California State University, San Bernardino
CSUSB ScholarWorks

Theses Digitization Project | John M. Pfau Library

2006

# Testing for the Solicitation Management System

Lu-Yi Wu

Follow this and additional works at: https://scholarworks.lib.csusb.edu/etd-project

 Part of the Software Engineering Commons

Recommended Citation
Wu, Lu-Yi, "Testing for the Solicitation Management System" (2006). *Theses Digitization Project*. 3486.
https://scholarworks.lib.csusb.edu/etd-project/3486

This Thesis is brought to you for free and open access by the John M. Pfau Library at CSUSB ScholarWorks. It has been accepted for inclusion in Theses Digitization Project by an authorized administrator of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.

# TESTING FOR THE SOLICITATION MANAGEMENT SYSTEM

A Project

Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

by

Lu-Yi Wu

June 2006

# TESTING FOR THE SOLICITATION MANAGEMENT SYSTEM

A Project

Presented to the

Faculty of

California State University,

San Bernardino

by

Lu-Yi Wu

June 2006

Approved by:

[redacted]

Dr.David Turner, Chair, Computer Science

6/1/2006

Date

[redacted]

Dr. George Georgiou

[redacted]

Dr. Ernesto Gomez

ABSTRACT

This project is to test the Solicitation Management System (SMS). The SMS is an online system that facilitates processing of a solicitation at the Office of Technology Transfer and Commercialization (OTTC). It allows potential applicants to submit applications to OTTC for further processing.

Testing done in this project can mainly be divided into two distinct parts. They are manual testing and automated testing. Each testing method has its advantages and disadvantages. Through a combination of both testing methods, it is hoped that faults in the system can be discovered.

This report includes a limited review of software testing literature.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Turner. He has been a wonderful advisor who offers support of knowledge and resources throughout this project. I would also like to thank my committee members Dr. Georgiou and Dr. Gomez for their valuable feedbacks to make this project better. Last, but not least, I would also like to thank Dr. Concepcion and Dr. Turner together to let me have a chance to join the SMS project, which helps me to deeper understand the project that I am testing on.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

# CHAPTER ONE

# INTRODUCTION

## 1.1 Purpose of This Project

The purpose of this project is to write a set of test cases that can detect undesired behaviors for the Solicitation Management System (SMS).

The SMS is a web-based application that facilitates processing of solicitations for the Office of Technology Transfer and Commercialization (OTTC). Potential applicants can submit their applications to OTTC using the SMS and officers at OTTC can process and assign evaluators to applications.

The project uses two major types of tests. One is manual testing, and the other is automated testing. Details of testing strategies will be described later.

With the combination of both manual testing and automated testing, it is the goal of this project to discover faults for the SMS system if it exists.

## 1.2 Scope of Project

### 1.2.1 Deliverables

This project contains the following deliverables:

1. Functional test code that validates basic functionality of SMS.

2. Security test code.

3. Concurrency test code.

4. Load test code.

### 1.2.2 Function of Testing

This project consists of a group of testing strategies that was written to capture faults of the SMS if it exists.

## 1.3 Significance of the Project

"If you didn't test it, it doesn't work" [1] might be the best description of the significance of testing. Many software developers concentrate on writing the program itself and neglect the importance of testing.

It is better to test a product and capture bugs before commercial release than to have to spend more time and money to have it fixed after it is delivered to the client. Fixing a product after delivery not only costs relatively more than fixing it during development, it would also affect customer's confidence in our product and capabilities of quality control.

## 1.4 Definition and Abbreviations

SMS - Solicitation Management System.

OTTC - Office of Technology Transfer and Commercialization.

JUnit - JUnit is a framework that can be used to perform testing. It provides a series of methods that can be useful when writing test cases.

HttpUnit - HttpUnit is a framework that can be used to test web applications. It emulates a web browser and can perform related behaviors and can be used to bypass the browser to test the web application. It can be used in conjunction with JUnit.

JUnitPerf - JUnitPerf is an open source that can be used with JUnit to perform timed and load testing.

## 1.5 Organization of the Documentation

The remaining sections of this document is organized as follows: Chapter 2 is a literature review of software testing. Chapter 3 introduces the Solicitation Management System. Chapter 4 illustrates the testing strategies. Chapter 5 presents the project implementation for manual testing. Chapter 6 presents the project implementation for automated testing. Chapter 7 provides conclusions and future directions.

# CHAPTER TWO

## LITERATURE REVIEW FOR SOFTWARE TESTING

Testing is an integral part of software development. Testing provides one means for stakeholders to verify the quality of a component within a system as the system is being developed, or to verify the overall quality of a software system prior to its deployment. The purpose of this chapter is to review different testing theories and techniques that are currently available. The theories reviewed can be categorized into three different types: test the application to full extent; test the application for all possible usage in the future; and test the application with selected test cases. The techniques reviewed cover a broad variety of software testing. They include techniques for general (vanilla) software testing, version-specific software testing, multi-version software testing, system level software testing, unit level software testing, and function level software testing. Details of individual techniques will be introduced later in this chapter. Of the studies reviewed, most of them claimed that the technique they introduced is effective. However, one study reports that some of the techniques introduced in its paper are effective while others are not.

## 2.1 Introduction

Testing is an important part of the software development cycle. Through testing, we can verify whether the software in question delivers the functionalities against specification and validate whether the software has rendered its expected behavior. Bob Colwell once wrote in Computer Magazine, "If you didn't test it, it doesn't work" [1], might best describe how essential testing is for software validation.

It is intuitive to understand testing is important. However, the process of testing can use up a lot of resources. If we take into consideration that software testing consumes at least 50% of software development cost and reusing test suites consumes almost 50% of software maintenance cost [4], we would come to realize that the problem involving testing has come down to simply how to test economically. As a result, in order to seek out solutions the above question, several studies had been conducted. The purpose of this paper is to review current theories and techniques available for software testing.

Section 2.2 will present the theories and techniques used in the studies reviewed. Section 2.3 will be a comparison between the studies.

## 2.2 Testing Theories and Techniques

Theories in testing can mainly be categorized into three different types. The first testing theory is to test the application to the full extent. The advantage of this method is that it might uncover underlying faults of the application since most things that are designed cannot be tested to saturation [1].

The second testing theory is to test the application for all possible usage in the future. However, due to the mass possibilities and combinations, it might be time consuming to conduct the test and it might also drive the tester crazy [1].

The third testing theory is to be selective and choose a number of test cases to test the application. This is more applicable when a large system is being tested. However, since only a portion is chosen to be tested, we run a risk that an error might go undetected.

### 2.2.1 Testing to the Full Extent

Testing to the full extent has its advantages and disadvantages. The advantage of testing to the full extent is that it is more likely to uncover faults within the application. However, the disadvantage of that is that it can be very time consuming and costly.

To solve this problem, a method of testing to full extent while preserving test efficiency was brought up by Gregg Rothermel et al. This method is called prioritization. In the studies reviewed, three [4] [5] [6] studies mentioned use prioritization as a mean to increase fault detection in early stages of testing.

When a test cannot fully run to the end, the rate of fault detection prior to the stop is crucial. The faults detected can give faster feedbacks and allow developers to fix the problem early on. This is of great value because in real world not all test cases can run to the end. Some are stopped due to crashes and some are interrupted or even canceled due to scheduling issues.

Different studies have different techniques for prioritization. In a study conducted by Hema Srikanth et al. [4], it proposes a system level prioritization technique. The idea was to assign a value between 1 and 10 to the four factors they identified: the customer-assigned priority (CP), the requirements complexity (RC), the requirements volatility (RV), and the fault proneness (FP). Each factor can be assigned a weight (total weight to be 1.0) to emphasize the importance of that feature for an individual program.

A Prioritization Factor Value (PFV) is then calculated by summing the product of the value and weight. PFV is used to calculate the Weighted Priority (WP). WP decides the priority of test cases. Test cases with higher values run before ones with lower values.

In another study conducted by Gregg Rothermel et al. [5], it proposes eight techniques for general prioritization. Prioritization techniques can mainly be categorized into two parts: total and additional. Techniques that do not require feedbacks are named with "total" and techniques that require feedbacks are named with "additional".

The first technique introduced is random prioritization. In random prioritization, the tests are run randomly. The second technique is optimal prioritization. In optimal prioritization, tests are run based on the number of faults each test case reveals. Tests that reveal more number of faults are run first.

The third technique is total statement coverage prioritization. Total statement coverage prioritization bases the ordering of tests on the number of statements that are covered by each test case. Tests that reveal more number of faults are run first.

The fourth technique is additional statement coverage prioritization. This technique first chooses a test case that covers the greatest number of statements. Then it selects from the remaining test cases that covers the most statements that has not been covered yet.

The fifth technique is total branch coverage prioritization. It chooses test cases based on the number of branches that are covered by each test case. Tests that cover more branches are run first.

The sixth technique is additional branch coverage prioritization. It first chooses a test case that covers the greatest number of branches. Then it selects from the remaining test cases that covers the most branches that has not been covered.

The seventh technique is total fault-exposing-potential (FEP). In this technique, summations of all FEP for all statements are assigned to an award value. Test with higher award values are run first.

The last technique is additional fault-exposing-potential (FEP) prioritization. It uses a term called confidence. Confidence is a value similar to the FEP used in total fault-exposing-potential prioritization. This technique first chooses a test case that has the greatest

confidence. The confidence value is then updated and the confidence values for the remaining test cases are recalculated.

In a third study conducted by Sebastian Elbaum et al. [6], it based its study on [5] and added several new techniques. It proposes eighteen techniques for version specific prioritization. The techniques it proposes can mainly be categorized into four parts.

The first part concerns granularity. It divides the techniques into function level and statement level. The second part concerns feedbacks. Techniques that do not require feedbacks are named with "total" and techniques that require feedbacks are named with "additional".

The third part concerns information from modified version. Techniques that do not require information from modified version are named with "FEP". Techniques that do require information from modified version are named "FI" (fault index). The fourth part concerns practicality. Techniques in this study are categorized by whether they are practical or not. Techniques that are based on coverage and FI are practical while techniques that are based on FEP are exploratory.

The first six techniques introduced in this study were covered in the previous study. They are random ordering, optimal ordering, total statement coverage prioritization, additional statement coverage prioritization, total FEP prioritization, and additional FEP prioritization. Of the techniques mentioned above, the last four techniques are statement level techniques.

The seventh technique is total function coverage prioritization. This technique is similar to that of total statement coverage prioritization except that it deals with functions instead of statements.

The eighth technique is additional function coverage prioritization. This technique is similar to that of additional statement coverage prioritization except that it deals with functions instead of statements.

The ninth technique is total FEP (function level) prioritization. This technique is similar to that of total FEP prioritization except that it processes at a function level.

The tenth technique is additional FEP (function level) prioritization. This technique is similar to that of additional FEP prioritization except that it processes at a function level.

The eleventh technique is total fault index (FI) prioritization. FI is used to estimate fault proneness. This technique is similar to total function coverage prioritization. Summations of all FI for all functions are calculated. It chooses test cases based on the value calculated. Tests with a higher value are run first.

The twelfth technique is additional fault-index (FI) prioritization. This technique is similar to additional function coverage prioritization except that it processes with FI.

The thirteenth technique is total FI with FEP coverage prioritization. This technique sums the product of FI and FEP for all functions that a test case executes. Then the test cases are chosen based on the value calculated. Tests with higher value are run first.

The fourteenth technique is additional FI with FEP coverage prioritization. This technique is similar to total FI with FEP coverage prioritization except that it involves feedback.

The fifteenth technique is total DIFF prioritization. In this technique, syntactic differences between two versions of a program are being calculated. This technique is similar to total DIFF prioritization except that it processes with diff.

The sixteenth technique is additional DIFF prioritization. This technique is similar to additional FI prioritization except that it processes with diff.

The seventeenth technique is total DIFF with FEP prioritization. This technique is similar to total FI with FEP prioritization except that it processes with diff.

The eighteenth technique is additional DIFF with FEP prioritization. This technique is similar to additional FI with FEP prioritization except that it processes with diff.

### 2.2.2 Testing for All Possible Future Usage

As mentioned before, testing for all possible future usage is both time consuming and quite irrelevant. There can be mass numbers of possibilities and combinations that may result in a new future usage. Spending a lot of time and energy to tackle this kind of problem is probably not wise.

### 2.2.3 Testing with Selection

Testing with a selection of test cases has its advantages and disadvantages. The advantage of testing with selection is the time and cost it saves to run the tests. The disadvantage, however, is that if the test selection was not chosen carefully, it might not detect all faults that are present.

Several studies and articles [3] [7] [8] [9] backs up the theory that testing should be done with a selection of test cases instead testing to the full extent despite that their techniques of test selection differs from one and another.

In an article written by Tim Menzies et al. [3] in the IEEE Software Magazine, the authors mentioned a technique called formal method. In formal methods, essential details and logical constraints are specified and never be violate. Thus, test cases are written to check against violations of the rule.

In a study conducted by Yanping Chen et al. [7], it focuses on specification-based test selection. In this method, two kinds of regression tests are selected. One is the targeted test that checks the new release for the presence of current important customer feature. The other is the safety test that checks for potential problem areas.

In a second study conducted by Mary Jean Harrold et al. [8], it uses coverage-based predictors to perform test selection. There are two predictors used. They are the DejaVu, implemented by Rothermel and Harrold and the TestTube implemented by Rosenblum and Weyuker. This study has a hypothsis: "Given a system under test P, a

regression test suite T for P, and a selective regression testing method M, it is possible to use information about the coverage relation coversM induced by M over T and the entities of P to predict whether or not M will be cost-effective for regression testing future versions of P."

In a third study conducted by Todd L. Graves et al. [9], four test selection techniques were introduced. The first technique is the minimization technique. In this technique, test cases that cover the modified part of the program are selected. However, the test cases selection is kept to a minimum.

The second technique is the dataflow technique. In this technique, test cases that have data interaction with the modified part of the program are selected.

The third technique is the safe technique. In this technique, test cases that are selected include all test cases in the original version that can detect faults in the modified version.

The fourth technique is the ad hoc/ random technique. This technique has been introduced early in section 2.1. The ad hoc portion of this technique is usually based on experience of hunches that the developer gets.

## 2.3 Study Comparisons

The studies reviewed in this paper mostly aim at the goal of introducing a more efficient way for testing. Most of the studies are aimed toward this goal in one way or another. Studies [4][5][6] mainly focuses on prioritization while [7][8][9] introduces different methods of test selection. The researches or studies are mainly done with the goal of raising the fault exposing rate in early stages of testing. Regardless of what technique it employs, the final objective is to efficiently and effectively expose as much fault as possible within the initial stages.

Of the six studies reviewed, two[8][9] of them had a hypothesis. [8] hypothesized that current information can be used to predict cost-effectiveness for future version regression testing. [9] hypothesized that trade-offs between the cost of test selection and execution with fault detection sufficiency differs with different test selection techniques. Even though the two hypotheses look irrelevant at a glance, they provide a theory base for the techniques that are presented in the individual studies.

All six studies are done on software testing. They cover software testing from different aspects and perspectives. [4] covers testing on a system level; [5]

covers testing as a general rule; [6] covers testing that are version specific; and [8] covers testing over multiple versions. Since there are different coverage of software testing, it is essential to discuss all possible types of testing possible for different aspects (range/ coverage). Thus discussion of testing that provides different coverage suffices this purpose.

All six studies use techniques and methods introduced in their study to conduct their experiment or research. Techniques introduced are different from study to study. However, since [6] is a follow up research of [5], it uses six of the techniques introduced in [5]. A wide variety of techniques in this case is an advantage because sometimes one technique might suffice one aspect of testing while it might prove insufficient for another. Thus, in order to cover all aspects of testing, different techniques are necessary.

Four studies [4] [5] [6] [7] [8] claim effectiveness in the techniques they introduced. One study [9] reports that some of the techniques introduced in its paper are effective, some are not. It is important for a paper to stand by the idea it proposes. However, some studies only conduct tests or report results that are favorable to them.

A study that really tests all possibilities and report the outcome regardless of how it looks might be more convincing and thus less bias.

Since software testing can have many aspects, studies that test different facets may come to different conclusions. [4] states that customer satisfaction can be increased when severe faults are corrected early. [5] state that of the techniques they proposed, the FEP-based are not as practical as the code-coverage-based techniques due to cost. [6] states that adding fault proneness measurements into prioritization is not as beneficial as expected.

[8] states that predictive model test selection accuracy can be affected significantly by the distribution of modifications made to a program. Code coverage and modification distribution must be both accounted for to achieve a more precise accuracy. [9] states that the cost-effectiveness of regression testing is affected by the choice of selection algorithm.

# CHAPTER THREE

# INTRODUCTION TO THE SOLICITATION MANAGEMENT SYSTEM

This chapter is a brief introduction to the Solicitation Management System.

## 3.1 Introduction

The Solicitation Management System is an online application written for the Office of Technology Transfer and Commercialization (OTTC). It is a web application that can be used to facilitate processing of a solicitation.

OTTC is an office that assists in transitioning promising new technologies from government and academic laboratories alike into full commercialization. When a grant proposal is selected, an amount of founding will be rewarded to the applicant.

A system with the purpose of supporting the goal mentioned above via a grant proposal solicitation management system was implemented by the Department of Computer Science lead by Dr. Turner. The test cases in project are aimed at testing the latest (third) release of this system. Figure 1 shows the login page for this release.

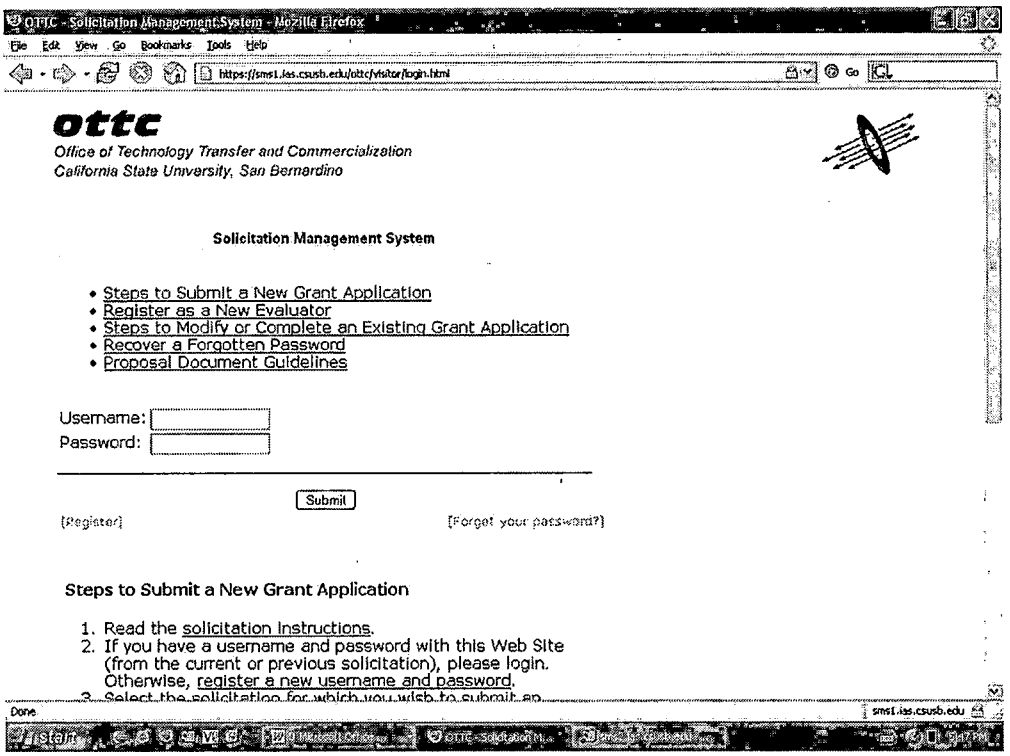


Figure 1. Solicitation Management System Login Page

## 3.2 User Roles

There are five user roles for the SMS. They are the administrator, applicant, evaluator, officer, and staff. Their main roles are described as follows.

The administrator, officer, and staff roles are mainly personnel from OTTC. The administrator manages the officer and staff member's user accounts. The officer runs the solicitation and can make changes to solicitation related activities if necessary. The staff member can view

solicitation related activities but cannot make any changes.

The applicant and evaluator roles are usually people from outside of OTTC. An applicant is anyone who registers himself into the SMS as an applicant. He then can view open solicitations and submit an application if he wishes to. An evaluator is usually a person assigned or invited by OTTC. He also registers himself as an evaluator and can login to view his assigned jobs.

## 3.3 Functions

The SMS has several functions that aid the processing of a solicitation. They are described as follows by the user roles.

The admin role can manage officers accounts (which includes create, edit and delete) and manage admin's own profile. Figure 2 is a use case diagram for the admin's role.

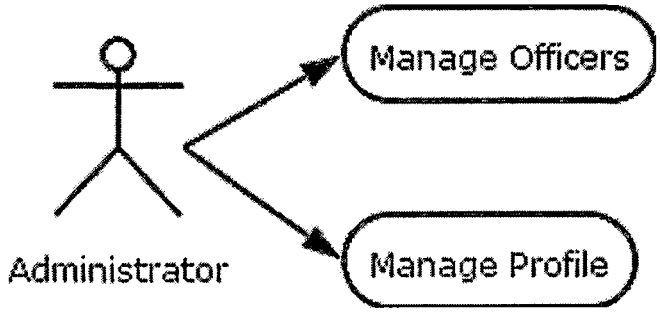


Figure 2. Use Case Diagram - Admin's Role

The applicant role can view details of open solicitations, manage (create, edit, and delete) his own applications to open solicitations, and manage his own profile. Figure 3 is a use case diagram for the applicant's role.

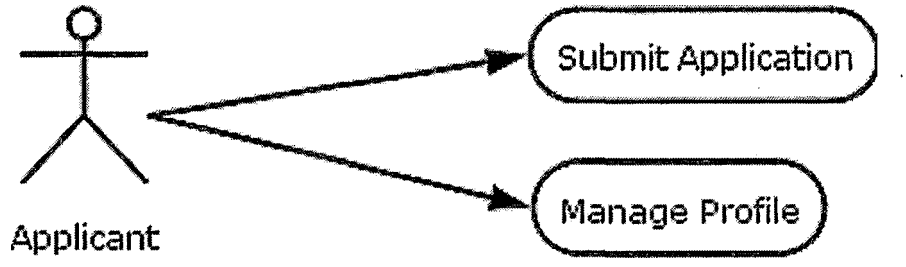


Figure 3. Use Case Diagram - Applicant's Role

The evaluator role can view his assigned proposals, write an evaluation, and manage his own profile. Figure 4 is a use case diagram for the evaluator role.

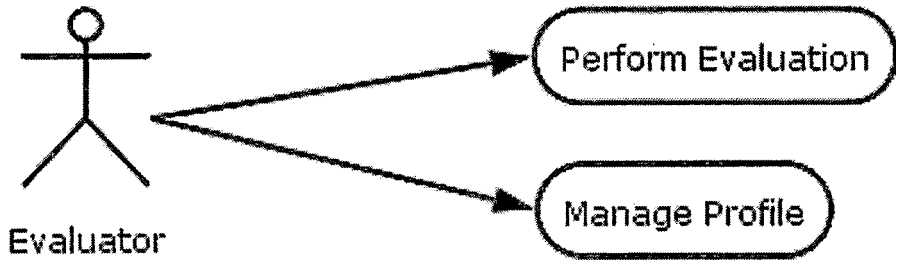


Figure 4. Use Case Diagram - Evaluator's Role

The officer role can manage (create, edit, delete, and assign evaluators) solicitations, manage his own profile, manage (create, edit, and delete) application groups, manage (edit and delete) evaluations, manage (edit and delete) applications, manage (delete evaluators, write memos regarding that evaluator and edit evaluator's profile) evaluators, manage applicants (delete applicant and edit applicant's profile), and generate real time reports (the applicant dump and evaluator dumps are global reports and the evaluation reports and application reports are solicitation specific reports). Figure 5 is a use case diagram for the officer role.

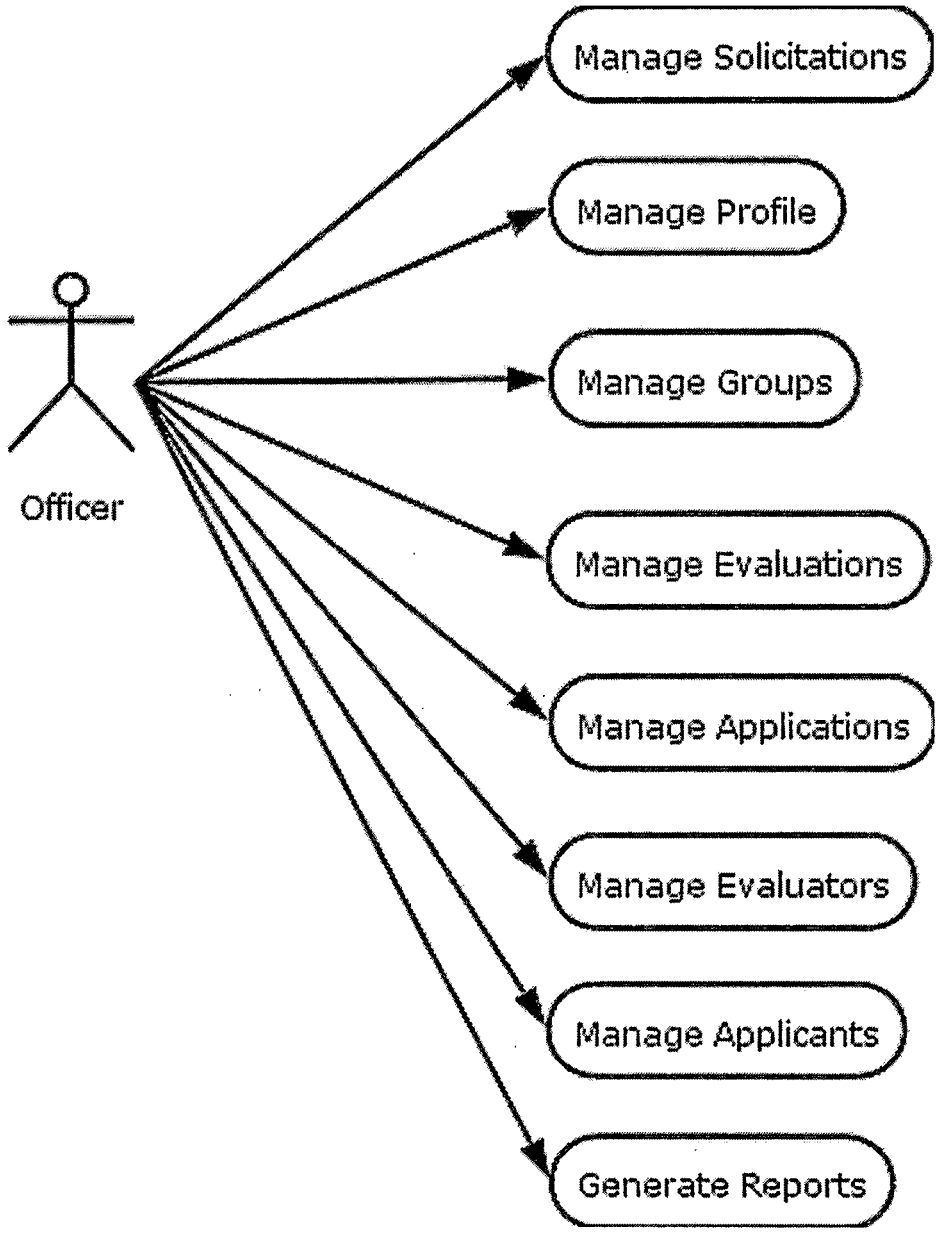


Figure 5. Use Case Diagram - Officer's Role

The staff member role can view solicitations, manage his own profile, view application groups, view evaluations, view applications, view evaluators, and view applicants. Figure 6 is a use case diagram for the staff member's role.

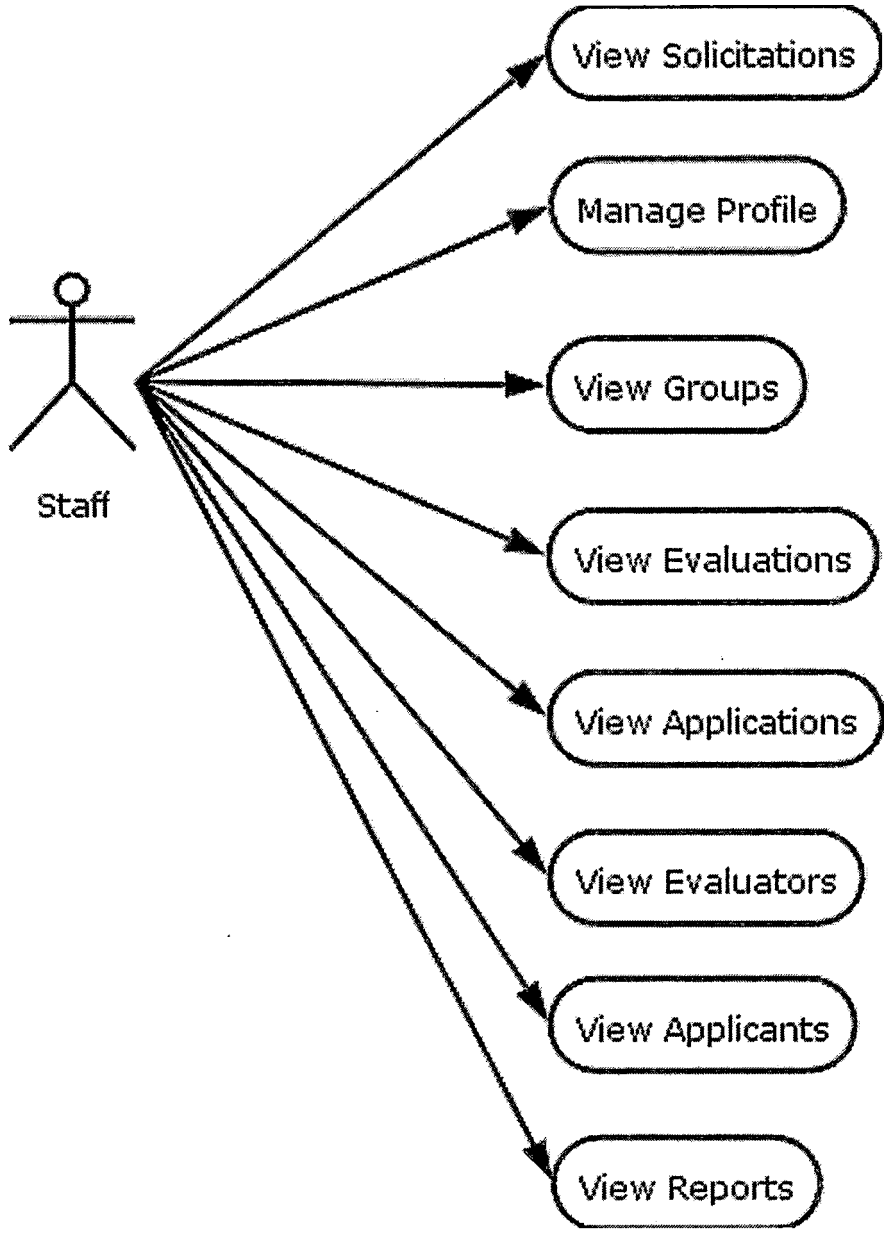


Figure 6. Use Case Diagram - Staff Member's Role

# CHAPTER FOUR

# TESTING STRATEGIES

## 4.1 Introduction

Testing is a way of ensuring the quality of a product. With fair test cases implemented along the actual coding of a system, erroneous scenarios can be dealt with from early phases of development.

This is a valuable asset because if the problem shows up after a system is in production; it might take more effort to do massive debugging and changing the system as a whole than what could have been done if the error was corrected earlier.

Further more, if the bugs (or malfunctions) of a system occur after a system is in service, it is more likely that it will result in high maintenance and let alone the fact that our customer might lose faith in us due to a faulty product.

## 4.2 Testing Frameworks

### 4.2.1 JUnit

JUnit is a framework that can be used to conduct testing. This framework comes with a junit.jar (which at this time is junit-3.8.1.jar) and is comprised of fixtures,

test cases, suites, and testrunners. Tests can be carried out by writing simple test cases or by writing a test suite.

A simple test case can be written in four consecutive steps. In the first step, an instance of TestCase is created. After creating an instance of TestCase, a constructor should be created which accepts a String as a parameter and passes it to the super class. Next overwrite the runTest() method. And finally, use one of the assert functions, for example the assertTrue(), to validate values. A Boolean true is passed for assertTrue() if the test succeeds and a Boolean false is passed if the test fails.

When the numbers of test start to grow, a fixture may be used when operating on similar objects. Using a test fixture can avoid duplicating the initialization (the setUp() method) and cleanup (the tearDown() method) of the common objects for each test. Tests can use objects in a test fixture. Each test runs and invokes different methods on objects within its own fixture.

When running more than one test at a time is necessary, test suites can be used. First, a new TestSuite is declared. After the declaration of a new TestSuite, addTest method is used to add tests to the suite. The suite can then be accessed and executed by a TestRunner.

There are two ways of using addTest. One way is to declare a new instance of the test case under consideration. E.g.,

```
TestSuite suite = new TestSuite();
Suite.addTest(new EditAreas());
```

Another is to pass the class of the TestCase under consideration to the TestSuite constructor. E.g.,

```
TestSuite suite = new TestSuite(EditAres.class);
```

### 4.2.2 HttpUnit

HttpUnit can be used to test web applications. It emulate the properties of a browser, thus it can be used to bypass the browser to access a website for testing purposes.

HttpUnit can emulate for submission, JavaScript, basic http authentication, cookies, and automatic page redirection. It also allows Java test code to examine returned pages either as text, and XML DOM, or containers for form, tables, and links.

HttpUnit can be used in conjunction with JUnit. With a combination of both frameworks, testing for a web-based application is made possible.

### 4.2.3 JUnitPerf

JUnitPerf can be used to conduct performance tests. It is an open source that can be used with the JUnit framework.

Performance measurements are done on existing JUnit tests. This leads to two advantages. The first advantage is the reusability of the existing JUnit code. It is because of the reusability, productivity for performance testing is higher. The second advantage is the reduction of the learning curve. Since JUnitPerf is used with JUnit, the coding style of JUnitPerf is very similar to that of JUnit.

JUnitPerf provides two kinds of performance tests: the timed test and the load test. The timed test provides two functionalities. The first functionality is the measurement of the time used to run a test. The second functionality is to validate whether the test is run within the given time limit.

The load test runs the given test with a specified number of users and iterations. It can be carried out with concurrent users or users with a specific time delay.

## 4.3 Testing Strategies

Testing strategies for this project can mainly be divided into manual testing and automated testing.

Manual testing is further divided into three sub-categories. The first category is the manual testing that developers do from moment to moment as the code is written. The second category is the manual testing performed by the development team within a progress review meeting. The third category is the manual testing performed by the client during prototype review sessions.

Automated testing is also divided into sub-categories. There are six areas that are defined for automated testing. The first category is the functional tests that are written prior to design to capture system requirements. The second category is the general functional tests written after implementing functionality to verify correctness. The third category is the security tests that are written to document and verifies security mechanisms, including authentication and authorization constraints defined for user roles.

The fourth category is the load tests to measure the capacity of the system. The fifth category is the concurrency tests to verify that the code is free from hard-to-find bugs that occur rarely in multi-threaded code.

The sixth category is the database population tests that are used to test system functionality as well as to populate the database with realistic data for manual testing and demonstration of the system to the client.

# CHAPTER FIVE

# MANUAL TESTING

Manual testing has the advantage of revealing flaws that were not anticipated by the test code writer. This is because project developers tend to test the application within the scope of intended use while users often do not limit themselves to this boundary. This leads the test to other possible uses of the system[2].

## 5.1 Modification Testing

This type of testing is usually done by the developer after a new functionality is written or when a requirement has changed and the code was modified to accommodate the change. The developer usually tries the new functionality on the website and verifies if the application has rendered the correct view or behaved appropriately.

## 5.2 Progress Review Meeting Testing

This type of testing is done by the whole development team at a progress review meeting. Usually a demonstration of newly implemented functionalities is done to the whole development team. At times, a pre-run of an intended demonstration to the client is also done. During progress

review meeting testings, functionalities of the application are performed and the actual behavior of the application is verified against the desired behavior.

### 5.3 Client Review Prototype Session Testing

This type of testing is done by the client. During a client review prototype session, the client tries to use the application and identifies unexpected behavior. This is more of specification verification than a technical specification even though at times the client might find a faulty function.

# CHAPTER SIX

## AUTOMATED TESTING

The major decision of this project is what to test. The pseudo code written by J.B. Rainsberger shown below can depict the complication of deciding what to test.

```
becomeTimidAndTestEverything
while writingTheSameThingOverAndOverAgain
   becomeMoreAggressive
   writeFewerTests
   writeTestsForMoreInterestingCases
   if getBurnedByStupidDefect
        feelStupid
        becomeTimidAndTestEverything
   end
end
```

There are a few method of testing. The first method is to test the application to the full extent. The advantage of this method is that it might uncover underlying faults of the application since most things that are designed cannot be tested to saturation[1].

One method of testing to saturation is prioritization. Several papers [4] [5] [6] agree on this method. In the test prioritization method, test cases are ordered to maximize the effectiveness for a performance goal for fault detection.

The second method of testing is to test the application for all possible usage in the future. However, due to the mass possibilities and combinations, it might take a long time to test and it might also drive the tester crazy[1].

The third method is to be selective and choose a number of test cases to test the application. This is more applicable when a large system is being tested.

Since both testing to saturation and anticipating possible usage of the application is not quite applicable, there should be a compromise. Just how exactly to draw that line itself is a question.

To solve this problem, several papers were researched. Each paper had their own theory and their conclusions are not always coherent. So, after reading the papers, it is necessary to process the information to understand the drawbacks and advantages of each theory and choose the one that works best for this project.

The first method of test selection is called formal method. One paper points out that "on the average, elaborate and expensive testing regimes will not yield much more information than inexpensive manual or simple automatic testing schemes" [3]. It claims that in formal methods, essential details and logical constraints should be specified and never be violated. Thus, test cases can be written to check against violations.

The second method of test selection is specification-based method. In this method, two kinds of regression tests are selected. One is the Targeted Test that checks the new release for the presence of current important customer feature. The other is the Safety Test that checks for potential problem areas [7].

The third method of test selection is to use coverage-based predictors. The predictors are designed to "predict the effectiveness of regression test selection strategies" [8]. In the paper that mentioned this method, the authors concluded that both modification distribution and code coverage must be considered to improve accuracy.

For details regarding different testing theories and methods, please refer to the paper in appendix A.

From the testing theories provided in the papers, the conclusion has drawn to test the SMS with essential data and logical constraints. Using this as a guideline, the implementation of automated testing following the testing strategies mentioned in section 3.3 is shown below.

## 6.1 Test Driven Design (TDD)

In Test Driven Design, functional tests are written prior to design to capture system requirements. This can be implemented with new functionalities (or components) that are added later to the Solicitation Management System.

## 6.2 General Functional Tests

The general functional tests examines whether the web application is behaving as expected. There is much functionality in the SMS system. To test all functions is tedious and inefficient. Thus only the essential functions that will affect the operation or behavior of the SMS are tested using automated testing. Other functionalities, such as the correctness of links and etc., will be tested randomly or through manual testing. The tests are categorized by their user role. Table 1 lists the tests that were done for function testing.

Table 1. Functional Tests

| Role | Test |
|---|---|
| Admin | Create Officer Test. Test creating an officer and uses the newly created account to log in. |
| | Create Staff Test. Test creating a staff member and uses the newly created account to log in. |
| | Change Other's Password Test. Test resetting an officer or staff member's password and tries to log in using the newly changed password. |
| | Delete Account Test. Test deleting an officer or a staff member's account and verifies that the account cease to exist. |
| | Change Own Password Test. Test changing admin's own password and tries to log in using the newly changed password. |
| Applicant | Application Without Proposal Test. Test the application process without uploading a proposal to see if the correct application number is generated. |
| | Application With Proposal Test. Test the application process with an uploaded proposal to see if the correct application number is generated. |
| | Change Own Password Test. Test changing applicant's own password and tries to log in using the newly changed password. |
| | Deleting Own Application Test. Test deleting the applicant's own application. |
| | Edit Tech Area Test. Test editing the tech area of the application. |
| Evaluator | Evaluation Status Test. Test writing and evaluation. The evaluation status is |

| | |
|---|---|
| | checked to see if the correct corresponding status is shown correctly. |
| | Change Own Password Test. Test changing evaluator's own password and tries to log in using the newly changed password. |
| | Edit Area Test. Test editing evaluator's tech area and bus area. |
| Officer | Submission Deadline View Test. Test changing the submission deadline and check to see if the applicant role has the correct corresponding view. |
| | Solicitation Status Test. Test changing the solicitation status and check to see if the applicant role and the evaluator role have the correct corresponding view. |
| | Evaluation Deadline Test. Test changing the evaluation deadline and check to see if the evaluator role has the correct corresponding view. |
| | Editing Awards Test. Test editing assigned awards and verify that the selected awards appear when the applicant applies for the solicitation. |
| | Reassign Application Group Test. Test editing selected application groups and verifies that only the selected groups appear in the officer managed field for an application. |
| | Delete Solicitation Test. Test deleting a solicitation and check for corresponding reactions (i.e. a warning message) |
| | Delete Application Test. Test deleting an application and check for corresponding reactions (i.e. a warning message) |
| | Application Status Change Test. Test changing the application status from complete to downselect 1 and check if the |

| | |
|---|---|
| | edit-assigned evaluator function will appear. |
| | Reassign Evaluator Test. Test the edit-assigned evaluator and log in as a newly assigned evaluator to check for jobs. |
| | Create Solicitation Test. Test creating a new solicitation. Check to see if the applicant role can see the newly created solicitation. |
| | Change Own Password Test. Test changing officer's own password and tries to log in using the newly changed password. |
| | Delete Award Test. Test deleting an award and check for corresponding reactions (i.e. a warning message). |
| | Edit Evaluator Memo Test. Test changing the evaluator memo and check the edit-assigned evaluator page to see if the correct corresponding behavior is shown. |
| | Delete Evaluator Test. Test deleting an evaluator both with and without and evaluation. |
| | Delete Applicant Test. Test deleting an applicant both with and without application. |
| | Change Applicant's Password Test. Test changing an applicant's password and tries to log in using the newly changed password. |
| | Change Evaluator Password Test. Test changing an evaluator's password and tries to log in using the newly changed password. |
| | Deadline Validation Test. Test to see if the submission deadline can be set after the evaluation deadline. |
| Staff | Change Own Password Test. Test changing a staff member's own password and tries to |

| | |
|---|---|
| | log in using the newly changed password. |

## 6.3 Security Tests

Security tests are written to document and verify security mechanisms, including authentication and authorization constraints defined for user roles. Table 2 lists tests that were done for security testing.

Table 2. Security Tests

| | |
|---|---|
| Admin | Test that the admin role cannot access homepages of other user roles by directly typing in the url. |
| Applicant | Test that the applicant role cannot access homepages of other user roles by directly typing in the url. |
| | Test that an applicant cannot view another applicant's application. |
| | Test that an applicant cannot view another applicant's proposal. |
| | Test that an applicant cannot delete another applicant's application. |
| | Test that an applicant cannot edit another applicant's application background. |
| | Test that an applicant cannot edit another applicant's application answers. |
| | Test that an applicant cannot edit another applicant's application awards. |
| | Test that an applicant cannot edit another applicant's application technology areas. |
| Evaluator | Test that the evaluator role cannot access homepages of other user roles by directly typing in the url. |
| | Test that an evaluator cannot view another evaluator's evaluation. |
| | Test that an evaluator cannot view proposals that are not assigned to him. |
| | Test that an evaluator cannot write or edit evaluations for applications that are not assigned to him. |
| Officer | Test that the officer role cannot access homepages of other user roles by directly |

| | |
|---|---|
| | typing in the url. |
| Staff | Test that the staff role cannot access homepages of other user roles by directly typing in the url. |

## 6.4 Load Tests

Load tests measures the capacity of the system. Load test for the Solicitation Management System uses the JUnitPerf. The application is tested under stress to see if the can still deliver its functions.

## 6.5 Concurrency Tests

Concurrency tests verify that the code is free from hard-to-find bugs that occur rarely in a multi-threaded code.

## 6.6 Database Population Tests

Database population tests are used to test system functionality as well as to populate the database with realistic data for manual testing and demonstration of the system to the client. This part of testing was originally done by Robert Chen. Modifications to the database population tests have been done after the application functionalities had changed. Table 3 lists tests that were done to populate the database.

Table 3. Database Population Tests

| | |
|---|---|
| Admin | CreateOfficersAndStaff. This test case creates officers and staff member accounts. |
| Applicant | CreateApplications. This test case creates applications for applicants. |
| Evaluator | CreateEvaluations. This test case creates evaluation for evaluators. |
| Officer | AssignAbbreviatedTitle. This test case assigns abbreviated title to applications and at the same time changes application status as well. |
| | AssignEvaluatorNumber. This test case assigns evaluator numbers to evaluators. |
| | AssignEvaluators. This test case assigns evaluators to applications. |
| | ChangeDeadline. This test case changes the submission deadline to allow evaluators to start his evaluation. |
| | CreateApplicationGroups. This test case creates application groups that can be selected during the creation of a solicitation. |
| | CreateSolicitation. This test case creates a solicitation. |
| Visitor | CreateApplicants. This test case allows creates applicant accounts. |
| | CreateEvaluators. This test case creates evaluator accounts. |

# CHAPTER SEVEN

# CONCLUSION AND FUTURE DIRECTIONS

## 7.1 Conclusion

This project was written to test the Solicitation Management System. Through testing, the goal is to find faults with the system. The project is divided into manual testing and automated testing with three and six subcategories defined in each respectively.

There are three tools that were used in this project: JUnit, HttpUnit, and JUnitPerf. These tools can be used to produce the functionalities we need in order to get the testing done.

There are several difficulties encountered in this project. The first difficulty is to find a way to upload a file using an automated test case. The second difficulty is to mean to validate the information within the pdf file. The last difficulty is to learn the language Jython in order to write scripts for load testing purposes.

The conclusion of this project is that relying solely on automated testing alone will not suffice the purpose of exposing as much defects as possible. With a combination of automated testing and manual testing, the goal can more likely be reached. One of the reasons that contribute to

more defect exposure is that during the client review prototype session testing, the client will sometimes test the system in ways that was not expected of use. This can reveal unforeseen faults.

## 7.2 Future Directions

The Test Driven Design was part of the original plan for testing. However, since the new component, the panel review section, for the Office of Technology Transfer and Commercialization was canceled; the TDD has not really been put into practice.

Future directions for the expanding this project is to implement Test Driven Design.

REFERENCES

[1] Bob Colwell. "If Your Didn't Test It, It Doesn't Work", Computer, May 2002.

[2] M. Hertzum. "User Testing in Industry: A Case Study of Laboratory, Workshop, and Field Tests", User Interfaces for All: Proceedings of the 5th ERCIM Workshop, November 1999.

[3] Tim Menzies and Bojan Cukic. "When to Test Less", IEEE Software, September/October 2000.

[4] Hema Srikanth and Laurie Williams. "On the Economics of Requirements-Based Test Case Prioritization", Proceedings of the Seventh International Workshop on Economics-Driven Software Engineering Research EDSER '05, May 2005.

[5] Gregg Rothermel, et al. "Prioritizing Test Cases For Regression Testing", IEEE Transactions on Software Engineering, October 2001.

[6] Sebastian Elbaum, et al. "Test Case Prioritization: A Family of Empirical Studies", IEEE Transactions on Software Engineering, February 2002.

[7] Yanping Chen, et al. "Specification-Based Regression Test Selection with Risk Analysis", Proceedings of the 2002 Conference of the Centre for Advanced Studies on Collaborative Research, September 2002.

[8] Mary Jean Harrold and Gregg Rothermel. "Empirical Studies of a Prediction Model for Regression Test Selection", IEEE Transactions on Software Engineering, March 2001.

[9] Todd L. Graves, et al. "An Empirical Study of Regression Test Selection Techniques", ACM Transactions on Software Engineering and Methodology, April 2001.

[10] JUnit Testing Framework; (http://junit.org/)

[11] HttpUnit Testing Framework; (http://httpunit.sourceforge.net/)

[12] JUnitPerf; (http://clarkware.com/software/JUnitPerf.html/)