful in practice, techniques should be applicable within various testing environments and contexts. Recent research on test case prioritization has employed empirical studies to evaluate the cost-benefit tradeoffs among techniques considering various factors and testing contexts [14], [15], [16], [17], [18]. For instance, Do et al. [14] have studied regression testing under time constraints. They perform multiple experiments to assess the effects of time constraints on the costs and benefits of prioritization techniques. At first, they show that time constraints can play a significant role in determining both the cost-effectiveness of prioritization and the relative cost-benefit trade-offs among techniques. Later they manipulate the number of faults present in programs to examine the effects of faultiness levels on prioritization and show that faultiness level affects the relative cost-effectiveness of prioritization techniques. Walcott et al. [18] present genetic algorithm to reorder test cases under time constraints. Qu et al. [17] consider prioritization in the context of configurable systems. They utilize combinatorial testing techniques to model and generate configuration samples for the regression testing.

Studies such as these have allowed researchers and practitioners to understand factors that affect the assessment of techniques and to compare techniques in terms of costs and benefits relative to actual software systems. However, studies to date have not considered strategies for selecting appropriate techniques under particular circumstances as systems evolve. Only few studies [19], [20] have done on the problem of helping practitioners choose appropriate techniques under particular system and process constraints. Harrold et al. [20] present a coverage based predictor model that predicts the cost effectiveness of a selective regression testing strategy. They show that only coverage information cannot successfully predict the cost-effectiveness of regression test selection method   code modifications that has been made in the ongoing version play a significant role to improve the accuracy of the prediction model. Elbaum et al. [19] perform experiments exploring characteristics of program structure, test suite composition, and changes on prioritization, and identified several metrics characterizing these attributes that correlate with prioritization effectiveness. The empirical results of their study provide insights into which prioritization technique is appropriate (or not appropriate) under specific testing scenarios. Unlike our approach, these two studies evaluate techniques solely relied on software metrics and did not consider the notion of software evolution context.

Since many factors can be involved in evolving systems, selecting appropriate techniques for each version can be a multiple criteria decision making (MCDM) problem. Analytic Hierarchy Process (AHP) is one of the widely used MCDM methods, and many areas that involve complex decision problems, such as business, manufacturing, science and engineering. For instance, Kamal and Al-Harbi [21] utilize AHP in project management to determine the best contractors to complete the project. They construct the AHP hierarchy with prequalification criteria and contractors. They prioritize the criteria and obtain a sorted list of contractors by applying the AHP process. AHP has also been used in determining the best manufacturing system [22], layout design [23], and the evaluation of technology investment decisions [24].

Recently AHP has been used in software engineering areas. Barcusa and Montibellerb [25] use AHP to allocate software development work in distributed teams. They develop a multi-criteria decision model to support the distributed team work allocation decision by using decision conferencing and multi-attribute value analysis. Finnie et al. [26] use AHP to prioritize software development productivity factors, and Perini et al. [27] compare AHP with other alternative method in prioritizing software requirements. Karlsson et al. [9] investigate six methods for prioritizing requirements: analytic hierarchy process (AHP), hierarchy AHP, spanning tree matrix, bubble sort, binary search tree, and priority groups. They apply all methods to prioritize 13 well-defined quality requirements on a small telephony system. They showed that the analytic hierarchy process is the most promising method among those six methods. Yoo et al. [11] use AHP to improve test case prioritization techniques by employing expert knowledge, and compare the proposed approach with the conventional coverage-based test case prioritization technique. Unlike their study, in this paper, we utilize AHP to develop adaptive regressions testing strategy, which helps identify the best test case prioritization techniques across system lifetime.

CHAPTER 3. ADAPTIVE REGRESSION TESTING (ART) STRATEGY

In this section, we describe AHP method and how AHP is used for creating ART strategy using an example.

3.1. AHP Method

The Analytic Hierarchy Process (AHP) is a technique for structuring and analyzing complex decision problems. It was developed by Thomas L. Satty in 1970. Since then it has been studied by many practitioners. It is used in a wide variety of multi-objective decision situations [22], [23].

To use AHP, decision makers first define a hierarchy that describes the problem they want to solve. As shown in Figure 3.1, adapted from [7], an AHP hierarchy consists of a goal that they want to achieve, alternatives that are available to reach the goal, and criteria that are factors that may be used in decision making about these alternatives. The criteria can be further partitioned into sub-criteria if necessary.

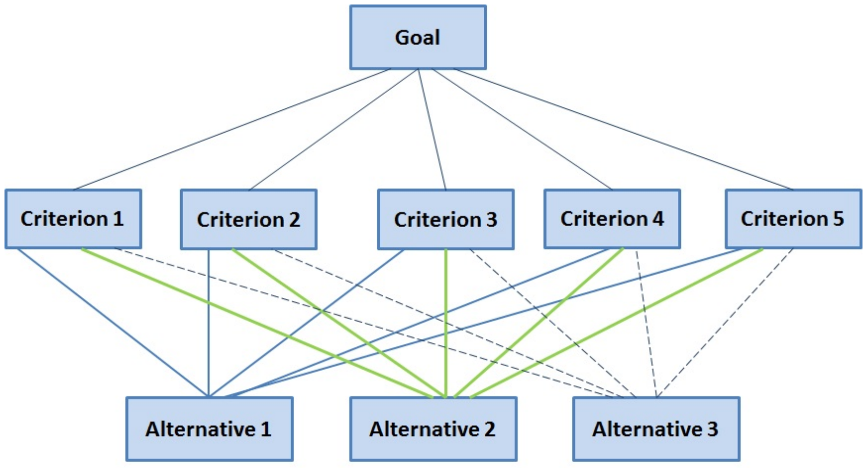
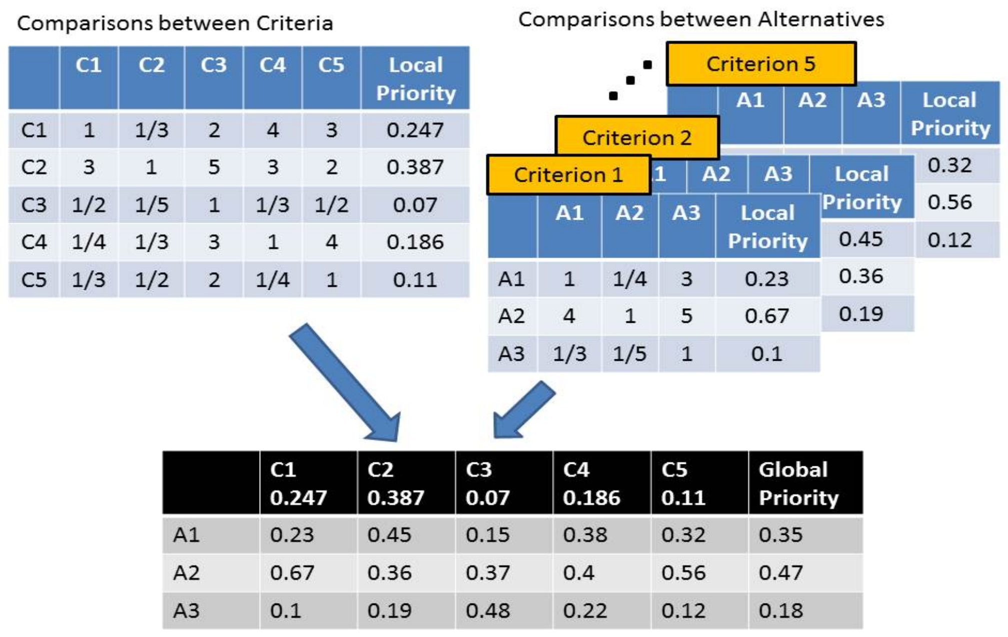


Figure 3.1. An AHP hierarchy

Once decision makers define an AHP hierarchy, two types of pairwise comparisons are performed: between pairs of criteria and between pairs of alternatives as shown in Figure 3.2. When comparing pairs of criteria (the upper left table), decision makers assign relative importance weights to criteria; for example, C1 is given importance 4 relative to C4. After completing this matrix, the assigned values are normalized and the local priority of each criterion is produced, which is shown in the rightmost column of the table (and in the top row of the bottom table in the figure). The local priority is calculated by the following equation:

$$LP_i = \frac{\sum_{j=1}^{N}(RW_{ij})}{\sum_{i=1}^{N}\sum_{j=1}^{N}(RW_{ij})} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \text{(1)}$$

, where *LP$_i$* is a local priority of criterion *i*, *RW$_{ij}$* is a relative weight of criterion *i* over criterion *j*, and *N* is the number of criteria (The local priorities of alternatives are calculated in the same way).



Comparisons between Criteria

	C1	C2	C3	C4	C5	Local Priority
C1	1	1/3	2	4	3	0.247
C2	3	1	5	3	2	0.387
C3	1/2	1/5	1	1/3	1/2	0.07
C4	1/4	1/3	3	1	4	0.186
C5	1/3	1/2	2	1/4	1	0.11

Comparisons between Alternatives

Criterion 5

	A1	A2	A3	Local Priority
				0.32
				0.56
				0.12

Criterion 2

	A1	A2	A3	Local Priority
				0.45
				0.36
				0.19

Criterion 1

	A1	A2	A3	Local Priority
A1	1	1/4	3	0.23
A2	4	1	5	0.67
A3	1/3	1/5	1	0.1

	C1 0.247	C2 0.387	C3 0.07	C4 0.186	C5 0.11	Global Priority
A1	0.23	0.45	0.15	0.38	0.32	0.35
A2	0.67	0.36	0.37	0.4	0.56	0.47
A3	0.1	0.19	0.48	0.22	0.12	0.18

Figure 3.2. An AHP example

Similarly, matrices that show the relative importance of alternatives for each criterion are constructed. In this example, five matrices are constructed because there are five criteria (the upper right tables). Again, the assigned values are normalized for each matrix, and local priorities are produced for each alternative (the resulting local priorities appear in the bottom table in the figure).

After calculating the local priorities for criteria and alternatives, an *M* x *N* matrix is constructed, as shown in the bottom table in Figure 3.2, where *M* is the number of alternatives considered and *N* is the number of criteria. In our example, *M* is 3 and *N* is 5. Then, weighted sums of the values per technique are calculated; these are shown in the rightmost column ("global priorities" *i*). The global priority is calculated by the following equation:

$$GP_k = \sum_{j=1}^{N} (LPA_{kj} * LP_{j)} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \text{(2)}$$

,where GP_k is a global priority for alternative *k*, *N* is the number of criteria, LPA_{kj} is a local priority of alternative *k* for criterion *j*, LP_j is a local priority of criterion *j*. Based on the weighted sum values, decision makers can determine which alternative should be selected. In this example, *T2* (0.47) performs best and *T1* (0.35) next best, with *T3* (0.18) far behind.

3.2. Applying AHP to Prioritization Strategy

We now describe how AHP is applied to prioritization strategy that we use in this work. While we describe this in terms of test case prioritization using one of the programs we used in our study, the approach could be applied to any regression testing techniques and any system for which the required information is available.

As outlined in the prior section, to apply AHP to prioritization strategy, the following steps are required:

1. Set a goal

2. Identify alternatives that are available to reach the goal

3. Identify evaluation criteria for alternatives

4. Pairwise comparisons: between pairs of criteria and between pairs of alternatives

5. Obtain global priorities of alternatives

The following subsections describe each of these in detail.

1) Step 1: Set a Goal: Suppose that the goal of test engineers is to choose the most cost-effective test case prioritization technique in application to a particular system version.

2) Step 2: Identify Alternatives: To achieve this goal, test engineers consider several different types of prioritization techniques as alternatives. For instance, test engineers could consider traditional coverage-based test case prioritization techniques, such as total block coverage based test case prioritization, and additional block coverage based test case prioritization.

3) Step 3: Identify Evaluation Criteria: As criteria, test engineers choose factors that are influential in evaluating test case prioritization techniques. For instance, test engineers could consider the cost factors that can affect the choice of techniques, such as the cost of applying test case prioritization technique or the cost of software artifact analysis.

4) Step 4: Pairwise Comparisons: Next, two types of pairwise comparisons are performed: between pairs of criteria and between pairs of techniques as we explained in Chapter 3.1. To do so, test engineers assign relative importance weights to criteria and techniques using the scale of weights they define. In this step, test engineers rely on their experiences and history data regarding the performance of test case prioritization techniques.

5) Step 5: Obtain Global Priorities: Once test engineers assign relative weights, global priorities are calculated as explained in Chapter 3.1 and this step can be automated by building an AHP tool. Based on global priorities, test engineers determine which technique they should use for the particular version of the program. Steps 2 and 3 are dependent on an organization's testing practices and environment. Figure 3.3 summarizes steps 4 and 5 graphically. As we can see from the figure, the test engineer may consider three knowledge sources for educated judgment. He studies previous empirical study on the test case prioritization technique he is using as alternatives. He obtains knowledge on the criteria he is going to consider and the relationship between the criterion and test case prioritization techniques. He investigates data of previous releases and test results history. He develops the knowledge of using prioritization techniques on versions with various degrees and types of modifications. Based on the knowledge he examines various software artifacts for the current version, and assigns relative weights for criteria and techniques. This process requires human judgment, so it is done manually. In practice, often organizations rely on human experts' opinions or experienced members' judgment when they make important technical decisions (e.g., which tools or techniques should be used), so this is not an uncommon process in software industry.

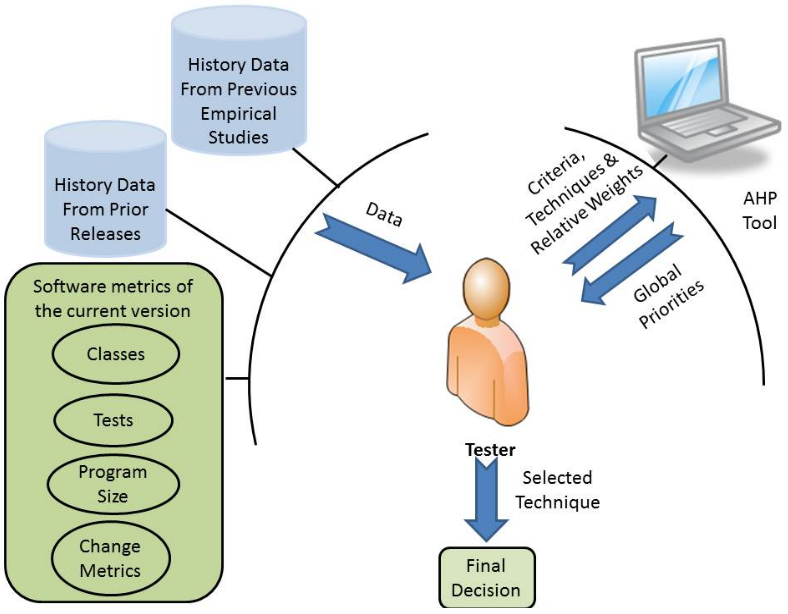

History Data From Previous Empirical Studies
History Data From Prior Releases
Software metrics of the current version
Classes
Tests
Program Size
Change Metrics
Data
Criteria, Techniques & Relative Weights
Global Priorities
AHP Tool
Tester
Selected Technique
Final Decision

Figure 3.3. AHP process

The rest of the processes can be automated. The AHP tool takes relative weights of criteria and techniques, and produces matrices shown in Figure 3.1 and 3.2. Then, finally the test engineer can decide which technique should be used based on global priorities that the tool produced.

CHAPTER 4. EMPIRICAL STUDY

To investigate the potential use of the Analytic Hierarchy Process (AHP) method in the adaptive regression testing strategy, we performed a controlled experiment considering the following research question:

RQ: Is AHP effective for selecting appropriate test case prioritization techniques across system lifetime?

The following subsections present, for this experiment, our objects of analysis, variables and measures, experiment setup and design, and threats to validity. Following this presentation, in Chapter 5 we present our data and analysis, in Chapter 6 we address threats to validity, and in Chapter 7 we discuss practical implications of the results.

4.1. Objects of Analysis

We considered five Java programs obtained from the SIR infrastructure [28] as our objects of analysis: *ant*, *xmlsecurity*, *jmeter*, *nanoxml*, and *galileo*. *Ant* is a Java-based build tool, *jmeter* is a load testing tool for client/server application, and *xml-security* provides security functionality for XML data. *nanoxml* is a small XML parser for Java, and *galileo* is a Java bytecode analyzer. Several sequential versions of each of these programs are available. The first three programs are provided with JUnit test suites, and the last two are provided with TSL (Test Specification Language) test suites [29].

Table 4.1 lists, for each of our objects of analysis, data on its associated "Versions" (the number of versions of the object program), "Classes" (the number of class files in the latest version of that program), "Size (KLOCs)" (the number of lines of code in the latest version of the program), and "Test Cases" (the number of test cases available for the latest version of the program). To study the research question we require fault data, so we utilized mutation faults provided with the programs [30]. The rightmost column, "Mutation Faults", is the total number of mutation faults of the program (summed across all versions).

Objects	Versions	Classes	Size (KLOCs)	Test Cases	Mutation Faults
ant	9	914	61.7	877	412
jmeter	6	434	42.2	78	386
xml-sec	4	145	15.9	83	246
nanoxml	6	64	3.1	216	204
galileo	16	68	14.5	912	2494

Table 4.1. Experiment objects and associated data

4.2. Variables and Measures

1) Independent Variable: To investigate our research question, we manipulate one independent variable: *test case prioritization technique application mapping strategy*, which assigns specific test case prioritization techniques to a specific sequence of versions Si, Si+1 , . . . Sj of system *S*. As test case prioritization techniques, we utilize original order (Orig: the order in which test cases are executed in the original testing scripts provided with the object programs), random order (Rand: in our experiment, averages of runs of 30 random orders), and two test case prioritization heuristics total block coverage (Tcov) and additional block coverage (Acov) prioritization techniques (explained in Chapter 2).

We consider five mapping strategies as follows:

- Tcov-all: Use of the total coverage technique across versions (a control)

- Acov-all: Use of the additional coverage technique across versions (a second control)
- Rand-all: Use of the random technique across versions (a third control)
- Orig-all: Use of the original technique across versions (a fourth control; it is used as a baseline strategy)
- AHP: Evolutionary adaptation of techniques following the AHP method described in Chapter 3. The AHP method selects the best technique among four prioritization techniques (Tcov, Acov, Rand, and Orig) for each version based on the criteria we identifies and expert's opinion. More details on how we applied AHP are described in Chapter 4.3

2) Dependent Variable and Measures: Our dependent variable is a *relative cost-benefit value* produced by applying EVOMO economic model presented in [31] (see Appendix A), using a further calculation described below (Equation 3). The cost and benefit components are measured in dollars. To determine the *relative cost-benefit* of prioritization technique *T* with respect to baseline technique *base*, we use the following equation:

$$(\text{Benefit}_T - \text{Cost}_T) - (\text{Benefit}_{base} - \text{Cost}_{base}) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3)$$

When this equation is applied, positive values indicate that T is beneficial compared to base, and negative values indicate otherwise. We used the original technique as a baseline in this experiment. This means that the Orig-all strategy functions as a baseline strategy when we consider the cost-benefit values across all versions of the program.

EVOMO1 involves two equations as shown in Equations 4 and 5: one that captures costs related to the salaries of the engineers who perform regression testing (to translate time spent into monetary values), and one that captures revenue gains or losses related to changes in system release time (to translate time-to-release into monetary values).

$$Cost = \text{salary} * \sum_{i=2}^{n}(setup(i) + obsoleteTests(i) + resultValidation(i) + missedFaults(i)) \dots\dots\dots (4)$$

$$Benefit = reveneue * \sum_{i=2}^{n}\Big(deliveryTime(i) - \big(setup(i) + obsoleteTests(i) + analysis(i-1) + runTechnique(i) + testExecution(i) + resultValidation(i) + faultDetectionDelay(i)\big)\Big) \dots\dots\dots (5)$$

Significantly, the model accounts for costs and benefits across entire system lifetimes, rather than on snapshots (i.e. single releases) of those systems, through equations that calculate costs and benefits *across entire sequences of system releases*. The major cost components that EVOMO captures are as follows: costs for applying regression testing techniques, costs associated with missed faults, costs for artifact analysis, costs of delayed fault detection feedback, and costs associated with obsolete tests.

4.3. Setup and Procedure

To measure costs of delayed fault detection feedback and costs for applying regression testing techniques, we required object programs containing faults. Similar to our early studies [14], [32], to obtain the fault data required to investigate our research question, for each version of each program we randomly selected a *mutant group* from the set of that version's mutation faults. Each mutant group contained at most 10 mutants.

To apply AHP, we followed steps described in Chapter 3. As a human tester, one graduate student who has three years of software industry experience performed the AHP processes. The student considered the following criteria to evaluate prioritization techniques:

- Cost of applying test case prioritization technique: the time required to run a test case prioritization algorithm
- Cost of software artifact analysis: the costs of instrumenting programs and collecting test execution traces
- Cost of delayed fault detection: the waiting time for each fault to be exposed while executing test cases under a test case prioritization technique
- Cost of missed fault: the time required to correct missed faults

Next, the student performed pairwise comparisons using the scale of weights as shown in Table 4.2, which has been commonly used by others [7], [11]. When the student assigned relative weights, he utilized history data regarding the performance of test case prioritization techniques observed from previous several empirical studies [14], [15], [32], [33].

To support the rest of the processes, we implemented a Java Swing-based AHP tool. The AHP tool takes relative weights of criteria and techniques, and produces local and global priorities based on the AHP algorithm [7]. Finally the student determined which technique should be used for each version of the program using global priorities.

Weight	Definition of Weight
1	equally important
3	moderately more important
5	strongly more important
7	very strongly more important
9	Extremely more important

Table 4.2. Scale of weights

Often software companies have time pressure with the product release, due to the constraint budgetary problem and competitive software market. In practice, situations in which time constraints intervene to affect product release are frequent in the software industry, and typically software companies cut back on testing activities in order to ensure timely release of their product. Further, the degree of time constraints can vary as systems evolve. For instance, for a certain release, a company could suffer more time constraints compared to other releases due to the complex feature addition or the technical personnel loss. Thus, in this experiment, we consider the situation with time constraints that vary with each version when we evaluate test case prioritization techniques.

To simulate this situation, for each of the test case prioritization techniques, we randomly assigned the level of time constraints (25%, 50%, or 75%) for each version and foreshortened the test execution process. For example, as shown in Figure 4.1,in the first set of random assignment (Run 1) we randomly pick 50% time constraint for version one (V1). It means that, we run 50% test cases of the current test suite of the version.

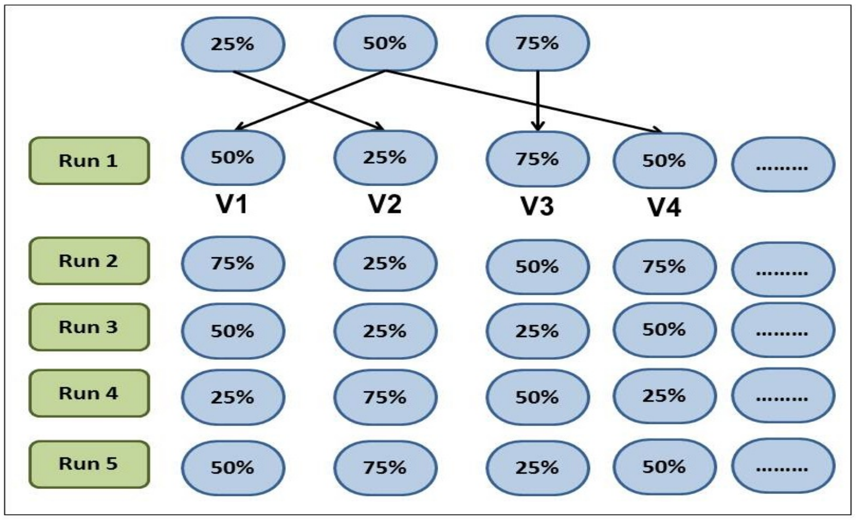


Figure 4.1. Random assignment of time constraint level

In the same way we choose 25% time constraint for version two (V2). It means that we reduce the test suite by 25%. So we run 75% of total test cases. We follow the same process for subsequent version of the program. It ends our run 1.

We ran five sets of random assignments across all versions for each program, applied the AHP processes we just explained, and collected cost-benefit values for all strategies. As we described in Chapter 4.2, for Tcov-all strategy, we run Tcov (total block coverage) prioritization technique for each version of the program under test (Figure 4.2). For Acov-all, Rand-all and Orig-all strategy, we run the Acov, Rand, Orig technique respectively for each version of the program. For AHP strategy, we conduct our AHP process and run the AHP tool. We run the technique chosen by the AHP tool. For instance, our AHP tool chose Tcov technique for version 1 and Orig technique for version 2 and so on.

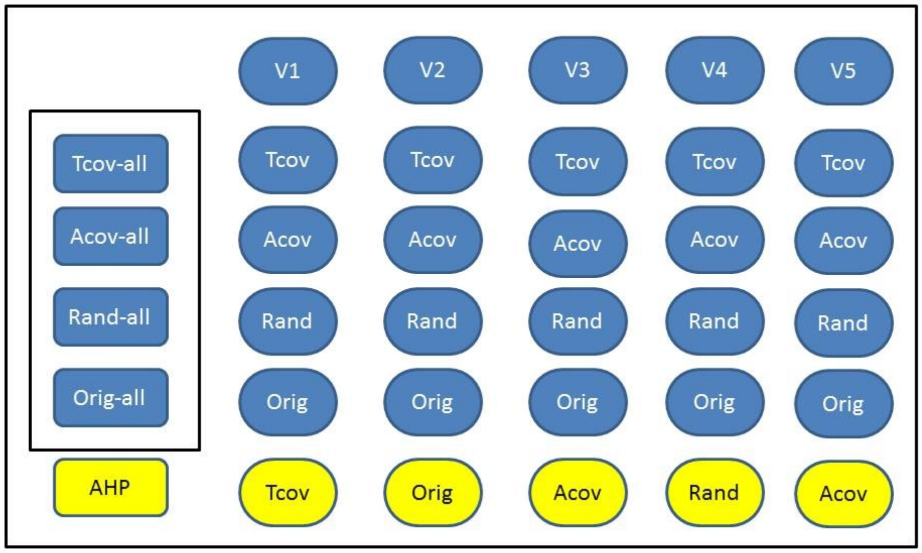


Figure 4.2. Regression testing strategy

CHAPTER 5. DATA AND ANALYSIS

In this section, we present the results of our study. We summarize the data in Tables 5.1 to 5.5. Each table consists of five sub tables, showing experiment results that were collected by running five sets of random assignments (run 1 through run 5 in the table) of three time constraint levels for each version of the program. Since the Orig-all strategy is the baseline used in our relative cost-benefit calculation, results for that strategy are not shown explicitly in the tables.

All of the data in these tables shows the relative cost-benefit value in dollar with respect to the baseline technique (Orig) as defined according to the EVOMO model. Higher values indicate greater cost-benefits. Within each sub table in the tables, the first rows are labeled with five runs, for each run listing four test case prioritization testing strategies. Rows are labeled with versions of the program and the last row ("Total") shows the sum of the cost-benefit values for all versions. Now, we describe each of these tables.

Table 5.1and 5.2 show the results for *ant*. The results vary across versions, but the total cost-benefit values indicate that the prioritization techniques selected by AHP were more cost-effective than those used by the control strategies except for one case (Rand-all in run 3 was better than AHP). In particular, the cost-benefit value gap between the AHP strategy and the two control strategies (Tcov-all and Acovall) is large, and Tcov-all was even worse than the baseline strategy in some cases (run 1 and run 4). Among the control strategies, Rand-all produced the best results.

Run 1						
Version	Time constraint	Tcov-all	Acov-all	Rand-all	AHP	
V1	50%	135.00	77.00	-40.00	Rand	-40.00
V2	25%	205.00	209.00	139.00	Acov	209.00
V3	75%	-58.00	-62.00	48.00	Rand	48.00
V4	50%	-66.00	14.00	0.00	Acov	14.00
V5	75%	-99.00	-133.00	26.00	Rand	26.00
V6	25%	-142.00	-180.00	7.00	Rand	7.00
V7	50%	-160.00	-201.00	32.00	Rand	32.00
V8	25%	-107.00	-248.00	146.00	Rand	146.00
Total		-292.00	-524.00	358.00		442.00
Run 2						
Version	Time constraint	Tcov-all	Acov-all	Rand-all	AHP	
V1	75%	367.39	231.99	101.73	Rand	101.73
V2	25%	205.26	208.95	138.99	Acov	208.95
V3	50%	-151.07	91.89	49.35	Rand	49.35
V4	75%	-155.30	-71.81	-57.90	Rand	-57.90
V5	50%	-156.83	-190.82	17.52	Rand	17.52
V6	25%	-142.06	-179.54	7.24	Rand	7.00
V7	75%	275.13	234.27	324.06	Rand	324.06
V8	25%	-107.31	-248.32	146.06	Rand	146.06
Total		135.21	76.59	727.05		796.77
Run 3						
Version	Time constraint	Tcov-all	Acov-all	Rand-all	AHP	
V1	25%	-18.94	-70.46	-98.40	Rand	-98.40
V2	50%	206.77	208.91	79.32	Acov	208.91
V3	75%	-57.83	-61.58	47.84	Rand	48.00
V4	25%	12.02	12.65	42.05	Acov	13.00
V5	75%	-99.26	-133.39	25.57	Rand	26.00
V6	50%	-37.32	-112.78	87.30	Tcov	-37.32
V7	25%	-141.54	-182.70	47.97	Rand	48.00
V8	50%	143.29	115.57	291.87	Rand	291.87
Total		7.19	-223.78	523.53		500.06

Table 5.1. Experiment results for ant: relative cost-benefit values (dollars)

Run 4						
Version	Time onstraint	Tcov-all	Acov-all	Rand-all	AHP	
V1	50%	135.11	76.90	-39.81	Rand	-39.81
V2	75%	55.41	325.98	160.74	Acov	325.98
V3	25%	-59.00	91.00	46.00	Rand	46.00
V4	50%	-65.83	13.65	-0.21	Acov	14.00
V5	25%	-145.38	-179.41	31.51	Rand	32.00
V6	75%	336.66	406.57	560.18	Rand	406.57
V7	50%	-160.07	-201.08	31.89	Rand	31.89
V8	75%	-127.91	115.08	214.92	Rand	214.92
Total		-31.01	648.69	1005.22		1031.55
Run 5						
Version	Time constraint	Tcov-all	Acov-all	Rand-all	AHP	
V1	75%	367.39	231.99	101.73	Rand	101.73
V2	50%	206.77	208.91	79.32	Acov	208.91
V3	25%	-59.00	91.00	46.00	Rand	46.00
V4	75%	-155.30	-71.81	-57.90	Rand	-57.90
V5	25%	-145.38	1.00	31.51	Rand	32.00
V6	50%	-37.32	-112.78	87.30	Tcov	-37.32
V7	75%	275.13	234.27	324.06	Rand	324.06
V8	50%	143.29	115.57	291.87	Rand	291.87
Total		595.58	698.14	903.88		909.35

Table 5.2. Experiment results for ant: relative cost-benefit values (dollars)

Table 5.3 shows the results for *jmeter*. Similar to the results on *ant*, the AHP strategy was more cost effective than the control strategies except for run 2 and run 5. Rand-all and Acov-all were better than AHP in run 2 and 5 respectively. Among the control strategies, Acov-all produced the best results (3 out of 5 runs), Rand-all performed relatively well (2 out of 5 runs), but Tcov-all was even worse than the baseline strategy in most cases. The cost-benefit values of Acov-all were significantly higher than the other two control strategies in three runs- run 3, 4 and 5. Tcov-all's cost-benefit values were considerably lower than the Orig-all strategy in run 2, 3 and 4.

Run 1						
Version	Time constraint	Tcov-all	Acov-all	Rand-all	AHP	
V1	25%	15.00	17.00	50.00	Acov	17.00
V2	50%	-51.00	153.00	93.00	Acov	153.00
V3	75%	130.00	266.00	277.00	Rand	277.00
V4	50%	121.00	31.00	5.00	Tcov	121.00
V5	25%	-196.00	-196.00	-135.00	Acov	-196.00
Total		19.00	271.00	290.00		372.00
Run 2						
Version	Time constraint	Tcov-all	Acov-all	Rand-all	AHP	
V1	50%	47.04	134.78	179.63	Acov	134.78
V2	25%	-84.55	-84.74	-6.20	Acov	-84.74
V3	75%	130.03	265.52	276.71	Rand	276.71
V4	25%	-64.17	-64.51	-142.30	Tcov	-64.17
V5	50%	-174.45	-144.16	-135.73	Tcov	-174.45
Total		-146.11	106.90	172.11		88.14
Run 3						
Version	Time constraint	Tcov-all	Acov-all	Rand-all	AHP	
V1	25%	15.00	17.00	50.00	Rand	50.00
V2	50%	-51.00	153.00	93.00	Acov	153.00
V3	25%	-66.00	22.00	-36.00	Acov	22.00
V4	75%	34.85	273.80	5.35	Acov	274.00
V5	50%	-174.45	-144.16	-135.73	Tcov	-174.45
Total		-241.61	321.63	-23.38		324.55
Run 4						
Version	Time constraint	Tcov-all	Acov-all	Rand-all	AHP	
V1	75%	-73.31	115.87	97.24	Acov	115.87
V2	50%	-51.00	153.00	93.00	Acov	153.00
V3	25%	-66.00	22.00	-36.00	Acov	22.00
V4	75%	34.85	273.80	5.35	Acov	273.80
V5	25%	-196.00	-196.00	-135.00	Acov	-196.00
Total		-351.46	368.67	24.59		368.67
Run 5						
Version	Time constraint	Tcov-all	Acov-all	Rand-all	AHP	
V1	25%	15.00	17.00	50.00	Rand	50.00
V2	75%	170.62	345.14	272.15	Acov	345.14
V3	50%	70.67	172.68	101.72	Rand	101.72
V4	75%	34.85	273.80	5.35	Acov	273.80
V5	25%	-196.00	-196.00	-135.00	Acov	-196.00
Total		95.14	612.62	294.23		574.67

Table 5.3. Experiment results for jmeter: relative cost-benefit values (dollars)

Table 5.4 shows the results for *xml-security*. As the results show, the AHP strategy was more cost-effective than the two control strategies (Tcov-all and Rand-all), but it was not better than the Acov-all strategy. Unlike the results on *ant* and *jmeter*, Tcov-all produced better results than the baseline strategy. It even produced better result than the Rand-all strategy in all five runs.

Run 1						
Version	Time constraint	Tcov-all	Acov-all	Rand-all	AHP	
V1	75%	177.00	274.00	88.00	Acov	274.00
V2	50%	26.00	117.00	-44.00	Acov	117.00
V3	25%	170.00	170.00	71.00	Tcov	170.00
Total		373.00	561.00	115.00		561.00
Run 2						
Version	Time constraint	Tcov-all	Acov-all	Rand-all	AHP	
V1	25%	37.00	38.00	6.00	Acov	38.00
V2	50%	26.00	117.00	-44.00	Acov	117.00
V3	75%	498.75	545.99	315.02	Acov	545.99
Total		561.75	700.99	277.02		700.99
Run 3						
Version	Time constraint	Tcov-all	Acov-all	Rand-all	AHP	
V1	50%	268.11	330.75	202.70	Acov	330.75
V2	75%	-48.18	13.86	-189.91	Acov	13.86
V3	25%	170.00	170.00	71.00	Tcov	170.00
Total		389.92	514.61	83.79		514.61
Run 4						
Version	Time constraint	Tcov-all	Acov-all	Rand-all	AHP	
V1	25%	37.00	38.00	6.00	Acov	38.00
V2	50%	26.00	117.00	-44.00	Acov	117.00
V3	75%	498.75	545.99	315.02	Tcov	499.00
Total		561.75	700.99	277.02		654.00
Run 5						
Version	Time constraint	Tcov-all	Acov-all	Rand-all	AHP	
V1	50%	268.11	330.75	202.70	Acov	331.00
V2	25%	21.67	22.31	-62.38	Acov	22.31
V3	75%	498.75	545.99	315.02	Tcov	499.00
Total		788.53	899.05	455.34		852.31

Table 5.4. Experiment results for xml-security: relative cost-benefit values (dollars)

Table 5.5 and 5.6 show the results for *nanoxml*. Overall, the AHP strategy outperformed all control strategies except for one case (Acov-all in run 2 was better than AHP). Similar to the results on *xml-security*, Tcov-all produced better results than the baseline strategy.

Run 1						
Version	Time constraint	Tcov-all	Acov-all	Rand-all	AHP	
V1	50%	931.00	966.00	975.00	Rand	975.00
V2	75%	468.00	778.00	596.00	Acov	778.00
V3	25%	-43.00	40.00	-27.00	Acov	40.00
V4	75%	-48.00	-50.00	1.00	Tcov	-48.00
V5	25%	-27.00	39.00	-40.00	Acov	39.00
Total		1281.00	1773.00	1505.00		1784.00
Run 2						
Version	Time constraint	Tcov-all	Acov-all	Rand-all	AHP	
V1	75%	928.09	961.65	859.79	Acov	859.79
V2	50%	564.71	789.52	682.78	Acov	789.52
V3	25%	-43.25	39.86	-26.94	Acov	39.86
V4	75%	-47.63	-49.75	0.54	Tcov	-47.63
V5	50%	450.73	541.12	453.12	Acov	541.12
Total		1852.66	2282.39	1969.28		2182.65
Run 3						
Version	Time constraint	Tcov-all	Acov-all	Rand-all	AHP	
V1	25%	-59.29	-22.67	-14.67	Rand	-15.00
V2	50%	564.71	789.52	682.78	Acov	789.52
V3	75%	163.25	657.23	525.29	Acov	657.23
V4	75%	-48.00	-50.00	1.00	Acov	-50.00
V5	25%	-27.00	39.00	-40.00	Acov	39.00
Total		593.66	1413.09	1154.39		1420.76
Run 4						
Version	Time constraint	Tcov-all	Acov-all	Rand-all	AHP	
V1	50%	931.00	966.00	975.00	Rand	975.00
V2	75%	468.00	778.00	596.00	Acov	778.00
V3	50%	509.00	563.00	482.00	Acov	563.00
V4	75%	-48.00	-50.00	1.00	Tcov	-48.00
V5	25%	451.00	541.00	453.00	Acov	541.00
Total		2311.00	2798.00	2507.00		2809.00

Table 5.5. Experiment results for nanoxml: relative cost-benefit values (dollars)

Run 5						
Version	Time constraint	Tcov-all	Acov-all	Rand-all	AHP	
V1	50%	931.00	966.00	975.00	Rand	975.00
V2	25%	-255.00	-5.00	-111.00	Acov	-5.00
V3	75%	163.25	657.23	525.29	Acov	657.23
V4	25%	-47.00	-49.00	2.00	Acov	-49.00
V5	50%	450.73	541.12	453.12	Acov	541.12
Total		1242.98	2110.35	1844.41		2119.35

Table 5.6. Experiment results for nanoxml: relative cost-benefit values (dollars)

From the Table 5.7 to 5.11 show the result for *galileo*. The results show that the AHP strategy was more cost-effective than control strategies except for one case (Acov-all in run 3 was better than AHP). Acov-all produced better results than Rand-all strategy in all five runs. Tcov-all produced worse results than the baseline strategy in all cases.

Run 1						
Version	Time constraint	Tcov-all	Acov-all	Rand-all	AHP	
V1	50%	172.00	691.00	580.00	Acov	691.00
V2	25%	-115.00	366.00	297.00	Acov	366.00
V3	50%	235.00	526.00	381.00	Acov	526.00
V4	75%	168.00	309.00	380.00	Rand	380.00
V5	25%	-3.00	56.00	9.00	Tcov	-3.00
V6	75%	-115.00	344.00	283.00	Acov	344.00
V7	50%	-186.00	216.00	130.00	Acov	216.00
V8	25%	-75.00	379.00	289.00	Acov	379.00
V9	50%	-311.00	204.00	118.00	Acov	204.00
V10	75%	-77.00	456.00	151.00	Acov	456.00
V11	25%	-4.00	575.00	528.00	Acov	575.00
V12	50%	-105.00	148.00	154.00	Acov	148.00
V13	75%	-72.00	112.00	250.00	Rand	250.00
V14	50%	-86.00	-251.00	-249.00	Tcov	-86.00
V15	75%	-80.00	211.00	293.00	Acov	211.00
Total		-654.00	4342.00	3594.00		4657.00

Table 5.7. Experiment results for galileo: relative cost-benefit values (dollars)

Run 2						
Version	Time constraint	Tcov-all	Acov-all	Rand-all	AHP	
V1	25%	-51.15	461.23	400.86	Acov	461.23
V2	50%	-114.22	368.45	251.21	Acov	368.45
V3	25%	50.89	242.29	187.15	Rand	187.15
V4	75%	-70.59	168.06	309.35	Rand	309.35
V5	50%	667.06	699.69	564.62	Tcov	667.06
V6	75%	-114.67	343.69	282.93	Acov	343.69
V7	25%	-97.62	224.29	170.15	Acov	224.29
V8	50%	-126.05	364.02	246.35	Acov	364.02
V9	25%	-248.82	223.81	145.96	Acov	223.81
V10	75%	-76.57	455.59	151.01	Acov	455.59
V11	50%	-173.85	578.74	462.42	Acov	578.74
V12	25%	-3.11	135.87	177.39	Acov	135.87
V13	75%	-72.24	111.84	250.43	Rand	250.43
V14	25%	-82.61	-216.30	-123.53	Tcov	-82.61
V15	75%	-80.07	210.66	293.32	Acov	210.66
Total		-593.63	4371.95	3769.61		4697.72

Table 5.8. Experiment results for galileo: relative cost-benefit values (dollars)

Run 3						
Version	Time constraint	Tcov-all	Acov-all	Rand-all	AHP	
V1	75%	-125.47	715.47	515.06	Acov	715.47
V2	50%	-114.22	368.45	251.21	Acov	368.45
V3	75%	-74.62	451.67	318.22	Acov	451.67
V4	25%	4.90	55.89	13.00	Rand	13.00
V5	50%	667.06	699.69	564.62	Tcov	667.06
V6	25%	-49.00	228.00	246.00	Acov	228.00
V7	75%	-76.04	284.67	250.32	Acov	284.67
V8	50%	-126.05	364.02	246.35	Acov	364.02
V9	75%	-76.23	469.01	374.21	Acov	469.01
V10	25%	-74.00	405.00	256.00	Acov	405.00
V11	50%	-173.85	578.74	462.42	Acov	578.74
V12	75%	-84.18	228.27	232.47	Acov	228.27
V13	25%	-85.00	-57.00	120.00	Rand	120.00
V14	75%	-122.05	273.29	187.65	Tcov	-122.05
V15	25%	-111.36	-125.43	64.24	Rand	-111.36
Total		-620.11	4939.75	4101.76		4659.94

Table 5.9. Experiment results for galileo: relative cost-benefit values (dollars)

Run 4						
Version	Time constraint	Tcov-all	Acov-all	Rand-all	AHP	
V1	25%	172.00	691.00	580.00	Acov	691.00
V2	75%	-115.00	366.00	297.00	Acov	366.00
V3	25%	235.00	526.00	381.00	Acov	526.00
V4	50%	4.90	55.89	13.00	Rand	13.00
V5	75%	-3.00	56.00	9.00	Tcov	-3.00
V6	50%	-39.87	272.38	261.82	Tcov	-39.87
V7	25%	-186.00	216.00	130.00	Acov	216.00
V8	75%	-75.00	379.00	289.00	Acov	379.00
V9	25%	-311.00	204.00	118.00	Acov	204.00
V10	50%	-74.00	405.00	256.00	Acov	405.00
V11	75%	-173.85	578.74	462.42	Acov	578.74
V12	25%	-105.00	148.00	154.00	Acov	148.00
V13	50%	-85.00	-57.00	120.00	Rand	120.00
V14	25%	-86.00	-251.00	-249.00	Tcov	-86.00
V15	50%	-111.36	-125.43	64.24	Tcov	64.24
Total		-953.18	3464.58	2886.47		3582.11

Table 5.10. Experiment results for galileo: relative cost-benefit values (dollars)

Run 5						
Version	Time constraint	Tcov-all	Acov-all	Rand-all	AHP	
V1	50%	172.00	691.00	580.00	Acov	691.00
V2	25%	-115.00	366.00	297.00	Acov	366.00
V3	75%	-74.62	451.67	318.22	Acov	451.67
V4	75%	168.00	309.00	380.00	Rand	380.00
V5	25%	-3.00	56.00	9.00	Tcov	-3.00
V6	75%	-115.00	344.00	283.00	Acov	344.00
V7	25%	-97.62	224.29	170.15	Acov	224.29
V8	25%	-75.00	379.00	289.00	Acov	379.00
V9	50%	-311.00	204.00	118.00	Acov	204.00
V10	75%	-77.00	456.00	151.00	Acov	456.00
V11	50%	-173.85	578.74	462.42	Acov	578.74
V12	50%	-105.00	148.00	154.00	Acov	148.00
V13	75%	-72.00	112.00	250.00	Rand	250.00
V14	25%	-82.61	-216.30	-123.53	Tcov	-82.61
V15	75%	-80.00	211.00	293.00	Acov	211.00
Total		-1041.7	4314.41	3631.26		4598.10

Table 5.11. Experiment results for galileo: relative cost-benefit values (dollars)

The total cost savings across all versions are one measure that shows the effectiveness of the strategies, but this measure can be misleading because abnormal cost-benefit values for particular version could affect the entire outcome. Thus, we examined how often the strategies produce the best results across all versions. Figure 5.1 presents bar graphs of the results. The figure contains five subfigures that present results for each of the object programs, and each subfigure contains bar graphs for four prioritization strategies showing the total number versions that produced the best results by those strategies, for the given object program and five runs. For instance, in run 1 for *ant*, Tcov-all performed best for one version (version 1 in Table 5.1) and Acov-all performed best for two versions (versions 2 and 4 in Table 5.1).

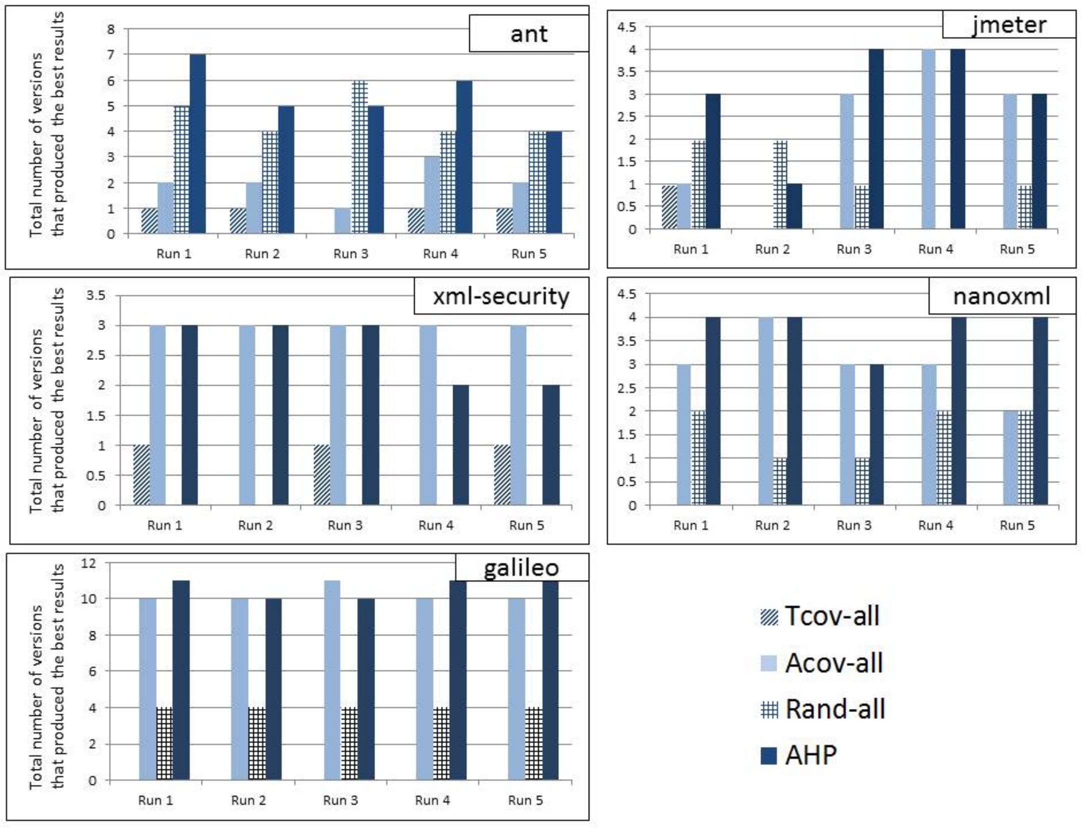


Figure 5.1. The total number of versions that produced the best results.

Overall, the AHP strategy produced the best results (20 out of 25 cases were better than the control strategies – in total, we have the 20 observed data points.) and the Acov-all strategy performed relatively well compared to other control strategies (11 out of 25 cases performed best), but the trend varied across programs.

In the cases of *ant* and *nanoxml*, the AHP strategy was consistently better than all three strategies across all runs with one exception. In the case of *xml-security*, Acov-all was slightly better than AHP.

Comparing the control strategies, Rand-all outperformed others in *ant*, and it was even better than AHP for one case (run 3). However, in other cases, Rand-all did not perform well. In particular, in the case of *xml-security*, Rand-all did not produce any single best result. Overall, Tcov-all performed worst. Only in three programs (*ant, jmeter*, and *xml-security*), it produced the best result for one version. In other programs, it did not produce any single best result.

Overall, the trends we observed from this figure are consistent with those we observed from the above five tables, but we also found some differences. While AHP outperformed 18 out 25 cases when we considered the total cost-benefit values, it outperformed 20 out of 25 cases when we considered the total number of versions that performed best. In the case of *jmeter*, the total cost-benefit of Acov-all was higher than that of AHP, but it did not perform better than AHP when we compared the number of versions that produced the best results by Acov-all and AHP.

CHAPTER 6. THREATS TO VALIDITY

This section describes the construct, internal and external validity threats to the validity of our study.

6.1. Construct Validity

Two issues involve threats to construct validity. (1) We identified four evaluation criteria to apply the AHP method mainly considering the costs that are associated with test case prioritization techniques. Other evaluation criteria, such as risks for estimated cost-benefit factors, applicability of a technique to a certain type of software artifact, and relevance to the specific testing process, could be considered. (2) The pairwise comparison value in AHP is subject to human judgment (in our case, a graduate student) and thus the results can be biased by personnel’s knowledge and experience.

6.2. Internal Validity

The inferences we made about the effectiveness of AHP could have been affected by potential faults in our experiment tools. To control this threat, we validated our AHP tool using several examples. Other tools were from SIR [28], and they have been validated through numerous experiments.

6.3. External Validity

Three issues limit the generalization of our results. (1) MCDM approach and test case prioritization technique representativeness. In this study, we considered only one type of MCDM approaches and two conventional test case prioritization techniques, so our results cannot be generalized because they are not representative of MCDM approaches and test case prioritization techniques. (2) Object program and mutation fault representativeness. The object programs are of small and medium size. Complex industrial programs with different characteristics may be subject to different cost-benefit tradeoffs. We used mutation faults generated by our mutation tool, but there is some evidence that mutation faults can be representative of real faults [30], [34]. Control for these threats can be achieved only through additional studies with wider populations of programs and faults, and different prioritization techniques.

CHAPTER 7. DISCUSSION

We now draw on the results of our analyses, together with additional consideration of our data, to derive practical implications of these results.

ART strategy results: Our results indicate that the prioritization techniques selected by AHP across the entire system lifetime can be more cost-effective than those used by the control approaches with the exception of some cases.

Through the empirical study, we observed the following trends. Overall, the AHP strategy's performance was stable across all programs for all runs, and the Acov-all strategy also produced better results compared to other control strategies and in some cases (run 4 and 5 in *xml-security* and run 3 in *galileo*), it even outperformed the AHP strategy. However, it was not as stable as the AHP strategy. For instance, on *ant* for all runs, the Acov-all strategy was worse than all other strategies, and on *jmeter*, it was close to the worst case for half of the cases.

In the case of the Rand-all strategy, it was better than the Tcov-all strategy in most cases (except for all cases in *xml-security*). However, since our results for the random technique involve averages of multiple runs, individual random orders may vary widely in performance. The Tcov-all strategy was not worst for all cases, but overall performance is not preferable to others. In particular, in several cases (two runs in *ant*, three runs in *jmeter*, and all runs in *galileo*), it was even worse than the baseline strategy.

Practical implications of the results: So far we have discussed our major findings and the results of our experiment. Now, we discuss practical implications of our results. From several prior empirical studies of prioritization [14], [15], [32], [33], we learned that typically prioritization heuristics are more cost-effective compared to control techniques, but we also learned that various factors related to software, its associated artifacts (e.g., program size, test suite size, test suite granularity, and the amount of change between versions), and organization’s testing environment could affect the relationships between techniques. Thus, adopting different types of test case prioritization techniques considering such factors is potentially a practical approach for organizations that have time pressure with the product release, due to the constraint budgetary problem and competitive software market.

To our knowledge, our study is the first attempt to investigate the effectiveness of adaptive regression testing strategy. Our proposed strategy produced promising results, and we believe that our empirical methodology and findings from our study provide insights into how such investigation can be performed and what types of MCDM approaches and evaluation criteria can be considered.

CHAPTER 8. CONCLUSION

In this paper, we have investigated an adaptive regression testing (ART) strategy that utilizes one of the multiple criteria decision making (MCDM) approaches, Analytic Hierarchy Process (AHP), and presented an empirical study assessing the ART strategy. Our results show that our ART strategy can assist researchers and practitioners in choosing cost-effective techniques across system lifetime.

As with all empirical studies, our study also has several limitations as we discussed in Chapter 6. These limitations can be addressed only through further studies of additional artifacts and regression testing techniques. For future work, we intend to investigate ART strategies further considering several aspects.

First, in this study, we chose the AHP method to implement an ART strategy, but there are many other MCDM approaches available including Weighted Sum Model and modified AHP methods. Thus, the next natural step is to investigate whether different types of approaches help improve ART strategies.

Second, in this study, we used only 4 evaluation criteria, but in order to limit threats to validity as we addressed in Chapter 6, we intend to investigate ART strategies considering other types of evaluation criteria.

Third, we considered only two test case prioritization heuristics, but we intend to investigate ART strategies that employ other types of prioritization techniques including other regression testing techniques, such as regression test selection techniques. Also, we intend to develop new regression testing techniques so that we can improve our chances of detecting faults under time-constrained situations.

REFERENCES

[1] A. Orso, N. Shi, and M. J. Harrold, “Scaling regression testing to large software systems,” in *Proceedings of the International Symposium on Foundations of Software Engineering*, vol. 29, issue 6, Nov. 2004, pp. 241–251.

[2] X. Ren, F. Shah, F. Tip, B. G. Ryder, and O. Chesley, “Chianti: A tool for change impact analysis of Java programs,” in *Proceedings of the International Conference on Object-Oriented Programming Systems, Languages, and Applications*, vol. 39, issue 10, Oct. 2004, pp. 432 –448.

[3] G. Rothermel and M. J. Harrold, “A safe, efficient regression test selection technique,” *ACM Transactions on Software Engineering and Methodologies*, vol. 6, no. 2, Apr. 1997, pp. 173–210.

[4] S. Elbaum, A. Malishevsky, and G. Rothermel, “Prioritizing test cases for regression testing,” in *Proceedings of the International Symposium on Software Testing and Analysis*, vol. 27, issue 10, Aug. 2000, pp. 102–112.

[5] A. Srivastava and J. Thiagarajan, “ Effectively prioritizing tests in development environment,” in *Proceedings of the International Symposium on Software Testing and Analysis*, vol. 27, issue 4, Jul. 2002, pp. 97–106.

[6] W. E. Wong, J. R. Horgan, S. London, and H. Agrawal, “A study of effective regression testing in practice,” in *Proceedings of the International Symposium on Software Reliability Engineering*, Nov. 1997, pp. 230–238.

[7] T. L. Saaty, *The Analytic Hierarchy Process*. McGraw-Hill, 1980.

[8] E. Triantaphyllou and K. Baig, “The impact of aggregating benefit and cost criteria in four MCDA methods,” *IEEE Transactions on Engineering Management*, vol. 25, no. 2, Feb. 2005, pp. 213–226.

[9] B. R. J. Karlsson, C. Wohlin, “An evaluation of methods for prioritizing software requirements,” *Information and Software Technology*, vol. 39, 1998, pp. 939–947.

[10] J. E. Steiguer, J. Duberstein, and V. Lopes, “The analytic hierarchy process as a means for integrated watershed management,” in *Interagency Conference on Research on the Watersheds*, Oct. 2003, pp. 736–740.

[11] S. Yoo, M. Harman, P. Tonella, and A. Susi, “Clustering test cases to achieve effective and scalable prioritization incorporating expert knowledge,” in *Proceedings of the International Conference on Software Testing and Analysis*, Jul. 2009, pp. 201–212.

[12] G. Rothermel, R. Untch, C. Chu, and M. J. Harrold, “Prioritizing test cases for regression testing,” *IEEE Transactions on Software Engineering*, vol. 27, no. 10, Oct. 2001, pp. 929–948.

[13] S. Yoo and M. Harman, “Regression testing minimization, selection and prioritization: A survey,” *Software Testing, Verification, and Reliability*, vol. 22, issue 2, Mar. 2010, pp. 67–120.

[14] H. Do, S. Mirarab, L. Tahvildari, and G. Rothermel, “The effects of time constraints on test case prioritization: A series of controlled experiments,” *IEEE Transactions on Software Engineering*, vol. 26, no. 5, Sep. 2010.

[15] S. Elbaum, A. G. Malishevsky, and G. Rothermel, “Test case prioritization: A family of empirical studies,” *IEEE Transactions on Software Engineering*, vol. 28, no. 2, Feb. 2002, pp. 159–182.

[16] A. Malishevsky, G. Rothermel, and E. S., “Modeling the cost benefits tradeoffs for regression testing techniques,” in *Proceedings of the International Conference Software Maintenance*, Oct. 2002, pp. 204–213.

[17] X. Qu, M. Cohen, and R. G., “Configuration-aware regression testing: An empirical study of sampling and prioritization,” in *Proceedings of the International Conference on Software Testing and Analysis*, Jul. 2008, pp. 75–86.

[18] A. Walcott, M. L. Soffa, G. M. Kapfhammer, and R. S. Roos, “Time-aware test suite prioritization,” in *Proceedings of the International Conference on Software Testing and Analysis*, Jul. 2006, pp. 1–12.

[19] S. Elbaum, G. Rothermel, S. Kanduri, and A. G. Malishevsky, “Selecting a cost-effective test case prioritization technique,” *Software Quality Journal*, vol. 12, no. 3, Sep 2004, pp. 185–210.

[20] M. J. Harrold, D. Rosenblum, G. Rothermel, and E. Weyuker,“Empirical studies of a prediction model for regression test selection,” *IEEE Transactions on Software Engineering*, vol. 27, no. 3, Mar. 2001, pp. 248–263.

[21] K. M. A.S. Al-Harbi, “Application of AHP in project management,” *International Journal of Project Management*, vol.19, Jan. 2001, pp. 19–27.

[22] R. N. Wabalickis, “Justification of FMS with the analytic hierarchy process,” *IEEE Transactions on Software Engineering*, vol. 7, 1988, pp. 175–182.

[23] K. E. Cambron and G. W. Evans, “Layout design using the analytic hierarchy process,” *Computers and Industrial Engineering*, vol. 20, issue 2, 1991, pp. 221–229.

[24] T. O. Boucher and E. L. MacStravic, “Multi attribute evaluation within a present value framework and its relation to the analytic hierarchy process,” *The Engineering Economist,* vol. 37, no. 1, 1990, pp. 55–71.

[25] A. Barcusa and G. Montibeller, “Supporting the allocation of software development work in distributed teams with multi-criteria decision analysis,” in *Multiple Criteria Decision aking for Engineering*, Jun. 2008, pp. 464–475.

[26] G. E. W. Gavin R. Finnie and D. I. Petkov, “Prioritizing software development productivity factors using the analytic hierarchy process,” *Journal of Systems and Software*, vol. 22, Aug. 1993, pp. 129–139.

[27] A. S. Anna Perini, Filippo Ricca, “Tool-supported requirements prioritization: Comparing the AHP and CBRank methods,” *Information and Software Technology*, vol. 51, Jun. 2009, pp. 1021–1032.

[28] H. Do, S. Elbaum, and G. Rothermel, “Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact,” *International Journal on Empirical Software Engineering*, vol. 10, no. 4, 2005, pp. 405–435.

[29] T. Ostrand and M. J. Balcer, “The category-partition method for specifying and generating functional tests,” *Communications of the ACM,* vol. 31, no. 6, Jun. 1988, pp 676-686.

[30] H. Do and G. Rothermel, “On the use of mutation faults in empirical assessments of test case prioritization techniques,” *IEEE Transactions on Software Engineering*, vol. 32, no. 9, Sep. 2006, pp. 733–752.

[31] H. Do and G. Rothermel, “Using sensitivity analysis to create simplified economic models for regression testing,” in *Proceedings of the International Conference on Software Testing and Analysis*, Jul. 2008, pp. 51–62.

[32] H. Do and G. Rothermel, “An empirical study of regression testing techniques incorporating context and lifecycle factors and improved cost-benefit models,” in *Proceedings of the ACM SIGSOFT Symposium on Foundations of Software Engineering*, Nov. 2006, pp. 71–82.

[33] G. Rothermel, R. Untch, C. Chu, and M. J. Harrold, “Test case prioritization: An empirical study,” in *International Conference of Software Maintenance*, Aug. 1999, pp. 179–188.

[34] J. H. Andrews, L. C. Briand, and Y. Labiche, “Is mutation an appropriate tool for testing experiments?” in *International Conference of Software Engineering*, May 2005, pp. 402–411.

APPENDIX. EVOLUTIONARY AWARE ECONOMIC MODEL (EVOMO)

EVOMO has two equations as shown in Equation 6 and 7:

1. To calculate the cost of applying regression testing process.

$$Cost = \mathrm{PS} * \sum_{i=2}^{n}\big(CS(i) + CO_i(i) + CO_r(i) + b(i) * CV_i + C(i) * \mathrm{CF}(i)\big) \ldots\ldots\ldots\ldots.. (6)$$

2. To calculate the benefits gained from applying the regression testing process.

$$Benefit = \mathrm{REV} * \sum_{i=2}^{n}(ED(i) - \big(CS(i) + CO_i(i) + CO_r(i) + a_{in}(i-1) * CA_{in}(i-1) + a_{tr}(i-1) * CA_{tr}(i-1) + CR(i) + b(i) * \big(CE(i) + CV_i(i) + CV_d(i)\big) + \mathrm{CD}(i))\big) \ldots\ldots\ldots\ldots (7)$$

The term and coefficients are described as follows:

- **Cost of test setup** (CS). This is the costs associated with setting up the system for testing, compiling the version under test, and configuring test drivers and scripts.
- **Cost of identifying obsolete test cases** (COi). This is the costs associated with manually inspect a version and its test cases.
- **Cost of repairing obsolete test cases** (COr). This is the costs to examine the specifications, test cases, and test drivers, and modify the test cases for the version under test.
- **Cost of supporting analysis** (CA). This is the costs to instrument programs (CAin) and collect test traces (CAtr).
- **Cost of technique execution** (CR). This is the time needed to execute a regression testing technique itself.
- **Cost of test execution** (CE). This is the time needed to execute test cases.

- **Cost of test result validation (automatic via differencing)** (CVd). This is the time needed to run a differencing tool on test outputs as test cases are executed.
- **Cost of test result validation (human via inspection)** (CVi). This is the time required by engineers to inspect test output comparisons.
- **Number and cost of missing faults** (c and CF). Regression testing technique could not find all kinds of faults in the system. It could fail to discover some of them. There is a cost associated with each missing fault. To estimate the cost of missed faults, we follow the earlier study (Do, 2006), choose 1.5 person hours to localizing and correcting one fault.
- **Cost of delayed fault detection feedback** (CD). To calculate the cost we follow an earlier study (Do, 2006). We measure the rate of fault detection of a test suite. Then, we translate this rate into the cumulative cost (in time) of waiting for each fault to be exposed while executing test cases under a particular order, defined as *delays*.
- **Revenue** (REV). We calculate the revenue by utilizing revenue values cited in a survey of software products ranging from $116,000 to $596,000 per employee (Do, 2006). Because of the small size of the programs, we consider the least revenue value mentioned, and an employee headcount of ten.
- **Programmer salary** (PS). We consider a figure of $100 per person-hour, obtained by adjusting an amount cited in (Do, 2006) by an appropriate cost of living factor.
- **Expected time-to-delivery** (ED). In our empirical work we rely on comparisons of techniques to a control suite using Equation 2; this approach cancels out the ED values because these are the same for all cases considered.