B.Sc. in Computer Science and Engineering Thesis

A Cost Effective Regression Testing Approach for Web Application

Submitted by

Sumsun Nahar Jesy
200914011

Md. Raihan Majumder
200914049

Md. Arifur Rahman
200914014

Supervised and Approved by

Dr. Muhammad Masroor Ali
Professor
Department of Computer Science and Engineering, BUET

Department of Computer Science and Engineering
Military Institute of Science and Technology

CERTIFICATION

This thesis paper titled "A Cost Effective Regression Testing Approach for Web Application" is submitted by the group as mentioned below has been accepted as satisfactory in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering on December 2012.

Group Members:

Sumsun Nahar Jesy
Md. Raihan Majumder
Md. Arifur Rahman

Supervisor:

_____

Dr. Muhammad Masroor Ali
Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology
Dhaka-1000, Bangladesh

CANDIDATE'S DECLARATION

This is to certify that the work presented in this thesis paper is the outcome of the investigation and research carried out by the following students under the supervision of Dr. Muhammad Masroor Ali, Professor, BUET, Dhaka, Bangladesh.

It is also declared that neither this thesis paper nor any part thereof has been submitted anywhere else for the award of any degree, diploma or other qualifications.

_____

Sumsun Nahar Jesy
200914011

_____

Md. Raihan Majumder
200914049

_____

Md. Arifur Rahman
200914014

ACKNOWLEDGEMENT

We are thankful to Almighty Allah for his blessings for the successful completion of our thesis. Our heartiest gratitude, profound indebtedness and deep respect go to our supervisor Dr. Muhammad Masroor Ali, Professor, BUET, Dhaka, Bangladesh, for his constant supervision, affectionate guidance and great encouragement and motivation. His keen interest on the topic and valuable advices throughout the study was of great help in completing thesis.

We are especially grateful to the Department of Computer Science and Engineering (CSE) of Military Institute of Science and Technology (MIST) for providing their all out support during the thesis work.

Finally, we would like to thank our families and our course mates for their appreciable assistance, patience and suggestions during the course of our thesis.

ABSTRACT

Regression testing is an expensive testing procedure utilized to validate modified software. During regression testing, a modified system is retested using the existing test suite. Because the size of the test suite may be very large, testers are interested in detecting faults in the system as early as possible during the retesting process. Regression test selection techniques attempt to reduce the cost of regression testing by selecting a subset of a program's existing test suite. In this paper we have briefly discussed two of the selection techniques and choose the safe algorithm technique. The previous researches indicate that safe regression test selection can be cost-effective, but that its costs and benefits vary widely based on a number of factors. In order to carry through the regression testing quickly and effectively, we have proposed a regression testing approach from a large test suite using hybrid technique based on safe selection algorithm and prioritization based on cost criterion. We examined some prioritization strategies and develop our proposed algorithm to improve the rate of fault detection for web applications. We propose a new hybrid regression testing approach that select less test cases and then prioritize them according to their cost, so that the regression testing for web application become more time effective and less expensive.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATION

SOAP : Simple Object Access Protocol

SOA : service-oriented architecture

WSDL : Web Services Description Language

UDDI : Universal Description, Discovery and Integration

TPG : Task Precedence Graph

TLTS : Timed Labeled Transition System

Ccv : credit card validation

LIST OF SYMBOLS

S : initial pages in the website

D : deleted pages

C : modified pages

A : added pages

U : unchanged pages

CHAPTER 1
INTRODUCTION

1.1 Overview

The development of web applications has received significant attention in the past few years. The use of web services also provided a common communication infrastructure to communicate through the internet, and enabled developers to design applications that can span different operating systems, hardware platforms and geographical locations. Thus the design and the maintenance of reliable web applications and web services should be considered seriously. No matter how well conceived and tested before being released, web applications will eventually have to be modified in order to fix bugs or respond to changes in user specifications. During maintenance of evolving software systems, their specification and implementation are changed to fix faults, to add new functionality, and to change the existing functionality. Regression testing must be conducted to confirm that recent program changes have not adversely affected existing features and new tests must be conducted to test new features. Studies show that regression testing accounts for 80Regression test selection techniques reduce the cost of regression testing by selecting an appropriate subset of the existing test suite, based on information about the program, modified version, and test suite. New software development processes such as extreme programming also promote a short development and testing cycle and frequent execution of fast test cases. Running all of the test cases in a test suite, however, can require a large amount of effort, time and cost. For example, one of our industrial collaborators reports that for one of its products of about 20,000 lines of code, the entire test suite requires seven weeks to run. Therefore, there is a clear need for another technique to reduce again the test cases form selected test suite. A cost effective technique is prioritization technique that has the potential to be more effective when a test suites' allowed execution time is known, particularly when that execution time is short. There are two benefits brought by prioritization technique. First, it provides a way

to find more bugs under resource constraint condition and thus improves the reliability of the system quickly. Second, because faults are revealed earlier, engineers have more time to fix these bugs and adjust the project schedule. Also it can avoid the drawbacks that can occur when we apply test suite minimization technique which discard test cases.

1.2 Thesis Goal

In this paper, we have used two techniques, test suite selection and prioritization for regression testing. We first select a regression test selection algorithm that executes any of the modifications in the old version of a web application. Then, depending on the available resources, a tradeoff between what we should ideally do in regression testing and what we can afford to do is applied to determine which tests, among those necessary, should be re-executed first, and which ones have lower priority or are to be omitted from re-execution. That is our work is an attempt to develop an algorithm for reduction of test cases from a large test suite using selection algorithm that will reduce both time and effort and produce optimal results. Hence, we propose a hybrid regression testing optimization technique, using regression test selection and prioritization, which reduces the overall cost of regression testing phase by selecting subset of test suite which was developed throughout the life of the software.

1.3 Thesis Outline

In this paper, we have proposed a hybrid approach for regression testing using a safe selection algorithm and a prioritization algorithm. In this regard in Chapter 2 we discussed about the background study of web applications. Following Chapter 3 describe about regression testing. In Chapter 4 we have briefly described two method of regression testing selection techniques. We have presented our proposal and algorithm in Chapter 6 and conclude our paper in Chapter 7.

CHAPTER 2
BACKGROUND

2.1 Web Services Overview

Web Services were defined differently by vendors, researchers, or standards organizations. In our study, we define Web Services as self-contained component-based applications, residing on different servers, and communicating with other applications by exchanging XML messages wrapped in SOAP interfaces. Web services infrastructure is based on service-oriented architecture (SOA) that involves three kinds of participants: service providers, service requesters and a service broker (Figure 1). A service provider publishes services to a service broker. Service requesters find required services using a service broker and then bind to them [1]. This infrastructure uses the following standards to make web services function together: Web Services Description Language (WSDL), Universal Description, Discovery and Integration (UDDI), The Extensible Markup Language (XML), and Simple Object Access Protocol (SOAP). The WSDL file is a description of how to access the web service and what operations this service can perform. On the service broker, the UDDI registry holds the specification of services and the URL that points to the WSDL file of services. The service requester searches for a web service in the UDDI registry, then binds to it, and transmits massages and data using XML wrapped in SOAP interfaces.

2.2 Modeling Web Application

Web-based software systems are constructed by integrating different interacting-components from a variety of sources. The schedule of invoking the interacting-components is restricted by the requirements specification of the web application and by time constraints. These components interact with the main application as well with other components by exchanging messages (actions) that might also involve timing constraints. Many researchers have

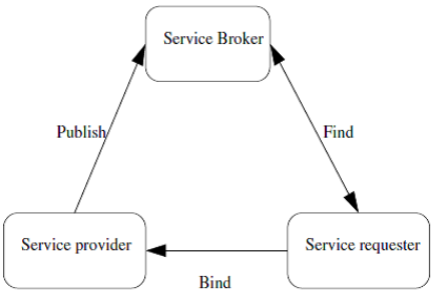

Figure 1: Web service architecture

Figure 2.1: web service architecture

model web applications in different model, though basic view or infrastructures are similar. We here discussed two types of web application representations in following paragraphs.

2.2.1 Web Application Representation- 1

According to Xu, Chen, Jiang, Huowang [2] web applications consisted of multiple pages, and pages contact with each other by the relationships of hyperlink dependent and data dependent. These pages are mostly described in HTML or other script languages. These form a complicated system, and any change to one of the contents in the system may influence others. The number of hyperlinks in one page is called as its out-degree, and the number of all the hyperlinks, which point to one page, is called as the page's in-degree. A Web page is consisted of many categories of elements, and the changes to some elements cannot influence others, such as the adjustment for page layout, or the literal changes to the content, etc. One page can also be consisted of several pages, i.e., there exists including relationship among these pages. But in fact, the pages are independent (except framework) and they can be disposed as several single pages. Otherwise, there may be indirect-dependent between two pages caused by visiting the shared variables. As the changes to some elements may influence other pages, these dependent relationships are divided into two categories: direct-dependent and indirect-dependent. Suppose S= initial pages in the website, and after adjusting the structure of the website, D= deleted pages, C= modified pages, A= added pages, U= unchanged pages, and then U=S D C. The possible dependent relationships among all the pages are divided into four categories, shown as below:

- Before the website structure adjustment, there exist hyperlink (or submit) dependent relationships E1 from the set U to the sets D and C, i.e, E1= U$->$(D+C); where "$->$" means the hyperlink (or submit) dependent relationships.

- Before the website structure adjustment, there exist hyperlink (or submit) dependent relationships E2from the sets D and C to the set U, i.e., E2=(D+C)$->$U

- After the website structure adjustment, there exist hyperlink (or submit) dependent relationships E3from the sets C and A to the set U along with the relationships between C and A, i.e., E3= (C +A)$->$U, C$->$A, A$->$C

- After the website structure adjustment, there exist data dependent relationships E4 from the sets C and A to the set U along with the relationships between C and A i.e., E4= (C+A)$=>$ U, C$=>$A, A$=>$C; Where "$=>$" means the data dependent relationships.

  In order to obtain the indirect-dependent relationships caused by data transfer or sharing variables we should go deep into the insides of the pages and obtain the dependent set of a certain changed page element by analyzing the variable's definition-usage relationship.

2.2.2 Web Application Representation- 2

Another web application representation is a two-level abstract model [3]. The first level models the interaction of components with the main application. The second level models the internal behavior of each component in the system. This hierarchical model helps in minimizing the state explosion problem. The functional behavior of a web system could be represented as a Task Precedence Graph (TPG). However, web applications are composed of components [3] that interact by exchanging messages restricted by timing constraints, first level of abstraction models a web applications as an input-complete Timed Labeled Transition System (TLTS), where each node in the TLTS is an abstract representation of a single component in the system that models the behavior of its software modules, and an edge joining two nodes represents the flow of actions (transitions) between components. Every edge is labeled with an action and its corresponding timing constraint. A TLTS can be defined as follows: Definition 2.1 (Timed Labeled Transition System (TLTS)) An TLTS

is defined by M = (S, A, C, T, s0) where S is a finite set of states, s0 is the initial state, and A is a set of actions. A is partitioned into 2 sets: AI is the set of input actions (written ?i), AO is the set of output actions (written ?o). C is a set of clocks. T is a transition set having the form {Tr1.T r2...T rn}; Tri = < s; a; d; EC; Cs >, where: s ∈ S and d ∈ S are starting and destination states; a ∈ A is the action of the transition; EC is an enabling condition evaluated to the result of the formula a b where ∈ { <, >, ≤, ≤, = } or to a constant valued either true or false; Cs is a set of clocks to be reset at the execution of transition Tri. Definition 2.2 (input-complete) A TLTS M is said to be input-complete if all states accept any input a ∈ AI. A TLTS will be input-completed by adding to each controllable state (a state whose action is only input) a loop labeled by all complementary actions in the input alphabet AI.

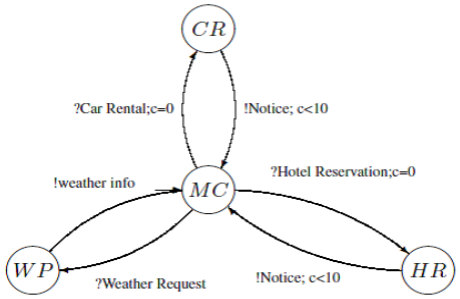

Figure 2: An example of TLTS representing a simple travel agency web application

Figure 2.2: An example of TLTS representing a simple travel agency web application

Figure 2 illustrates a TLTS representing a simple travel agency web application that is composed of four components: Main Component (MC), Hotel Reservation (HR), Car Rental (CR), and Weather Prediction (WP). The second level of abstraction models every single component in the web application. In this level, each component is modeled as an input-complete Timed Labeled Transition System (TLTS). Each state in the TLTS represents a state of the modeled component. An edge joining two states is labeled with an action and its corresponding timing constraint. It represents a transition from one state to another. Figure 3 shows an example of TLTS representing a simple hotel reservation component (HR) with initial state s0. A transition is represented by an arrow between two states and labeled by the action, the timing constraint and clocks to reset (action; EC; Cs).

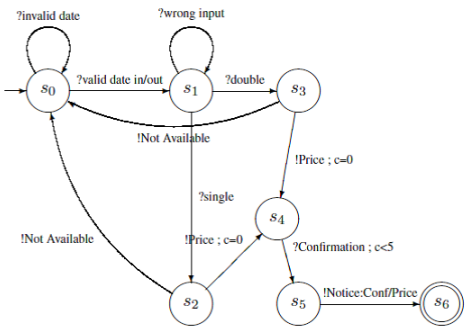

Figure 2.3: A TLTS representing simple hotel reservation

2.3 Test Suite Generation

Regression testing technique for web services is based on testing web services. In the following subsections we briefly discuss two works on testing web services. Then, we present the regression test case selection algorithm on next Chapter.

2.3.1 Test Suite-1

Xu, Chen, Jiang, Huowang [2] shortly told about test suite generation for web application of slicing based representation described in section 2.2.1. There are direct-dependent and indirect-dependent relationships among the sets U, S, A, D mentioned in section 2.2.1. The testing for direct-dependent relationships generated by hyperlinks is easy, for we only need to validate whether these pages are reachable and effective. In order to obtain the indirect-dependent relationships caused by data transfer or sharing variables, we look through insides of the pages and obtain the dependent set of page elements by analyzing the variable's definition-usage relationship, thus we can gain the definition-usage paths about variables and generate the testing cases for the web application.

2.3.2 Test Suite-2

In another work [3], Tarhini, Fouchal, and Mansour presented testing technique that guarantees the availability of services in web applications modeled as. It selects and then associates all suitable web services to our web application before invocation time; moreover, it suggests testing the functionality of the web service integrated in the web application by executing

three sets of test cases generated from (1) the WSDL files and (2) the specification of both the component fulfilled by a web service and (3) the specification of the whole web application. The links to all selected suitable web services are saved into a log file associated with the component to be fulfilled by the web service. The log file contains the urls of all suitable web services and the set of test sequences used to test this component, it also contains a priority ranking number for each of these services. The first set tests the adequacy of the web service independently. It is generated based on boundary value testing analysis [4] bounded by the limitations defined in the XML schema. The second set of test sequences is used to test the behavior of the web service individually. Thus, it is generated by traversing all loop-free paths going from the initial state of the TLTS specification representing the component to be fulfilled. The third set of test sequences tests the interaction of the web service as a part of the web application. Thus, this set is generated by traversing all loop free paths going from the initial state of the TLTS representing the web application including the loop-free paths of the TLTS representing the inner actions of the composed components. Next, test sequences and test histories are created. One test history is created for the web system; it consists of the third set of test sequences generated above, and an execution history for each test sequence. An execution history consists of a list of components and their internal states that experienced this test sequence. Other test histories are created for each composed component. Those test histories are attached to log files found in their corresponding component. A component's test history consists of only test sequences that experienced this component.

CHAPTER 3
REGRESSION TESTING

3.1 Definition

Regression testing is type of testing carried out to ensure that changes made in the fixes or any enhancement changes are not impacting the previously working functionality. It is executed after enhancement or defect fixes in the software or its environment. The word regress means to return to a previous, usually worse, state. Regression testing refers to that portion of the cycle in which a program P' is tested using test set T to ensure that not only does the newly added or modified code behaves correctly, but also that code carried over unchanged from the previous version P continues to behave correctly.

3.2 Typical Steps for Regression Testing Process

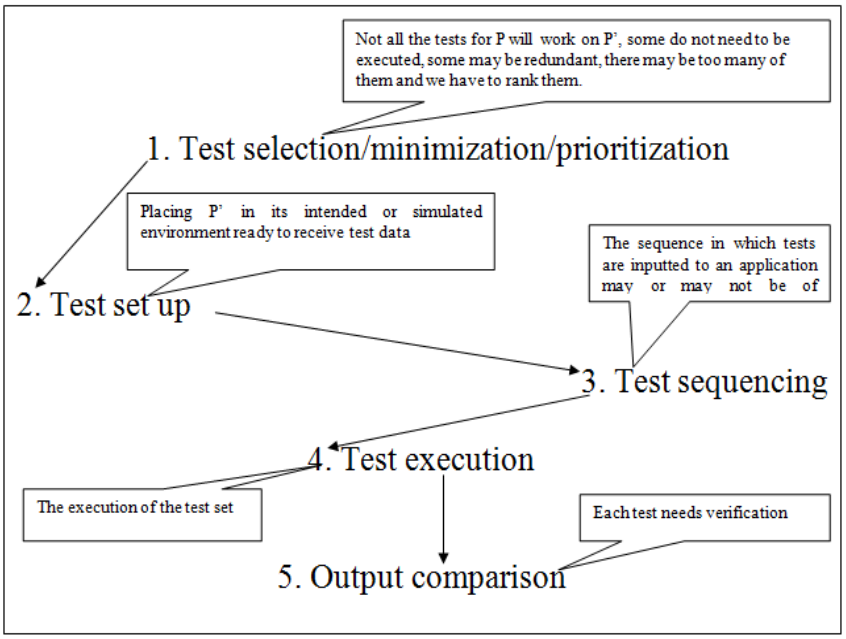

Figure 4: a subset of tasks in regression testing process

Figure 3.1: a subset of tasks in regression testing process

A regression testing process is exhibited in Figure 4. The process assumes that P' is available for regression testing. There is usually a long series of tasks that lead to P' from P. these tasks, not shown in Figure 4, include creation of one or more modification requests and the actual modification of the design and the code.

3.3 Regression Testing Approaches

Regression testing refers to selecting tests from the test suite generated during the initial development phase and to adding new tests to address enhancements and additions. One regression testing strategy is retest all which reruns every test in the initial test suite. This approach is normally very expensive and requires a lot of time. An alternative approach is to select a random subset of tests, which might be unreliable. Therefore, regression testing is a challenging task that should be both economic and reliable. The first task of regression testing process is to select the test suite T' from T. This is the most challenging task among the all steps of process. This task can be done by three ways: selection, minimization and prioritization. Sometimes these three approaches can be bind together for developing test suite T'. Again they can be applied individually on T.

CHAPTER 4
SELECTION OF REGRESSION TESTING

4.1 Regression Test Selection Problem

The first step of regression testing process is tests selection. Most of the cost of the regression testing is dependent on selecting tests from validated tests for modified version of a web application. In this Chapter we will discuss about selection techniques of regression testing.

4.2 Selection Methods

The selection of suitable test cases from T can be made in different ways and a number of regression-testing methods have been proposed. These methods are based on different objectives and techniques, such as: procedure and class firewalls [5, 6]; semantic differencing [7]; textual differencing [8]; slicing-based data-flow technique [9,10]; test case reduction [11,12]; and safe algorithm based on program's control graph [13]. In following two sections we will briefly discuss two methods of selection tests.

4.2.1 Based on Slicing

Xu, Chen, Jiang, Huowang [2] give a web regression testing method focused on the changes to the page elements. This method is based on the idea of slicing and only abstract related information to simplify problems, i.e., they started this work with analyze the changes themselves, then obtained the related contents with the changes by the Forward and Backward Search Method, at last gain the dependent set to the changes and generated the complete and effective testing cases for the web regression testing. The method description and the testing steps are briefly shown as below:

- Step 1: Scan the whole page that contains the changed variable, and record the sentences which have variables' definitions or usages, written as <sentence number, (Def(or Use) variables)*>, and "$*$" means occurring one time or more, since there may be several variables defined or used in one sentence.

- Step 2: Based on the records in Step 1, we can find out the definition usage set of the changed variable, written as <variable, DefSet: numbers of the sentences that define the variable, UseSet: numbers of the sentences that use the variable>, and this form can be seen as the extension of the traditional slicing criterion.

- Step 3: The definition-usage paths of the variable can exist inside process, between processes, inside a page, or between pages, and the level is extended one by one. If the related sentences of the variable's definition-usage are inside the same process, then we begin with the definition sentence, and execute the sentences in the order of first sequence, next branch, and then cycle till we reach the usage sentence. If the related sentences of the variable's definition-usage are between processes, we still need to add a process calling edge to join the two paths inside each process.

- Step 4: The definition-usage paths between pages are concerned with two pages, so we must deal with the other page in the Step1-3 also. Then we can add the related hyperlink to the existed paths in the two pages, such produce a complete path between the two pages. Among them, it is easier to search the usage set from the definition set of the variable backward, which only need traverse all the hyperlinks of the current page; while it is more difficult to search the definition set from the usage set of the

variable forward. But since we have added the in-page records in each page before, we can realize the reverse search.

- Step 5: Since the dependent relationships among variables have the transfer property, i.e., the change to a variable can not only influences the definition-usage of themselves, but also all the variables that rely on this one. These definition-usage relationships of the "infected" variables should be test also, and the method of generating testing paths is the same with before.

- Step 6. Record all the paths generated above and thus can generate the testing suit for the variable. Based on this testing suit, we can go along with the relevant testing, such as the page reachable testing, hyperlink effective testing, variable values testing, etc. At last, we can validate the functional correctness of the related pages.

4.2.2 Based on Safe Algorithm

Tarhini, Fouchal, and Mansour presented a safe regression testing technique [14] that is used to retest the web system whenever it is modified. They classify modifications to web service based applications into the following types: (a) Type-1: integrating a newly established web service into the application, (b) Type-2: adding, removing or fixing an operation or a timing constraint in an existing component, (c) Type-3: modifying the specification (operations or timing constraints) of the web application. Consider a web application ω and its modified version ω'. After any of the above modifications, we need to validate ω' by using the set of test sequences T, found in Test Suite-2, used previously to test the web application ω. Thus, we firstly identify all modifications done to ω, and then select a set of tests T'$\subseteq$ T that may reveal modification-related errors in ω'. Moreover, a new test set may be created to test required changes in the specification of the web application or any of its composed components. The algorithm for selecting the test set T' is shown following paragraph. This algorithm is used for Type-2 modification, which includes (a) fixing a time condition or action, (b) removing an operation, or (c) adding an operation. It is also used for part of Type-3 modification. The algorithm is presented in Figure 5. The input to the TestSelect algorithm

Algorithm: TestSelect

Input: TLTS for ω', Test set T, Test history.

Output: Test set T', updated Test history.

- *Step 1: Complete the TLTS for ω' to satisfy the Definition 3.2.*
- *Step 2: Generate Test set T' all for ω' by traversing all acyclic paths of TLTS ω' from the initial state.*
- *Step 3: Generate T' and update the test history as follows:*
  - *All test sequences found in T'_{all} and not found in T are executed, that is, add to the retest set T'. Thus $T' = T'_{all} \setminus T$.*
  - *All test sequences found in T and not found in T'_{all} are deleted from the test history.*
  - *All test sequences found in both T and T'_{all} are kept in the test history but not re-executed.*
  - *Add test sequences in T' to the test history.*

Figure 5: Algorithm to generate test set T'.

Figure 4.1: Algorithm to generate test set T'

presented in Figure 5 is the modified TLTS version for ω′, the previously generated test set T for ω, and the test history. The output is a selected set of test sequences T'⊆ T that is believed to reveal modified-related errors if executed on ω′, and the updated test history. The set T must include all test sequences that (1) traverse the newly added state in TLTS ω′, (2) traverse all newly added edges in TLTS ω′, and (3) traverse all edges with modified labels in the TLTS ω′. Thus, T' is generated by finding all acyclic paths in the input-complete TLTS of ω′ and not in the input-complete TLTS of ω. The test set T' is able to experience any of the changes like fixing, deleting or adding of a timing constraint or an action. To illustrate, we consider one of these changes in following:

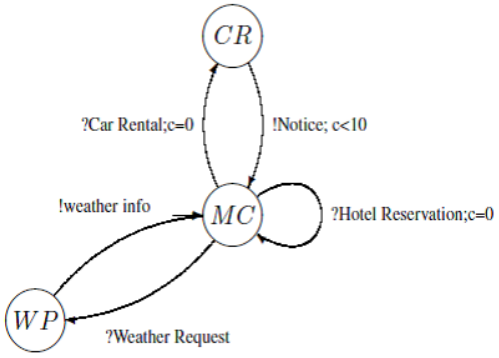

Figure 4.2: A modified TLTS for the original travel agency web application presented in figure-2

Fixing a condition or an action: Consider the TLTS ω in figure 6. Assume the time restriction in the transition <!Notice; c<10;− > is changed to (c<5). Thus, we get a modified version ω′ with time restriction <!Notice; c<5;− >. The set T generated from the original TLTS ω is:

T1: <?Hotel Reservation;c=0;− >

T2: <?Car Rental;c=0;− >.<!Notice;c<10;− >

T2: <?Weather Request;c=0;− >.<!Weather Info; −;− >

The set T' all generated from modified TLTS ω′ is:

T'all1 : <?Hotel Reservation;c=0;− >

T' all2: <?Car Rental;c=0;− >.<!Notice;c<5;− >

T'all3: <?Weather Request;c=0;− >.<!Weather Info; −;− >

Thus, the set T' = T'all T is:

T' 1: <?Car Rental;c=0;− >.<!Notice;c<5;− >

CHAPTER 5
PRIORITIZATION OF REGRESSION TESTS

5.1 Definition

Test case prioritization techniques schedule test cases for execution in an order that attempts to increase their effectiveness at meeting some performance goal. We formally define the test case prioritization problem as follows:

Definition 5.1: *The Test Case Prioritization Problem:*

Given: T: a test suite. PT: the set of permutations of T, and f: a function from PT to the real numbers.

Problem: Find $T' \in PT$ such that $(\forall T'')\ (T'' \in PT)\ (T'' \neq T')\ [f(T') \geq f(T'')]$.

In this definition, PT represents the set of all possible prioritizations (orderings) of T, and f is a function that, applied to any such ordering, yields an *award value* for that ordering. (For simplicity, and without loss of generality, the definition assumes that higher award values are preferable to lower ones.)

5.2 Importance of Prioritization

Regression test prioritization is often performed in a time constrained execution environment in which testing only occurs for a fixed time period. For example, many organizations rely upon nightly building and regression testing of their applications every time source code changes are committed to a version control repository. There are many possible goals of prioritization, including the following:

- Testers may wish to increase the rate of fault detection of a test suite that is, the likelihood of revealing faults earlier in a run of regression tests using that test suite
- Testers may wish to increase the coverage of coverable code in the system under test at a faster rate, allowing a code coverage criterion to be met earlier in the test process.
- Testers may wish to increase their confidence in the reliability of the system under test at a faster rate.
- Testers may wish to increase the rate at which high-risk faults are detected by a test suite, thus locating such faults earlier in the testing process.
- Testers may wish to increase the likelihood of revealing faults related to specific code changes earlier in the regression testing process.

  Besides meeting those goals, cost reduction and time constraints are also important issue for using prioritization methods in regression testing. While T' is the selected subset of T, it might be overly large for testing P'. One might not have sufficient budget and time to execute P' against all tests in T'. While test minimization algorithms could be used to further reduce the size of T', this reduction is risky. Tests removed from T' might be important for finding faults in P'. In such situations, one could apply techniques to prioritize tests in T' and then use only the top few high-priority tests.

5.3 Prioritization Methods

There are many methods/ techniques have been introduced for regression testing prioritization problem, like: time aware test suite prioritization [15], using system models [16], code coverage based technique [17], dynamic prioritization [18], genetic algorithm [19], using cost criterion etc. in our work we have used a prioritization method based on cost criterion.

CHAPTER 6
OUR PROPOSED REGRESSION TESTING METHOD

6.1 Our Proposal

After studying different regression test selection techniques we found that cost test can be reduced more if we reschedule the test sequences. That means applying prioritization techniques on selected test suite T' will be more cost and time effective. Moreover it can help tester in early fault detection in limited time constraint. So we have proposed to integrate a prioritization method within a regression test selection technique for web application.

6.2 Selected Test Selection Technique

In this paper, we have discussed two of the most effective regression test selection technique in section 4.2.1 (based on slicing) and 4.2.2 (based on a safe algorithm). Between them we have choose the selection technique based on a safe algorithm by Tarhini, Fouchal, and Mansour [14]. The motivation behind choosing this technique is that it is comparatively easy to understand, easy to implement and above all, it is little more cost efficient than slicing based technique. So we did our work based on that algorithm.

6.3 Prioritization Algorithm for Web Application

We have worked on a prioritization algorithm based on a cost criterion derived from residual coverage [20]. The test case that covers more transitions/ functions of a single component has lower cost and it should be executed first. Similarly in the case of first level abstraction of web application, the test case that covers most of the components, connected to main applications, have lower cost. In following section, we will derive the algorithm for test cases of single component of web application. But it is also applicable for whole web

application as first level abstraction.

6.3.1 Algorithm

Suppose P' is the modified version of component P. T' is our selected test suite from T by safe algorithm. Let E' is the set to transitions/ functions, actually called at least once during the execution of P' against T'. C(X) be the number of transitions/ functions that remain to be covered after having executed P' against tests in set X $\subseteq$ T'. Initially C ({ }) = |E'|. Here C(X) is the residual coverage of P' with respect to T'. The cost of executing P' against a test t in T' is the number of transitions/ functions that remain uncovered after execution of P' against T'. Thus, C(X) reduces or remains unchanged as tests are added to X.
The prioritization algorithm is presented in figure 7. Here the procedure PrioTest computes the residual coverage for all test in T' and determines the next highest-priority test. This procedure is repeated until all tests in T' have been prioritize.

Algorithm: PrioTest Input: T': set of regression tests for modified component P' selected by Selection Algorithm (Figure 5) TransCov: set of transitions in P' covered by tests in T'. Cov: Coverage vector such that for each test t$\in$T', Cov(t) is the set of transitions/ functions covered by executing P' against t. Output: PT: a sequence of tests such that - (i) each test in PT belongs to T', (ii) each test in T' appears exactly once in PT, (iii) tests in PT are arranged in the ascending order of cost. /* PT is initialized to test t with the list cost. The cost of each remaining test in T' is computed and the one with the least cost is appended to PT and not considered any further. This procedure continues until all test in T' have been considered. */

- Step 1: X'=T'. Find t$\in$T' such that |Cov (t)| $\leq$ |Cov(u)|,m for all u $\in$ X, u$\neq$t.
- Step 2: set PT = $<$ t $>$, X' = X' $\setminus$ {t}. Update TransCov by removing all transitions/ functions covered by t, from it. Thus TransCov = TransCov $\setminus$ Cov (t).
- Testers may wish to increase their confidence in the reliability of the system under test at a faster rate.
- Step 3: repeat the following steps while X'$\neq$ ϕ and TransCov$\neq\phi$.

3.1: compute the residual coverage for each test t $\in$ T. C(X) =|TransCov $\setminus$ (Cov (t) $\cap$ TransCov)|. C(X) indicates the count of currently uncovered transitions/ functions that will remain uncovered after having executed P' against t.

3.2: find test t $\in$ X' such that C (t) $\leq$ C (u), for all u $\in$ X', u $\neq$ t. If two or more such tests exist then randomly select one.

3.3: update the prioritized sequence, set of tests remaining to be examined and transitions/ functions yet to be covered by test in PT. PT = append (PT, t), X'= X' $\setminus$ t and TransCov= TransCov $\setminus$ Cov(t).

- Step 4: Append to PT any remaining tests in X'. All remaining tests have the same residual coverage which equals | TransCov |. Hence these tests are tied. Random selection is one way to break the tie.

  End of the procedure.

  Figure 7: Proposed Prioritization algorithm.

6.3.2 Example

Let the specification of the web application, discussed in Figure 2 has been modified. A change can be either adding or removing an operation/ component in the web application. To illustrate, assume a new component "credit card validation" (Ccv) is added to the web application so that the hotel reservation HR and car rental CR components can use it to validate credit cards. Figure 8 shows the TLTS of the modified web application and the TLTS of the component Ccv. With this modification we need to coordinate between the newly added web service Ccv and the components HR and CR. This is done by searching the UDDI registry again to find the perfect match that coordinates between composed web services.

The three sets of test sequences that make up test suite T? needed to test the modified web application are generated by TestSelect algorithm (Figure 7) as follows:

- It generates first set of test sequences to test only the newly added web service. This is done by parsing the corresponding WSDL of that service to get the data types and boundaries of the input parameters and output information. These test are mandatory,

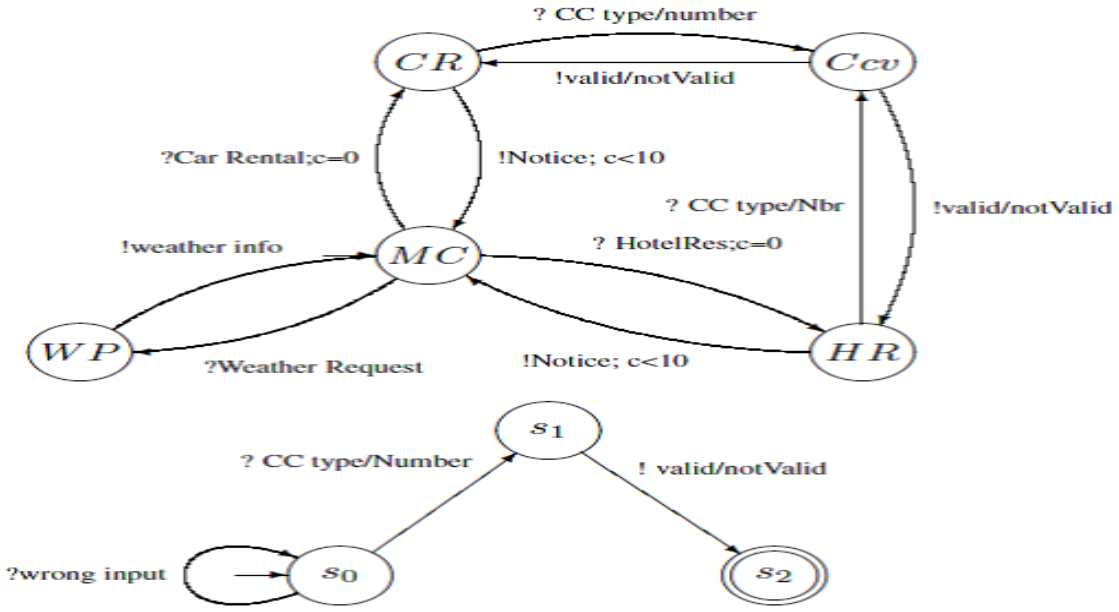

Figure 6.1: TLTS of the modified travel agency application and TLTS of the Ccv component

so that we don't apply prioritization on these test set.

- The second set of test sequences is generated by traversing all paths starting from the initial state of the TLTS representing the newly added component. For example, the second set of test sequences T', for credit card validator web service generated from the TLTS shown in figure 7 looks like: t'1: <?wrong input;-;− > t'2: <?CC type/ Number;-;− >.<! Valid/ notValid;-;− >

  So we have selected regression test set T' = {t'1, t'2}. Let the input functions, transitions and other functions like "<?CC type/Number;-;− >" be denoted by integer like:
  t'1: 1
  t'2: 2,3
  Now we made a table showing number of functions covered by test cases as follows:

Table 6.1 number of functions covered by test cases

Test (t)	Functions Covered	Cov(t)
t'1	1	1
t'2	2,3	2

Let us apply our PrioTest algorithm on T' to obtain prioritized list of tests. The inputs to PrioTest are: T', TransCov = {1, 2, 3 }, and the test coverage vectors for each test in T' as in table above.

Step 1: X'= {t'1, t'2}. Here t'2 covers more number of functions (2) in X' and hence has the least cost.

Step 2: PT= < t'2 >. X'= { t'1 }, TransCov = {1}.

Step 3: We continue the process as X' and TransCov is not empty.

Step 3.1: compute the residual coverage for each test in X'. Hence here left only one test t'1. C (t'2) = | { 1 } \ ({ 1 } ∩ { 1 }) | = ϕ

Step 3.2: as only one test has left and it covers the rest one functions so we must select this.

Step 3.3: PT = < t'2, t'1 >. X' = ϕ, TransCov = ϕ.

Step 4: as there no test is remained, we stop the procedure.

Output: PT = < t'2, t'1 >

Here we had only two tests and they are prioritized.

- o The third set of test sequences is generated using the algorithm TestSelect, consists of all paths in the web application's TLTS that traverse all components affected by the new modification, including the inner paths of the TLTS representing those components. The TLTS of the composed hotel reservation HR and Credit card validation Ccv components is shown in figure 9. A sample of the third set for the modified web application shown in figure 8 would be:

  t'1: <?HotelReservation;-;c=0>.<?invalid date;-;− >

  t'2:<?HotelReservation;-;c=0><?validdate;-;− ><?double;;><!Price;;c=0><?Confirmation;c<5

  <?CC:type/Number;-;− ><!valid/notValid;-;><!Notice;-; c<10>

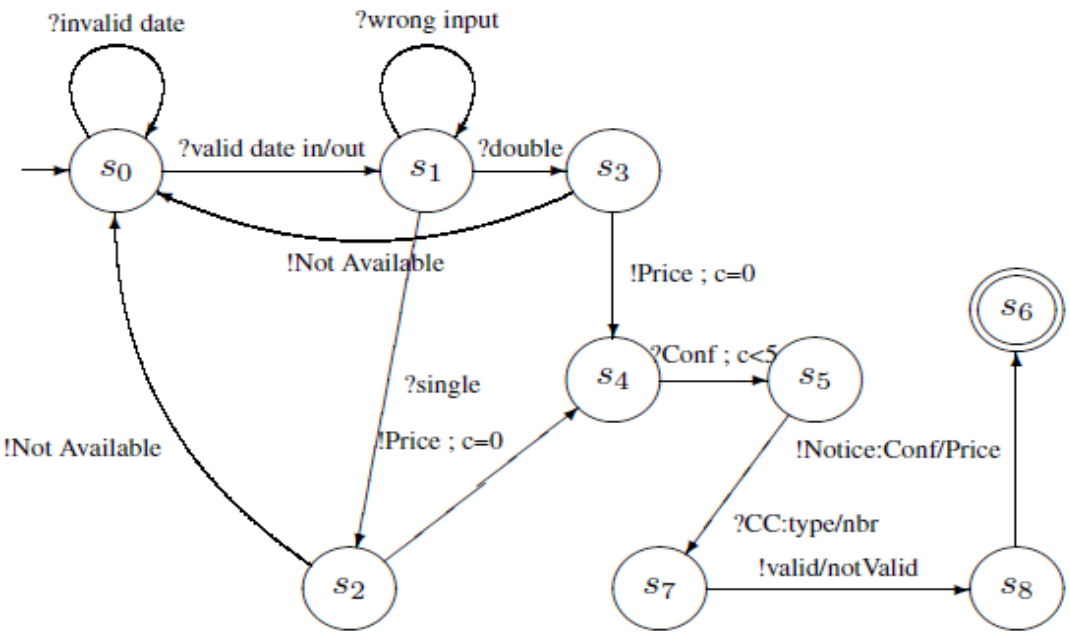

Figure 6.2: TLTS of the composed hotel reservation HR and Credit card validation Ccv components

t'3:<?HotelReservation;-;c=0><?validdate;-;− >.<?single;;>.<!Price;;c=0>.<?Confirmation;c<
;− >.<!valid/notValid;-;>.<!Notice;-; c<10>

t'4: <?HotelReservation;-;c=0><?validdate;-;− >.<?wrong input;-;− >

So we have selected regression test set T' = {t'1, t'2, t'3, t'4 }. Let the input functions, transitions and other functions like "<?CC type/Number;-;− >" be denoted by integer like:

t'1: 1,2

t'2: 1,3,4,6,7, 8,9,10

t'3: 1,3,5,6,7,8,9,10

t'4: 1,3,11.

Now we made a table showing number of functions covered by test cases as follows:

Table 6.2 number of functions covered by test cases

Test (t)	Functions Covered	Cov(t)
t'1	1,2	2
t'2	1,3,4,6,7,8,9,10	8
t'3	1,3,5,6,7,8,9,10	8
t'4	1,3,11	3

Let us apply our PrioTest algorithm on T' to obtain prioritized list of tests. The inputs to PrioTest are: T', TransCov = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 }, and the test coverage vectors for each test in T' as in table above. Step 1: X'= {t'1, t'2, t'3, t'4}. Here t'2 and t'3 both cover most of the number of functions (8) in X' and so both have least cost. Hence we randomly select t'3. Step 2: PT= <t'3 >. X'= {t'1, t'2, t'4 }, TransCov = {2, 4, 11 }. Step 3: We continue the process as X' and TransCov is not empty. Step 3.1: compute the residual coverage for each test in X' as follows:

C (t'1) = | {2, 4, 11} \ ({1, 2} ∩ {2, 4, 11}) | = | {4, 11 } | = 2. C (t'2) = — 2,4,11 (1,3,4,6,7,8,9,10 ∩ 2,4,11) — = —2,11— = 2 C (t'4) = — 2, 4, 11 (1, 3, 11 ∩ 2, 4, 11) — = —2, 4— = 2

Step 3.2: t'1, t'2 and t'4 all have the least cost (2). We arbitrarily select t'1. Step 3.3: PT = < t'3, t'1 >. X' = { t'2, t'4 }, TransCov = {4, 11 }. Since X' and TransCov is not empty we go back to step 3 again. Step 3.1: we again compute the residual coverage for each test in X' as follows:

C (t'2) = | {4,11} \ ({1,3,4,6,7,8,9,10 } ∩ {4,11}) | = | { 11 } | = 1 C (t'4) = | {4, 11} \ ({1, 3, 11} ∩ {4, 11}) | = | { 4 } | = 1

Step 3.2: t'2 and t'4 both have the least cost (1). We arbitrarily select t'2. Step 3.3: PT = < t'3, t'1, t'2 >. X' = {t'4 }, TransCov = { 11 }. Since X' and TransCov is not empty we go back to step 3 again. Step 3.1: we again compute the residual coverage for each test in X' as follows:

C (t'4) = | { 11 } \ ({1, 3, 11} ∩ {11}) | = ϕ

Step 3.2: t'4 has the least cost and as only one test has left and it covers the rest one

functions so we must select this. Step 3.3: PT = <t'3, t'1, t'2, t'4>. X' = ϕ, TransCov = ϕ. Step 4: as there no test is remained, we stop the procedure.

Output: PT = <t'3, t'1, t'2, t'4 >. Here we had all tests in T' are prioritized.

The two examples above showed the prioritization of tests in a test suite for two kind of modification in a web application. This reschedule will help testers to test some of tests with higher priority in a test suite when time and budget are limited.

CHAPTER 7
DISCUSSION AND CONCLUSION

7.1 Achievement of Our Work

In this paper we have developed a cost effective regression optimization technique, which reduces the regression cycle time and improves the quality of testing. We have presented a hybrid regression testing technique for retesting modified web applications. In this regard, we have choose a regression test selection algorithm that selects only necessary test sequences needed to ensure the correctness of the modified system. That algorithm is safe because it selects every test sequence that corresponds to a different behavior in the modified system. In order to make sure of what to test, we analyze the possible changes and their influences in web applications detailed and represent web application using two level abstractions. Finally, we can greatly improve the quality and efficiency of the regression testing developing a prioritization algorithm for selected regression test suit. Our analysis suggests that this technique can improve the rate of fault detection of test suites in least expensive way. Finally, as we use cost effective scheme for both regression test suite selection and prioritization, compared with other techniques, our approach has a wider scope of application.

7.2 Limitations

Due to shortage of time we could not implement and simulate our algorithm practically. But from empirical study we can ensure that our proposed approach will meet the goal successfully. Another limitation of our work is in selection of the base of prioritization technique. As different web applications have different goal or purposes, the cost criterion for prioritization can be different.

7.3 Future Plan

In future research, we plan to perform an experimental study on larger models and systems to have better understanding of the advantages and limitations of our proposed regression test technique. So under the condition of numerous changes of users' demands, we can ensure the functional correctness of web applications and our testing approach.

REFERENCES

[1] H. Fouchal A. Tarhini and N. Mansour. A simple approach for testing web service based applications. *international conference on Innovative Internet Community Systems, I2CS 2005, Paris-France*, 2005.

[2] Hacne Fouchal Abbas Tarhini and Nashat Mansour. Regression testing web services-based applications. *Computer Systems and Applications, IEEE International Conference*, 2006.

[3] D. Binkley. Semantic guided regression cost reduction. *IEEE*, 1997.

[4] E. Engstrm and P. Runeson. A qualitative survey of regression testing practices. *International Conference on Product-Focused Software Process Improvement*, 2006.

[5] M. J. Harrold G. Rothermel and J. Dehia. Regression test selection for c++. *software. J. Softw. Testing, Verif., and Rel. 10(2)*, 2000.

[6] Luay H. Tahat Bogdan Korel Mark Harman and Hasan Ura. Regression test suite prioritization using system models.

[7] Yogesh Singh K.Aggrawal and A.Kaur. Code coverage based technique for prioritizing test cases for regression testing. *ACM SIGSOFT Software Engineering Notes*, 2004.

[8] Arvinder Kaur and Shubhra Goyal. A genetic algorithm for regression test case prioritization using code coverage. *International Journal on Computer Science and Engineering (IJCSE)*, 20011.

[9] Chen Jixiang Jiang Lei Xu Baowen, Xu Zhenqiang. Regression testing for web applications based on slicing. *IEEE*, 2006.

[10] H. Leung and L. White. A firewall concept in both control-flow and data-flow in regression integration testing. *In proceedings of the Conference on Software Maintenance, pages 262-271*, 1992.

[11] P. Hisa X. Li D.C. Kung C.T. Hsu Y. Toyoshima L. Li and C. Chen. A technique for the selective revalidation of oo software. *Journal of Software Maintenance 9,pages 217-233*, 1997.

[12] Nilam Kaushik Mazeiar Salehie Ladan Tahvildari Sen Liz and Mark Moorez. Dynamic prioritization in regression testing.

[13] N. Mansour and R. Bahsoon. Reduction-based methods and metrics for selective regression testing. *In Information and Software Technology*, 2002.

[14] N. Mansour and K. El-Fakih. Simulated annealing and genetic algorithms for optimal regression testing. *International Journal of Computer Applications*, 1999.

[15] Aditya P. Mathura. Foundations of software testing.

[16] G. Rothermel and M. J. Harrold. A safe, efficient algorithm for regression test selection. *the conference on software maintenance*, 1993.

[17] F.I. Vokolos and P.G. Frankl. Pythia. a regression test selection tool based on textual differencing. *3rd International Conference on Reliability,Quality and Safety of Software-Intensive Systems*, 1996.

[18] Kristen R. Walcott and Mary Lou Soffa. Time aware test suite prioritization. *ACM SIGSOFT/SIGPLAN International Symposium on Software Testing and Analysis*, 2006.