5-2013

3-Way Test Suite Prioritization and Fault Detection: A Case Study

Arjun Roy Chaudhuri
Utah State University

3-WAY TEST SUITE PRIORITIZATION AND FAULT DETECTION: A CASE STUDY

by

Arjun Roy Chaudhuri

A report submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

_____	_____
Dr. Nicholas Flann	Dr. Vicki Allan
Major Professor	Committee Member

_____
Dr. Dan Watson
Committee Member

UTAH STATE UNIVERSITY
Logan, Utah

2013

ABSTRACT

3-Way Test Suite Prioritization and Fault Detection: A Case Study

by

Arjun Roy Chaudhuri, Master of Science

Utah State University, 2013

Major Professor: Dr. Nicholas Flann
Department: Computer Science

GUI and web based applications are becoming universal. Functional accuracy of those applications is vital. Software defects caused by poor software testing can cost billions of dollars. Further, web application defects can be costly due to the fact that most web applications handle regular user interaction. By improving the time efficiency of software testing, many of the costs associated with defects can be saved. Web application users generate large numbers of possible test-cases and out of all those test-cases only some of them are vital for functional testing. Therefore testing correctness of these applications is expensive and time consuming and hence challenging at times. However, software testing is often under time and budget constraints. Earlier studies came up with different abstract models to face this kind of challenges where a tester can select and execute a subset of all the possible test-cases (*test-case prioritization*) based on some criterion to assure performance goal. In the context of test suite prioritization, earlier studies showed that 2-way inter-window interaction coverage/criteria are effective at finding faults quickly in the test execution cycle. However, since faults may be caused by interactions between more than 2 parameters, in this project we exercise test suite prioritization by t-way combinatorial coverage of inter-window interactions on an existing web application Music-Store. Our results show that the rates of fault detection for 2-way and 3-way prioritization are very close to each other.

ACKNOWLEDGMENTS

Most significantly, I would like to convey my hearty gratitude to Dr. Nicholas Flann for giving me the opportunity in this project. He has been an outstanding advisor and guide. His guidance and support throughout my graduate career and during the course of this project are invaluable. The experience and knowledge I have gained throughout my graduate course work with him will stay with me for years to come. I am grateful to Dr. Dan Watson and Dr. Vicki Allan for both of their input and interest in this report.

I would also like to make a special thanks to Dr. Sampath and Dr. Renee Bryce; their help through my graduate research experience has been truly rewarding. It was nice learning experience to use different software testing tools like Replay tool, Test Oracle, CPUT made and maintained by her and her students. I would like to thank undergraduate and graduate researchers Chelynn Day and Schuyler Manchester for their contributions to the open-source software testing tool CPUT.

I would like to thank my parents for their continuous support and encouragement without which pursuing Master's degree in Utah State University would not have been possible. Finally, I thank my friends for their support and interest.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Figure Page

LIST OF TABLES

Table Page

I. INTRODUCTION

Software testing is an expensive and time-consuming activity that is often restricted by limited project budgets. Accordingly, the National Institute for Standards and Technology (NIST) reports that software defects cost the U.S. economy close to $60 billion a year [1]. They suggest that approximately $22 billion can be saved through more effective testing. There is a need for advanced software testing techniques that offer an effective cost-benefit ratio in identifying defects.

Due to their user-centric nature, web applications routinely go through changes as part of their maintenance process. In such situations, a large number of test-cases may be available from testing previous versions of the application that are often reused to test the new version of the application. However, running such tests may take a significant amount of time. Due to time constraints, a tester must often select and execute a subset of these test-cases, which is known as test-case prioritization [2].

Test-cases can be prioritized based on different criteria [3]. One of the criteria is parameter value interaction coverage based criteria. Interactions between multiple parameter-values make an application program follow a distinct execution path, and thus it delivers faults in the system. 1-way, 2-way, and n-way parameter value interaction coverage are possible. Interactions include combinations of options for different parameters. For example, a 2-way interaction for an online community can be [(new member, basic membership) or (new member, priority membership)]. Parameter-value interaction coverage is useful when exhaustive testing of all parameter-option interactions is not possible. Very recently, test suites have been prioritized by 2-way inter-window event coverage for event-driven systems, i.e., web and GUI systems [3]. Previous work introduces test prioritization to the domain of web applications and prioritizes user-session-based test-cases, i.e., test-cases created from usage logs of the web system [3]. Though 2-way is one of the best prioritization criteria, observations from the latest research and studies have shown some faults are missed by 2-way interaction test suites, so we decided to investigate higher strength prioritization strategies, such as 3-way.

Outline of Report

Chapter 2 discusses background and previous work on higher strength prioritization strategies (2-way, 3-way), while Chapter 3 illustrates existing algorithms for higher strength

prioritization technique (t-way prioritization) used in this project. Chapter 4 provides information on experiments that have been done to investigate the efficiency of t-way prioritization techniques in terms of fault detection. Chapter 5 summarizes the results, and Chapter 6 demonstrates the web application Music Store. Chapter 7 concludes the project and discusses the scope of possible future work.

Appendix A contains the details of CPUT. Appendix B shows sample XML test suites. Appendix C shows the code coverage details and Appendix D provides information on various faults seeded in Music Store application for this project.

II. BACKGROUND AND PREVIOUS WORK

Our study applies to test suite prioritization in the domain of web applications where we prioritize user session based test-cases. User session based test-cases are typically those that are created from the usage logs of web servers. So here, we will discuss related work in two areas: (1) web applications and user-session-based testing, and (2) test suite prioritization.

Web Application Testing

A web application consists of a set of pages that are accessible by users through a browser and are transmitted to the end-user over a network. A web page can be static—where content is constant for all users—or dynamic—where content changes with user input. Web applications exhibit characteristics of distributed, GUI, and traditional applications. They can be large with millions of lines of code and may involve significant interaction with users. Also, web applications are written using many programming languages, such as JavaScript, Ajax, PHP, ASP, JSP, Java servlets, and HTML. Languages such as JavaScript are referred to as client-side languages, whereas languages such as PHP, ASP, JSP are referred to as server-side languages. Even a simple web application can be written in multiple programming languages, e.g., HTML for the front-end, Java or JSP for the middle tier, and SQL as the back-end language—which makes testing difficult.

In web applications, an event can manifest itself in two ways: (1) an event triggered in the client-side code by a user results in a change to the page displayed to the user, without any server-side code execution, e.g., when a user moves the mouse over an HTML link, an event may be triggered that causes the execution of a JavaScript event handler, which in turn results in the link changing color; (2) an event is triggered in the client-side code by a user that results in server-side code being executed, e.g., when the user fills in a form and clicks on the submit button, the data is sent to a server-side program, and the server-side program executes and returns the outcome of the execution to the user. In our work, we focus on the latter types of events, i.e., events triggered by a user that result in server-side code execution, as they are readily available in the form of POST or GET requests in server web logs; we use the logs as the source for our web application test-cases.

Web application testing, can be defined as implementing the entire application code by generating URL-based inputs with the intent of finding failures that occur in output response HTML pages. Testing of web program code to identify faults in the program is largely a manual task.

Capture-replay tools capture tester interactions with the application and are then replayed on the web application [9].

Web application testing research has explored techniques to enable automatic test-case generation. Several approaches exist for model-based web application test-case generation [4, 5, 6, 7, 8, and 9]. These approaches investigate the problem of test-case generation during the development phase of an application. Another approach to generating test-cases, and the one used in this paper is called user-session-based testing; it advocates the use of web application usage data as test-cases [10].

In *user-session-based testing***,** a test-case is a series of HTTP requests having base requests and name-value pairs that are recorded when a user accesses the application. Fig. 1 shows a user session of music store web application and Fig. 2 shows an example of a test-case from that user session for following request: Login.php&name="arjun"&pwd="admin", the base request is Login.php and the parameter-value pairs are name="arjun" and pwd="admin". Base requests can be HTTP request accesses to both static and dynamic web page content. In previous work, Sampath et al. [11] and Sprenkle et al. [12] generate user-session-based test-cases from usage logs. When available, cookies were used to generate a user-session based test case. Otherwise, a user-session-based test-case begins when a request from a new IP address arrives at the server and ends when the user leaves the web site or the session times out.

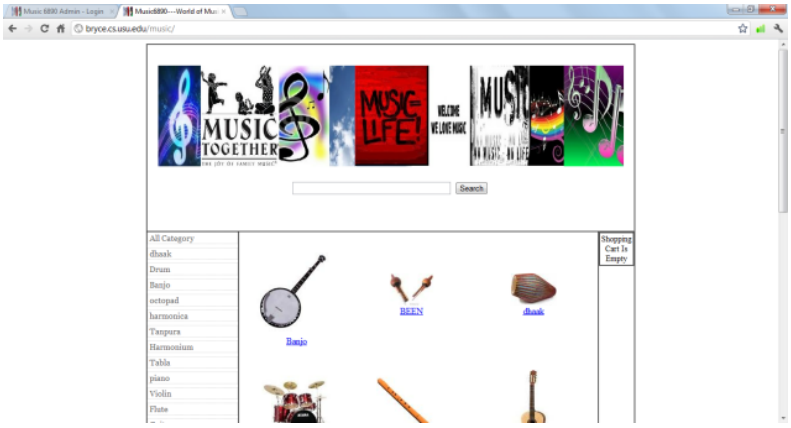

Fig. 2.1 example of a user session of music store web application

index.php search.php?query=mandolin&search=1 login.php?name=arjun&pwd=admin	
Base Request	**Parameter-Value pairs**
index.php	Null
login.php	name=arjun, pwd=admin
search.php	query=mandolin,search=1

Table 2.1 example of base request and parameter-value pair

Test Suite Prioritization

To transform a user session into a test-case, each logged request is changed into an HTTP request that can be sent to a Web server. A test-case consists of a set of HTTP requests that are associated with each user session. Different strategies can be applied to construct test-cases for the collected user sessions [9, 13, 17]. In such situations, a large number of test-cases may be available from testing previous versions of the application, which are often reused to test the new version of the application.

However, running such tests may take a significant amount of time. Rothermel et al. report an example for which it takes weeks to execute all of the test-cases from a previous version of the application [2]. Due to time constraints, a tester must often select and execute a subset of these test-cases. Test-case prioritization is the process of scheduling the execution of test-cases according to some criterion to satisfy a performance goal.

Consider the function for test prioritization as formally defined in [2, 14]. Given T, a test suite, Π, the set of all test suites obtained by permuting the tests of T, and f, a function from Π to the set of real numbers, the problem is to find $\pi \in \Pi$ such that $\forall \pi\_ \in \Pi, f(\pi) \geq f(\pi\_)$. In this definition, Π refers to the possible prioritizations of T and f is a function that is applied to evaluate the orderings. The selection of the function f leads to many criteria to prioritize software tests.

Sarah 4/8/13 10:17 PM

Comment [1]: You need to list the author here. The numbers will not do.

Previous Work

Web Application Testing

Today, a lot of different techniques are available for generating test-cases for web applications. For example, tools like HTTPUnit [27] and RationalRobot [28] let testers record test sequences and measure performance. Some tools verify broken links, validate HTML code, and measure performance. Another example is Veriweb, which offers a simple solution that starts at a given URL and non-deterministically navigates links in a web application [21]. Kung et al. added object relations, state, and page navigation diagrams in a web test model (WTM) [22]. Ricca and Tonella [6] used UML models to automatically generate test-cases for white box testing. Liu et al. used data flow interactions among clients [23]. Halfond and Orso [8] revealed web application interfaces from server code. Wang et al. found interaction faults by generating test-cases that cover pair wise interactions between five web pages [7]. Offut et al. used HTTPUnit and HtmlUnit to run bypass tests that bypass client-side checks [24]. Qian [25] used a genetic algorithm utilizing crossover and mutations to generate a large volume of test-cases for a test suite.

Cohen et al. [26] studied one test generating framework: automatic efficient tests generator (AETG). In that experiment, pair-wise test sets that were generated by AETG gave over 90% block coverage. They also did a comparison of pair-wise testing and random input testing and found better coverage for pair-wise testing.

All of these strategies generate test-cases from models of the web system. Additional work to test rich internet applications exists, but such work is outside of the scope of this project. We focus on a particular type of web testing that occurs during the maintenance phase of the system, user-session-based testing.

User Session Based Testing

Elbaum et al. have completed empirical studies and showed that user-session-based testing is a good way to enhance white box testing techniques as they found various faults [29]. Sampath et al. [31] and Sprenkle et al. [12] provided a framework for user-session-based testing of web systems. As per their extended work, test-cases are formatted in XML format and parsed from Apache web server [19]. Even though user-session-based testing has advantages, it has two major inconveniences: (1) user-sessions may become invalid during regression testing (i.e., the structure of the web

application changes, including page names, links, options on a page, etc.), and (2) a large number of user-sessions build up, making it unrealistic to run all tests in practice. Alshahwan and Harman [33] present work on the first issue of repairing user-session-based test-cases for use in regression testing. Two approaches have been taken to address the second issue of managing large test suites: test suite prioritization [3, 30] and reduction [11, 32].

Elbaum et al. [13] provided promising results that demonstrate the fault detection capabilities and cost effectiveness of user-session-based testing. Their user session-based techniques discovered certain types of faults; however, faults associated with rarely entered data were not detected. In addition, they observed that the effectiveness of user-session-based testing improves as the number of collected session's increases; however, the cost of collecting, analyzing, and replaying test-cases also increases. User-session-based testing techniques are complementary to the testing performed during the development phase of the application [2], [14], [15], [16]. In addition, user-session-based testing is particularly useful when the program specifications and requirements are not available for test case generation.

Xie et al. examined the characteristics of a good GUI test suite. The authors found there are two primary characteristics that increase the rate of fault detection: (1) diversity of states in which an event executes, and (2) the event coverage of a test suite. Several criteria have been applied for test suite prioritization to user-session-based test suites.

Test-Suite Prioritization

Previous work by Bryce and Memon [34] examines 2-way and 3-way inter-window event coverage for test suite prioritization on GUI applications. For each application, they applied 2-way and 3-way inter-window event coverage, unique event coverage, length of test-cases (longest to shortest and shortest to longest in terms of the number of parameter- values) and random ordering. The first application, a calculator, only had two windows, so with the exception of 3-way, each technique was applied. The results show that 2-way provides the best rate of fault detection. In the other three applications, there were three or more windows so the authors were able to apply all of the prioritization criteria. For a paint program, choosing the longest tests first resulted in the best rate of fault detection, followed by 3-way and finally 2-way. For a spreadsheet program, unique event coverage provided the best rate of fault detection in the first half of the test suite, but then 2-way and

3-way alternated in providing the overall best rate of fault detection in the latter half of the test suite. Finally, in a word-processing application example, 2-way and 3-way alternated in producing the best rate of fault detection. This work provides some motivation to explore the application of 3-way inter-window parameter-value interaction coverage in the domain of web applications.

Bryce et al. [3] examine several prioritization criteria, including the combinatorial criterion, pair-wise inter-window parameter-value interaction coverage (2-way), applied to user-session-based test suites, and empirically evaluate them on three web applications, including an online bookstore, a course project manager (CPM), and a conference management system. All three applications were seeded with faults, i.e. bugs were added in the applications. They found that prioritization criteria based on the longest tests with respect to the number of POST/GET requests, longest tests with respect to the number of parameters that users assigned values, and 2-way combinatorial coverage of inter-window inter- actions were usually efficient techniques compared to the original order in which test-cases were logged or ordered at random.

However, since existing literature recognizes that certain faults are detected by interactions between parameters that are stronger than pair-wise interactions (2-way), we will demonstrate competence and efficacy of the 3-way combinatorial interaction coverage in terms of rate of fault detection with less memory usage and in less time for our music store web application and user-session-based testing.

III. ALGORITHM

Here we will explain t-way test suite prioritization technique for t = 2 and t = 3 and discuss the existing prioritization algorithm using a simple example.

Example of t-way Prioritization

Consider an example of an online community where different membership options are possible. Table 3.1 shows the four possible pages of that online community web application. In the first page, the user may appear as one of the three options for the member status. There after user may select one of the three membership types in the second page. On third page user may choose one of the three discount offers. On the final page the user can opt for any of the three annual gift options.

Page-1, Member Status	**Page-2, Member Type**	**Page-3, Monthly Discount offer**	**Page-4, Annual Gift offer**
New Member Unverified	Basic	N/A	Up to $5 Wal-Mart gift card
New Member Verified	Silver	10% off on $1000 purchase in eBay	Up to $50 Wal-Mart gift card
Existing Member	Gold	20% off on $500 purchase in eBay	Up to $500 Wal-Mart gift card

Table 3.1 Web Testing Example with Four Factors and Three Levels for Each Factor

Selecting different options will execute different lines of code in the system. For instance, if the user selects any discount option other than “N/A”, the system generates a unique discount value and takes him to next page that describes conditions of corresponding discount offer. Thus, having test coverage for the several values for tracking could potentially uncover a fault that might have been overlooked by a different test. Table 3.2 shows an example test suite for the above application example in Table 3.1. This test suite contains a total of 3 different test-cases.

Test case #	Member status	Member type	Discount offer	Annual Gift offer	Parameter-Value
T1	New Member, unverified	Basic	N/A	$5 gift card	Status: New unverified Type: Basic Discount: N/A Gift: $5
T2	New Member, verified	Gold	10% off	$500 gift card	Status: New verified Type: Gold Discount: $10 Gift: $500
T3	Existing Member	Silver	20% off	$50 gift card	Status: Existing Type: Silver Discount: $20 Gift: $50

Table 3.2 Test Suite Example of Table 3.1 Web Testing Example

Existing Algorithm for t-way Prioritization

Based on the above example we will now demonstrate the existing algorithm for t-way prioritization. Figure 3.1 provides the pseudo code of the t-way prioritization algorithm. There are two parts in this algorithm.

Part 1 processes each test-cases in the test suite to create the parameter-value combinations for each page/URL and store them in memory as tuples (t-way). Part-2 does the actual t-way prioritization on the tuples stored in memory.

Part-1 (Pre-processing of test suite)

```
sizeOfTestSuite = 0
foreach test case t(i) in the test suite TS
{
        foreach URL w in t(i)
        {
                foreach parameter p assigned a value v in t(i)
                {
                        tuple t(x) = w + p + v
                        if tuplesList does not have t(x)
                                add t(x) in tuplesList
                }
        }
        sizeOfTestSuite++
}
// Generate all t-tuple combinations and insert into t-tuplesList
t-wayTuplesList = generate-t-wayTupleList(tuplesList)
```

Part-2 (Test Suite Prioritization)

```
test(bestTestCase) = choose a testcase that covers the most unique t-way tuples from t-wayTuplesList
add test(bestTestCase) to prioritized test suite TS(priority)
mark test(bestTestCase) as covered
delete t-way tuples that is present in test(bestTestCase) from t-wayTuplesList
selectedTestCount = 1
do
{
        tCountMax = -1
        for counter=1 to (sizeOfTestSuite-selectedTestCount)
        {
                if test(counter) is not covered
                {
                        compute tCount as the number of newly covered t-way tuples from t-wayTuplesList
                        in test(counter)
                        if (tCount > tCountMax)
                        {
                                tCountMax = tCount
                                test(bestT est)= counter
                                }
                        else if (tCount == tCountMax)
                                choose randomly
        }
        add test(bestTestCase) to TS(priority)
        mark test(bestTestCase) as covered
        delete t-way tuples that is present in test(bestTestCase) from t-wayTuplesList
        selectedTestCount++
}while (selectedTestCount < sizeOfTestSuite)
```

Fig. 3.1 Algorithm for test suite prioritization by t-way combinatorial coverage

Now we will explain the algorithm based on the example given in section 3.1.For the above algorithm, certainly input will be the test suite shown in Table 3.2. After the execution of Part 1 of the algorithm we will be able to get the t-way tuples (in this case 2-way and 3-way tuples are shown). Table 3.3 shows the deduced 2-way and 3-way parameter values covered from the example test

suites given in Table 3.2. The tuples stand for the inter-window parameter-value interactions in the test case.

Test case #	2-way tuples covered	3-way tuples covered
T1	[Status: new/unverified, type: basic] [Status: new/unverified, offer: N/A] [Status: new/unverified, gift: $5] [Offer: N/A, gift: $5] [Offer: N/A, type: basic] [Type: basic, gift: $5]	[Status: new/unverified, type: basic, offer: N/A] [Status: new/unverified, type: basic, gift: $5] [Status: new/unverified, offer: N/A, gift: $5] [type: basic, offer: N/A, gift: $5]
T2	[Status: new/verified, type: gold] [Status: new/verified, offer: 10%] [Status: new/verified, gift: $500] [Offer: 10%, gift: $500] [Offer: 10%, type: gold] [Type: gold, gift: $500]	[Status: new/verified, type: gold, offer: 10%] [Status: new/verified, type: gold, gift: $500] [Status: new/verified, offer: 10%, gift: $500] [type: gold, offer: 10%, gift: $500]
T3	[Status: existing, type: basic] [Status: existing, offer: 20%] [Status: existing, gift: $50] [Offer: 20%, gift: $50] [Offer: 20%, type: silver] [Type: silver, gift: $50]	[Status: existing, type: basic, offer: 20%] [Status: existing, type: silver, gift: $50] [Status: existing, offer: 20%, gift: $50] [type: silver, offer: 20%, gift: $50]

Table 3.3 2-way and 3-way P-V covered from test suite in Table 3.2

As per Part 2 of the algorithm, prioritizing by 2-way, we select the first test-case such that it covers the largest number of 2-tuples. The second column of Table 3.3 shows that all four test-cases cover six 2-tuples. We then break the tie at random, select T2, and mark the 2-tuples in this test as covered. We next examine which of the remaining tests cover the most remaining uncovered 2-tuples. Again there is a tie between T3 and T1 as both of them have six uncovered 2-tuples. We select T1 and mark it as covered. Hence the ordering for the 2-way prioritization is (T2, T1, and T3).

To prioritize by 3-way, we select the first test case that covers the most 3-tuples. All four test-cases cover four 3-tuples, so we break the tie at random and select T3. We next examine which of the remaining tests cover the most remaining uncovered 3-tuples. Again there is a tie between T2

and T1 as both of them have four uncovered 3-tuples. We select T1 and mark it as covered. Hence the ordering for the 3-way prioritization is (T3, T1, and T2).

IV. EXPERIMENT

Subject Application

Our subject application is a web-based application called Music Store. It is written in PHP and uses MySQL as database server. It runs on an Apache 2.2 web server. Music Store is a simple web application such as online stores like eBay/Amazon, but the functionalities built within the application are very basic to use it for a web testing research purpose. There are two different types of users that can log into this application: (1) admin, and (2) common user. Music Store Web application is discussed in detail in Chapter 6. Figure 4.1.1 and Figure 4.1.2 show the home pages of common user and admin of music store application.

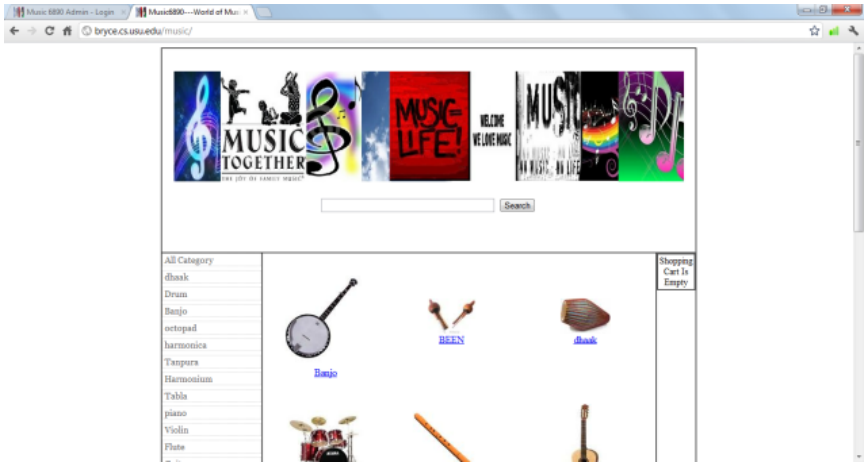

Fig. 4.1.1 Common user Home page view of Music Store web application

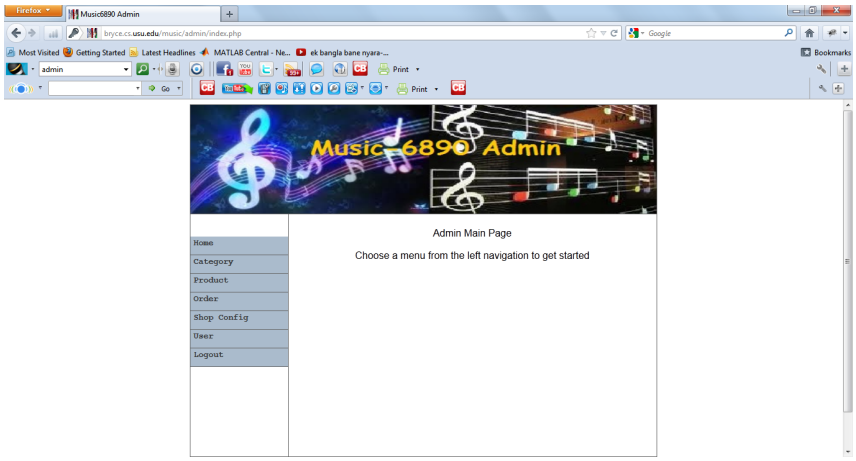

Fig. 4.1.2 Admin Home Page view of Music Store web application

Test Suites

The test suites for this study were gathered by a graduate student instructor. As an instructor of an undergraduate computer science class, he instructed his students to login to the web application and test out as many web pages as possible. The test-cases are constructed using the IP addresses that are associated with each GET/POST request. As per standard, if there is more than a 45-minute break in between a GET/POST request from the same user, we begin a new test-case. We initially collected a large test suite with 165 test-cases. Appendix B shows a detailed sample test-case and appendix C shows the code coverage information of each of those test-cases.

Faults

A total of 68 faults were seeded into Music Store by a graduate student. Each seeded faulty version belongs to two categories: (1) user type, and (2) fault classification. We seeded faults for each type of user of the system, i.e., admin and common user. Sampath et al. [11] and Guo and Sampath [18] presented a fault classification for web applications, which we used when seeding faults in Music Store (described below).

- *Appearance faults*: Faults in the application code that change the display of a web page. An example is that a missing print statement in the PHP code can sometimes cause the code to display in an incorrect manner.

- *Link faults*: Faults in the application code that manipulate the page pointed to by a URL. An example is a link that points to a non-existent page causing an error to display.
- *Data Store faults*: Faults in the code that modify data storage within the application. An example includes swapping variables for an SQL Insert query, which causes the data to be stored improperly.
- *Form faults*: Faults in the application code that manipulate a form's name-value pairs. An example includes swapped variables for the text and values for an HTML option list, which causes incorrect text to be displayed, and incorrect values to be sent to the server.
- *Logic faults*: Faults in the application code that manipulate control flow and/or business logic. An example is displaying an improper date format that causes the date to be saved incorrectly.

Logic faults have seven subcategories, of which we use five because the remaining categories were not applicable for our subject application, Music Store. The five subcategories we used are:

- *Session faults*: Faults in the application code that manipulate the current session state of the application or faults that manipulate other session-based operations such as using sessions to save information entered on a form and display the information after the sessions have been validated. An example is accidentally setting a variable that determines what menu navigation screen is displayed; this causes undesired behavior.
- *Paging faults*: Faults in the application code that manipulates the display of large amounts of data. An example is using a `<' instead of a `<=' when iterating through the pages of users, which causes the last page of users to never be displayed.
- *Server-side parsing faults*: Faults in the application code that change server-side parsing of data. An example is an escaped variable that causes the variable name to be saved instead of the value assigned to that variable.
- *Encoding/decoding faults*: Faults in the application code that encode or decode information during transmission, storage, and/or display. An example is a missing

convert from a database function that causes the data to not be decoded into a more readable format.

- *Locale faults*: Faults that exist in code that manipulate locale-specific information within the application, such as date format or language.

Appendix D shows details on the fault type distribution.

Prioritization Criteria

For our study we have following prioritization criteria in the prioritization tool CPUT [19]:

1. **2-way** orders test-cases in descending order of the number of unique 3-way parameter-value interactions between windows in each test case. Once a pair is covered in a test, we mark it as "covered" and only count unique pairs that have not been covered in previously selected tests. Ties are broken at random.
2. **3-way** orders test-cases in descending order of the number of unique 3-way parameter-value interactions between windows in each test case. Once a pair is covered in a test, we mark it as "covered" and only count unique pairs that have not been covered in previously selected tests. Ties are broken at random.
3. **Length (Gets/Posts)** selects the test-cases in descending order of the number of GET/POST requests. Ties are broken at random.
4. **Number of parameter-values** selects the test-cases in descending order of the number of parameter-values. Ties are broken at random.
5. **Random** ordering uses the random function that is available in Java to randomly swap the ordering of the test-cases. The tool will produce a different random ordering each time that the user chooses to prioritize at random.

Experiment Framework

The usage logs for music store are converted into test-cases and then prioritized within our tool, CPUT [19]. The n-way prioritization algorithm is implemented in CPUT for n = 2 and n = 3, in addition to other criteria, such as length, random, and frequency-based. Apache logs will be parsed and XML format test-cases are created. The test-cases are then be prioritized using the different prioritization criterion.

We then executed the test-cases using a replay tool we created that could execute the XML format test-cases. We also conducted the fault detection experiments using the framework presented by Sprenkle et al. [20]. Initially, we will execute the test-cases on a clean version of the application and save the returned files. That will be the expected output, since we consider the non-fault-seeded version of the application as our gold standard. Then, one fault will be seeded in the application at a time, and all the test-cases will be executed. The returned HTML files are saved (this is the actual output). The test oracle will then be executed on the returned files to determine if the test-case detects the fault. The struct-oracle [20] compares the expected and actual output in terms of the HTML tags in the files, to identify differences. A fault matrix will be generated that will show how many faults and which faults are detected by each test-case.

After that we computed Average Percentage of Fault Detection (APFD) [2]. APFD measures the area under the curve that plots test suite fraction and the number of faults detected by the test ordering.

V. RESULTS

Here we will present our findings about the effectiveness of prioritization by the 3-way inter-window parameter value interaction coverage in terms of fault-detection for our application Music Store, and later we will show the effectiveness of our algorithm in terms of space and execution time measurement.

We executed our algorithm five times for each of the criterion and we got the results as average of those executions. Our experiment shows 2-way and 3-way are little better than others.

Table 5.1 and Fig. 5.1 show average APFD (Average Percentage of Faults Detected) data for common users of the Music Store web application. Table 5.2 and Fig. 5.2 show average APFD data admin of the Music Store web application. Out of total 68 faults the full test suite detected 49.

From the plotting it clearly shows that 2-way is a little better in comparison with the other prioritization criteria for both user and admin test suites. The next best criteria are 3-way. P-V and GET/POST criterions are very close to each other, though for both admin and common user test suites GET/POST criterion has performed slightly better than P-V.

% of test suite	2-way	3-way	GET/POST	P-Vs	Random
10%	65.7	65.8	61.1	60.4	60.5
20%	69.9	68.4	64.6	63.9	61.9
30%	70.1	68.4	64.6	63.9	62.6
40%	70.1	69.1	66.4	65.8	63.2
50%	70.9	69.1	66.6	65.8	63.8
60%	70.6	69.52	66.9	65.8	64.1
70%	70.7	69.7	67.4	66.4	64.5
80%	71.1	69.9	67.4	66.5	64.7
90%	71.1	70.3	67.4	66.5	64.9
100%	71.2	70.4	67.6	66.8	64.9

Table 5.1 Average APFD of the Common User Test suites

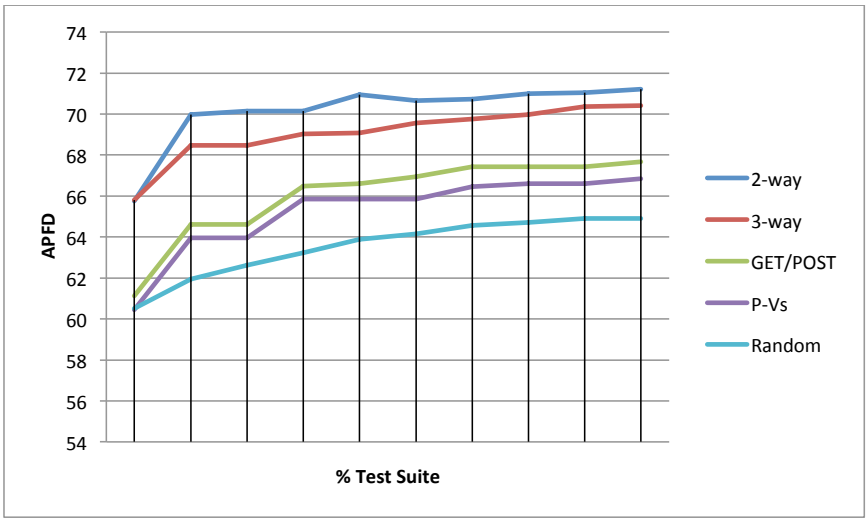

Fig. 5.1 APFD plotting for Common User test suite of Music-Store web application

% of test suite	2-way	3-way	GET/POST	P-Vs	Random
10%	67.9	65.2	54.1	60.8	55.1
20%	79.2	76.3	62.2	61.3	59.4
30%	79.2	78.7	64.5	63.5	61.1
40%	81.3	78.1	67.8	64.8	61.9
50%	81.3	79.1	67.8	65.6	62.3
60%	81.3	79.1	67.8	66.2	62.9
70%	82.8	79.1	68.8	66.7	69.8
80%	83.4	81.9	69.8	66.9	72.2
90%	90.2	84.1	85.6	69.6	78.8
100%	92.4	84.1	85.6	73.7	78.9

Table 5.2 Average APFD of the Admin Test suites

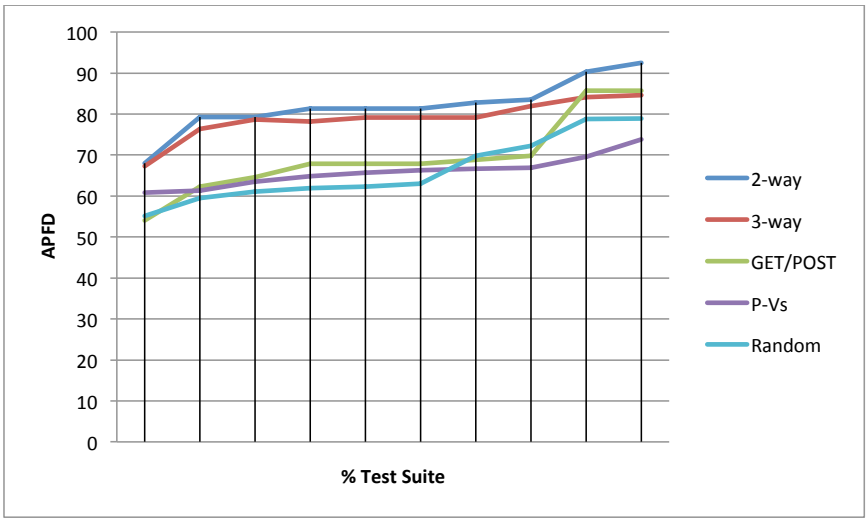

Fig. 5.2 APFD plotting for Admin test suite of Music-Store web application

Next we are going the present the observations for time and memory efficiency of our t-way prioritization algorithm. To examine this, we measured each component's execution time and space requirement of the output. The experiment was run on a machine with a Windows 7 OS with 3 GB of RAM and with an Intel i3 processor with 2.40 GHz speed. In this study, we split the log into 1-day usage, 3-day usage, 5-day usage, and 12-day usage (which is the entire log file), and doubled the size of the log file. To double the size of the log file, we modified the log file by changing the year from 2011 to 2012 to manually create different usage logs.

We present the results for each log file separately. For each log file, we present the time taken by the test-case creation engine and the different prioritization and reduction criteria (column 4). We also present the space occupied by the output of the different components of the framework (column 5). Execution time shows time taken for parsing the web log into test-cases, storing the data from the test suite and prioritization of the test suites.

Log File	Component name	Component output	Execution Time	Output Space
1 day	Test-case creation engine	XML format Test-case	0.003	31 kb (4 test-cases)
	Test Prioritization (Length)	Order file	0.003	42 bytes
	Test Prioritization (No of Parameters)	Order file	0.005	42 bytes
	Test Prioritization (2 way)	Order file	0.043	42 bytes
	Test Prioritization (3 way)	Order file	0.072	42 bytes
3 day	Test-case creation engine	XML format Test-case	0.005	230 kb (19 test-cases)
	Test Prioritization (Length)	Order file	0.004	111 bytes
	Test Prioritization (No of Parameters)	Order file	0.004	111 bytes
	Test Prioritization (2 way)	Order file	0.049	111 bytes
	Test Prioritization (3 way)	Order file	0.996	111 bytes
5 day	Test-case creation engine	XML format Test-case	0.009	302 kb (51 test-cases)
	Test Prioritization (Length)	Order file	0.008	293 bytes
	Test Prioritization (No of Parameters)	Order file	0.004	293 bytes
	Test Prioritization (2 way)	Order file	0.422	293 bytes
	Test Prioritization (3 way)	Order file	1.08	293 bytes
12 day	Test-case creation engine	XML format Test-case	0.013	473 kb (76 test-cases)
	Test Prioritization (Length)	Order file	0.008	352 bytes
	Test Prioritization (No of Parameters)	Order file	0.005	352 bytes
	Test Prioritization (2 way)	Order file	0.536	352 bytes
	Test Prioritization (3 way)	Order file	1.137	352 bytes
double	Test-case creation engine	XML format Test-case	0.023	756 kb (165 test-cases)
	Test Prioritization (Length)	Order file	0.008	678 bytes
	Test Prioritization (No of Parameters)	Order file	0.005	678 bytes
	Test Prioritization (2 way)	Order file	0.805	678 bytes
	Test Prioritization (3 way)	Order file	1.885	678 bytes

Table 5.3 Execution Time and Size of Test Suites for Music-Store Logs

Table 5.3 summarizes the results. From these results, we note that the time taken by the t-tuple prioritization algorithm is in the order of a few seconds. Therefore, our algorithm has the potential to scale to larger usage logs and test-cases on which the test prioritization criteria need to be applied.

VI. MUSIC STORE WEB APPLICATION

Introduction

The Music Store web application we made here is a basic one without any sophisticated features. The store has admin pages (where the shop admin can create categories, add products, etc), and shopper pages where all the shopping process take place.

After a user browses around she/he will see that the basic flow of the store is:

1. A customer visit the site
2. She/he browse the pages, clicking between categories
3. She/he can search product/categories.
4. View the product details that she/he found interesting
5. Add products to shopping cart
6. Checkout (entering the shipping address, payment info)
7. Leave (hopefully to return another time)

The customer doesn't need to register for an account.

Database Design

The database design for our shopping cart is quite simple. Below is the summary of what tables we need for this shopping cart plus the short description of each table.

. **Table Name**	**Description**
tbl_category	Storing all product categories
tbl_product	Storing all the products
tbl_cart	When the buyer decided to put an item into the shopping cart we'll add the item here
tbl_order	This is where all orders are saved
tbl-order_item	The items ordered
tbl_user	Stores all shop admin user account
tbl_shop_config	Contain the shop configuration like name, address, phone number, email, etc

Table 6.1 Database table summary of the Music Store Web Application

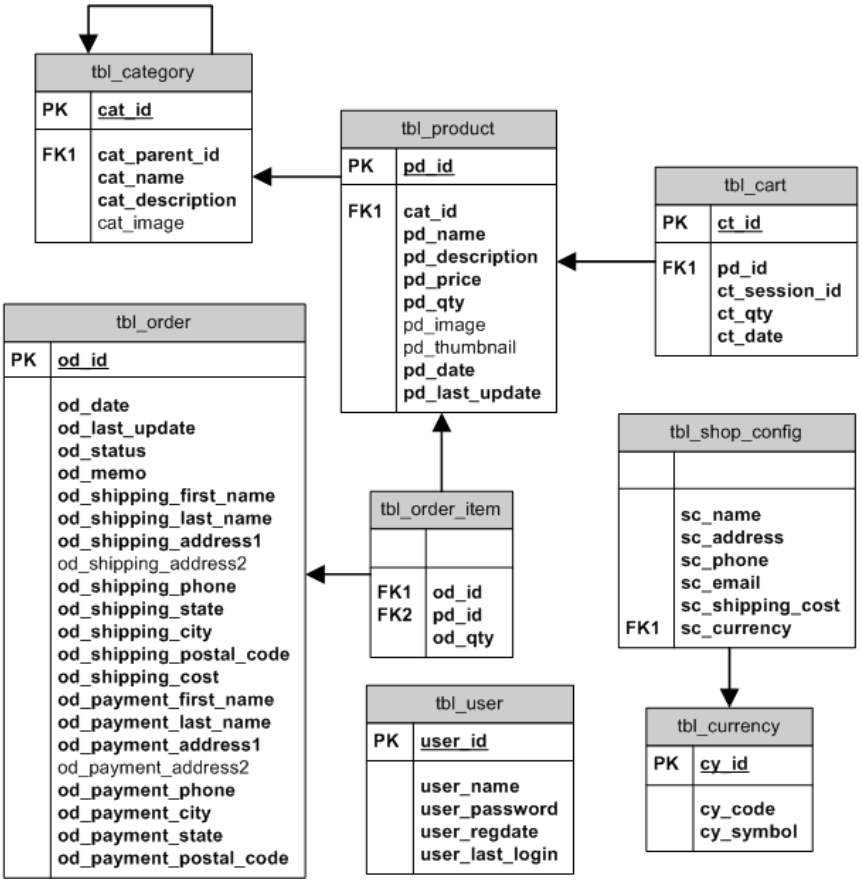

Fig. 6.2 Entity Relationship (ER) diagram of the Music Store web application database

Admin Features

Our music store admin page consists of the following:

- Login: The admin enter its username and password, script check whether that username and password combination do exist in the database. If it is set the session then go the admin main page else show an error message. Figure 6.3.1 shows an admin login page screen shot.

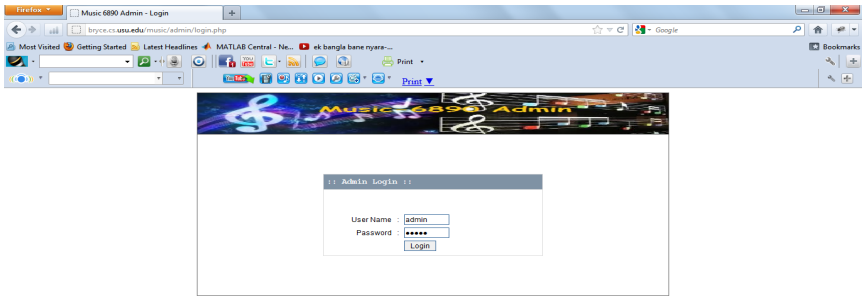

Fig. 6.3.1 Admin Login Page view of Music Store web application

- Category:
  - Add Category: Add a new category.
  - View Category: List all the category we have. We can also see all the child categories and show many products in each category
  - Modify Category: Update a category information, the name, description and image
  - Delete Category: Remove a category.

Figure 6.3.2, Figure 6.3.3, and Figure 6.3.4 show the corresponding screenshots.

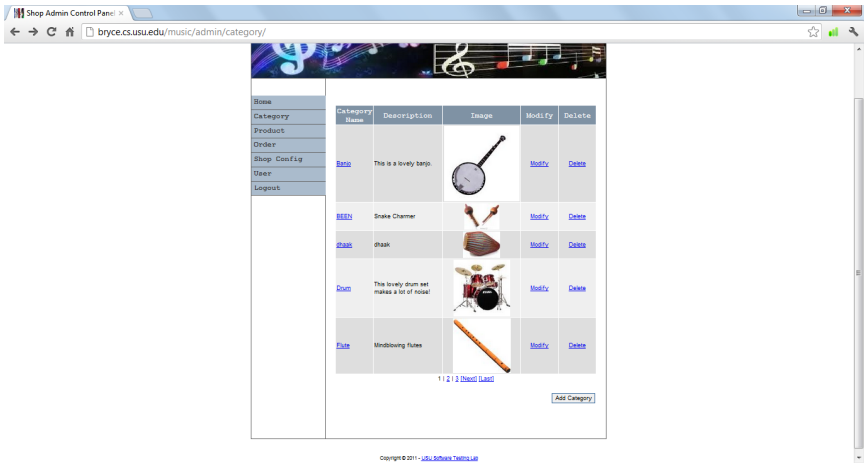

Fig. 6.3.2 Admin View/Delete Category Page of Music Store web application

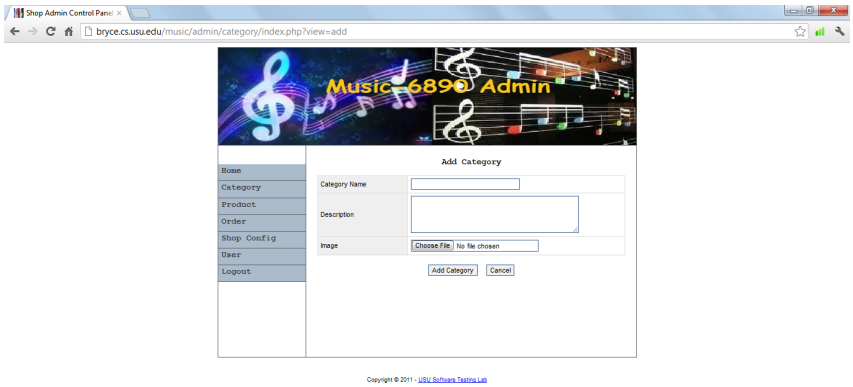

Fig. 6.3.3 Admin Add Category Page of Music Store web application

Fig. 6.3.4 Admin Modify Category Page of Music Store web application

- Product:
  - Add Product: Insert an item into our store. We also need to supply the product image and we'll create a thumbnail automatically from this image.
  - View Product: View all the products we have. Since our online shop can have many products we can view the products grouped by category.

- Modify Product: Modify product information. We can also remove the product image from this page.
- Delete Product: Remove a product from the shop.

Figure 6.3.5, Figure 6.3.6, and Figure 6.3.7 show the corresponding screenshots.

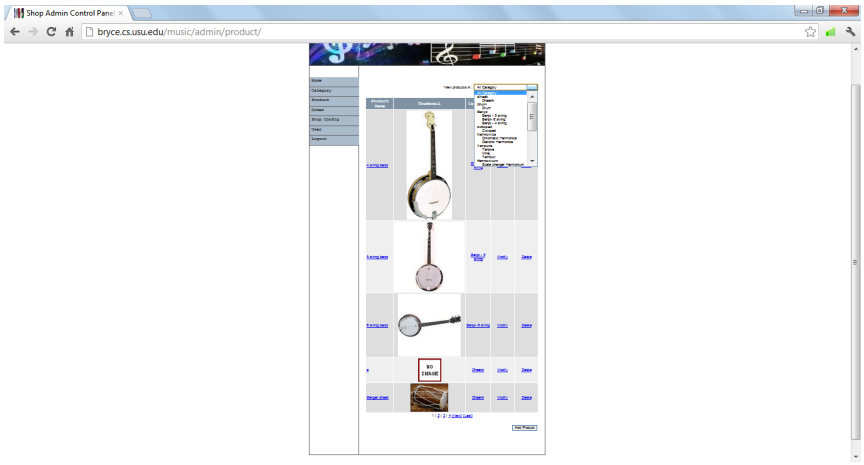

Fig. 6.3.5 Admin View/Delete Product Page of Music Store web application

Fig. 6.3.6 Admin Add Product Page of Music Store web application

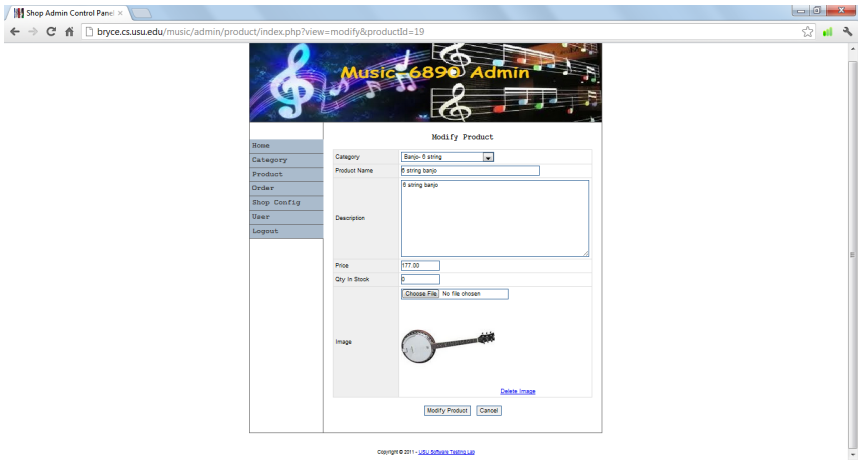

Fig. 6.3.7 Admin Modify Product Page of Music Store web application

- Order:
  - View Orders: Here we can see all the orders we have and their status. When you click the "Order" link on the left navigation you will go straight to the "Paid" orders. The reason is so you can respond immediately upon your customers that already paid for their purchase.
  - Modify Orders: Sometimes a customer might contact us saying that she made the wrong order like specifying the wrong product quantity or simply want her order cancelled so she can repeat the buying process again. This page enables the admin to do such a thing.

Fig. 6.3.8 Admin Order Management Page of Music Store web application

- Shop Configuration: This is where we can set and change our online shop appearance, behavior and information (shop name, main url, etc). Figure 6.3.9 shows the admin shop configuration screenshot.

Fig. 6.3.9 Admin Shop Configuration Page of Music Store web application

Common User Features

Our music store common user page consists of the following:

- Search category/product: Here customer can search for a specific category or product. Figure 6.4.1 shows a search result screenshot of this feature.

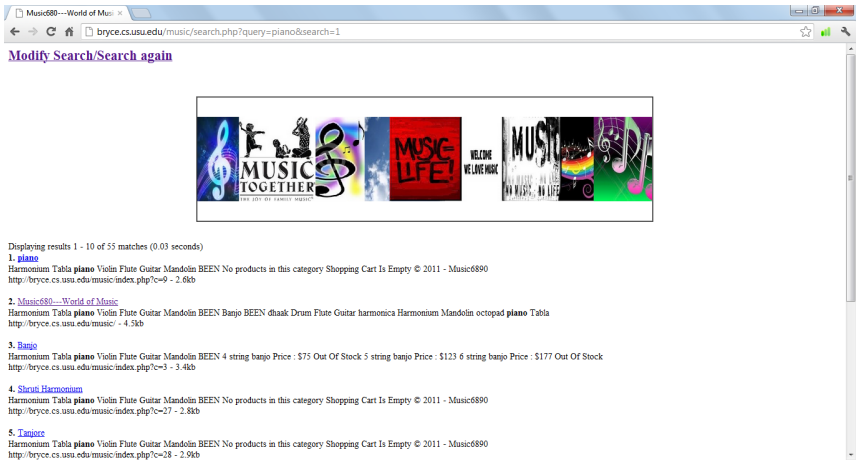

Fig. 6.4.1 Common user search result page of Music Store web application

- Browse category/product: Here customer can see a specific category or product. Figure 6.4.2 shows a corresponding screenshot of this feature.

Fig. 6.4.2 Common user view category/product page of Music Store web application

- Add to Cart: Here customer can add a specific product to his/her shopping cart. Figure 6.4.3 shows the corresponding screenshot.

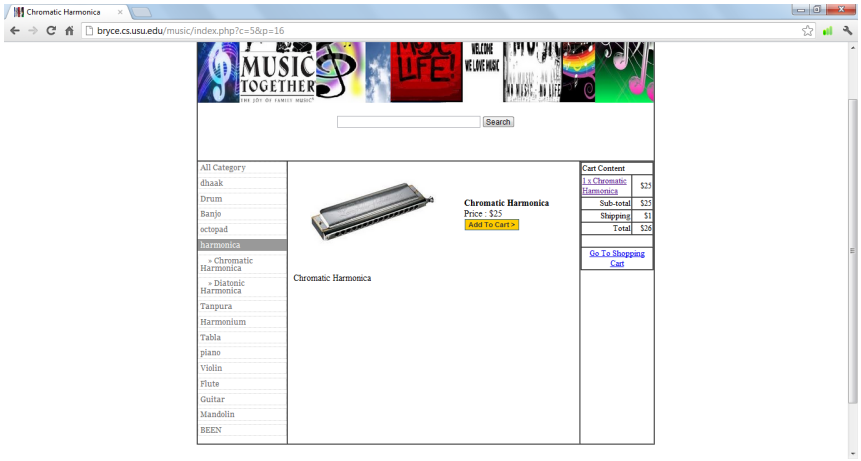

Fig. 6.4.3 Common user add to cart page of Music Store web application

- View/Update shopping Cart: Here customer can view/update his/her shopping cart before finalizing order or before. Figure 6.4.4 shows the corresponding screenshot.

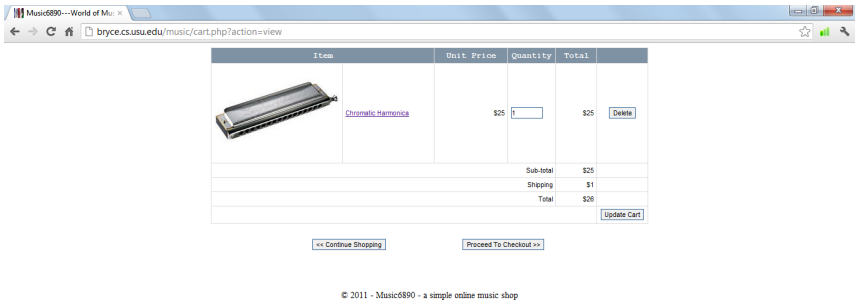

Fig. 6.4.4 Common user view/update cart page of Music Store web application

- Checkout Cart: Here customer provides shipping and payment information. Figure 6.4.5 and Figure 6.4.6 show the corresponding check out screenshots.

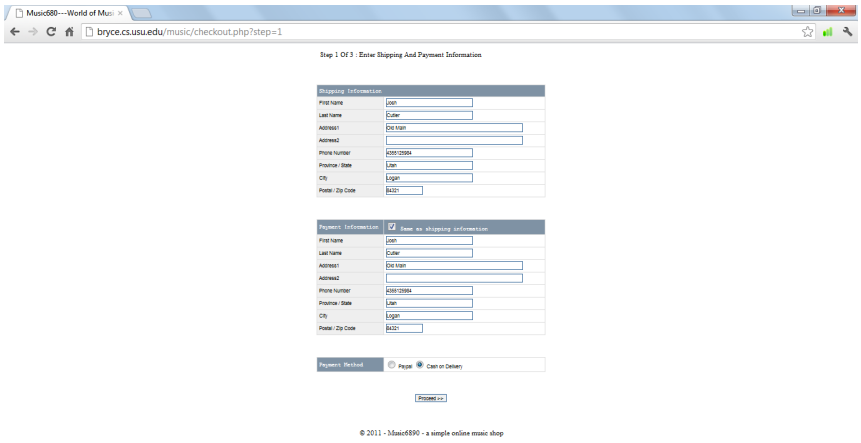

Fig. 6.4.5 Common user check out page 1 of Music Store web application

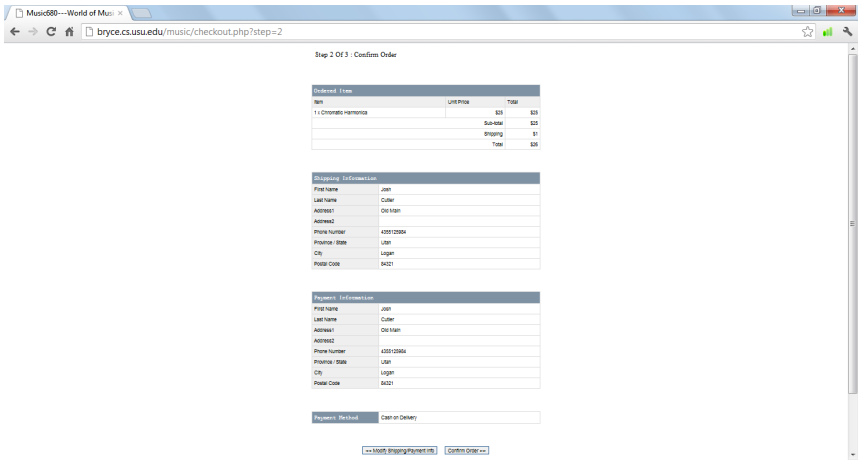

Fig. 6.4.6 Common user check out page 2 of Music Store web application

Table 6.1 summarizes the technical characteristics of the application, information about the test suite, and seeded faults.

Blank Lines	1812
Classes	0
Lines of Code	9940
Functions	207
Files	130
Executable statements	5152
No of branches	427
Declarative Statements	324
Comment to Code ratio	0.10
Comment lines	975
Lines	14146
Total no of test-cases	165
Total URL	3577
Largest no of GET/POST in a test-case	467
Average no of GET/POST in a test-case	21.68
Largest no of parameters in a test-case	808
Average no of parameters in a test-case	28.72
No of added faults	68
Largest 2-way score covered in a test suite	41666
Largest 3-way score covered in a test suite	32492

Table 6.2 Technical summary of Music Store Web Application and Test Suite

VII. CONCLUSION AND FUTURE WORK

Algorithms for combinatorial interaction testing provide systematic coverage of t-way interactions in a system. Our application of t-way combinatorial coverage for test suite prioritization of user-session-based testing differs in that the test suite already exists and may not contain all possible t-way interactions in a system, since test-cases are generated by users that visit a website. It is unlikely that users of many systems will exhaustively cover all t-way interactions during their visits, particularly when users have unique user ids, passwords, and personal information that they enter into a system. This raises the need for an algorithm that does not enumerate all possible t-tuples to track, and instead only stores the valid t-tuples in the test suite in order to save memory. Our experiments show that our approach scales well for a medium-sized web application and user base in which we capture test-cases for 12 days and then double the log. Further, our empirical study examines the application of 3-way inter-window parameter-value interaction coverage applied to the Music Store web application that was seeded with 68 faults. We collected test suites for each of the three user types for Music Store, prioritized the test suites, and compared the rate of fault detection with five prioritization criteria. Prioritization by 2-way and 3-way criteria were most effective, both performing within 1% of each other. However, 2-way prioritization provided a slightly better rate of fault detection. A closer look at the data revealed that the system contained more faults triggered by 2-way than by 3-way inter-window parameter-value interactions.

Future Work

Future work may examine a larger set of empirical studies with applications in which faults may potentially be triggered by higher strength interactions. Future work may also look at intra-window event interactions. Also, these higher order prioritization techniques can be applied to Rich Internet Applications (RIAs), specifically RIAs with many AJAX type requests to the server. Another area would be to have a slight variation on the way the calculation. For instance weights may be applied for preference to specific pages, parameters, or values. Also, other algorithmic techniques may be used to prioritize test suites.

REFERENCES

1) National Institute of Standards and Technology, The Economic Impacts of Inadequate Infrastructure for Software Testing, U.S. Department of Commerce, May 2002.

2) G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test-cases for regression testing," IEEE Trans. on Software Engineering, vol. 27, no. 10, pp. 929–948, Oct 2001.

3) R. C. Bryce, S. Sampath, and A. M. Memon, "Developing a Single Model and Test Prioritization Strategies for Event-Driven Software" IEEE Trans. on Software Engineering, vol. 37, pp. 48–64, Jan-Feb 2011.

4) Andrews, J. Offutt, and R. Alexander, "Testing web applications by modelling with FSMs," Software and Systems Modeling, vol. 4, no. 3, pp. 326–345, Jul 2005.

5) G. D. Lucca, A. Fasolino, F. Faralli, and U. D. Carlini, "Testing web applications," in the IEEE Intl. Conf. on Software Maintenance. Montreal, Canada: IEEE Computer Society, Oct. 2002, pp. 310–319.

6) F. Ricca and P. Tonella, "Analysis and testing of web applications," in the Intl. Conf. on Software engineering. Toronto, Ontario, Canada: IEEE Computer Society, May 2001, pp. 25–34.

7) W. Wang, S. Sampath, Y. Lei, and R. Kacker, "An interaction-based test sequence generation approach for testing web applications," in IEEE International Conference on High Assurance Systems Engineering. Nanjing, China: IEEE Computer Society, 2008, pp. 209–218.

8) W. Halfond and A. Orso, "Improving test-case generation for web applications using automated interface discovery," in ESEC / 15. SIGSOFT Foundations of Software Engineering. Dubrovnik, Croatia: ACM, Sep. 2007, pp. 145–154.

9) S. Artzi, A. Kiezun, J. Dolby, F. Tip, D. Dig, A. Paradkar, and M. D.Ernst, "Finding bugs in dynamic web applications," in ISSTA '08: Proceedings of the 2008 international symposium on Software testing and analysis. Seattle, WA, USA: ACM, Jul. 2008, pp. 261–272.

10) S. Elbaum, G. Rothermel, S. Karre, and M. F. II. Leveraging user session data to support web application testing. IEEE Trans. on Software Engineering, 31(3):187–202, May 2005.

11) S. Sampath, S. Sprenkle, E. Gibson, L. Pollock, and A. S. Greenwald. Applying concept analysis to user-session-based testing of web applications. IEEE Trans. on Software Engineering, 33(10):643–658, Oct. 2007.

12) S. Sprenkle, E. Gibson, S. Sampath, and L. Pollock. Automated replay and failure detection for web applications. In The Intl. Conf. of Automated Software Engineering, pp. 253–262, Nov. 2005.

13) S. Elbaum, G. Rothermel, S. Karre, and M. F. II. Leveraging user session data to support web application testing. IEEE Trans. on Software Engineering, 31(3), pp. 187–202, May 2005.

14) S. Elbaum, A. G. Malishevsky, and G. Rothermel. Test-case prioritization: A family of empirical studies. IEEE Trans. On Software Engineering, 28(2), pp. 159–182, Feb. 2002.

15) F. Ricca and P. Tonella. Analysis and testing of web applications. In the Intl. Conf. on Software Engineering, pp. 25–34, May 2001.

16) S. Pertet and P. Narsimhan. Causes of failures in web applications.Technical Report CMU-PDL-05-109, Carnegie Mellon University, 2005.

17) D. Jeffrey and N. Gupta. Test-case prioritization using relevant slices. In the Intl. Computer Software and Applications Conf., pp. 411–418, Sep. 2006.

18) Y. Guo and S. Sampath , “Web application fault classification - an exploratory study”, ESEM '08 Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement, pp. 303-305.

19) S. Sampath, R. C. Bryce, S. Jain and S. Manchester. A tool for combinatorial-based prioritization and reduction of user-session-based test suites. In International Conference on Software Maintenance: Tool Demo Track, pp. 574 -577, Sep. 2011.

20) S. Sprenkle, E. Gibson, S. Sampath, and L. Pollock. Automated replay and failure detection for web applications. In International Conference of Automated Software Engineering, pp. 253-262, Nov. 2005.

21) M. Benedikt, J. Freire, and P. Godefroid. VeriWeb: Automatically testing dynamic web sites. In the Eleventh International Conference on World Wide Web (May 2002).

22) D. C. Kung, C.-H. Liu, and P. Hsia, An object-oriented web test model for testing web applications. In the Asia-Pacific Conference on Quality Software, (Oct. 2000), IEEE Computer Society, pp. 111-120

23) C. Liu, D. Kung, P. Hsia, and C. Hsu. “Structural testing of web applications.” In International Symposium on Software Reliability Engineering , pp. 84-96, Oct. 2000

24) J. Ofutt, Y. Wu, X. Du, and H. Huang. Bypass testing of web applications. In International Symposium on Software Reliability and Engineering (Nov. 2004), IEEE Computer Society, pp. 187-197.

25) Z. Qian. Test-case generation and optimization for user session-based web application testing. Journal of Computers 5, 11, pp. 1655-1662, 2010.

26) D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The aetg system: An approach to testing based on combinatorial design. IEEE Transactions on Software Engineering 23, 7, pp. 437–444, Jul 1997.

27) HttpUnit. http://httpunit.sourceforge.net/, accessed on Dec. 19, 2011.

28) Rational Robot. http://www.ibm.com/software/awdtools/tester/robot/, accessed on Dec. 19, 2011.

29) S. Elbaum, S. Karre, and G. Rothermel. Improving web application testing with user session data. In International Conference on Software Engineering (Sep. 2003),pp. 49-59.

30) S. Sampath, R. Bryce, G. Viswanath, V. Kandimalla, and A. G. Koru. Prioritizing user-session-based test-cases for web application testing. In International Conference on Software Testing, Verification and Validation (Apr. 2008), pp. 141-150.

31) S. Sampath, V. Mihaylov, A. Souter, L. and Pollock. Composing a framework to automate testing of operational web-based software. In International Conference on Software Maintenance (Sep. 2004), IEEE Computer Society, pp. 104-113.

32) S. Sampath, S. Sprenkle, E. Gibson, L. and Pollock. Web application testing with customized test requirements|an experimental comparison study. In International Symposium on Software Reliability Engineering (Nov. 2006), pp. 266-278.

33) N. Alshahwan, and M. Harman. Automated session data repair for web application regression testing. In International Conference on Software Testing, Verification and Validation (Apr. 2008), pp. 298-307.

34) R. C. Bryce, and A. M. Memon. Test suite prioritization by interaction coverage. In Workshop on Domain-Specific Approaches to Software Test Automation (Sep. 2007), pp. 1-7.

APPENDICES

APPENDIX A

CPUT

Combinatorial-based Prioritization for User-Session-Based Testing (CPUT) is an open source software testing tool. There are 3 major functionalities of CPUT which can be listed as in the following…

i. New Logger module for Apache Web Server
ii. Conversion of web server usage logs into XML formatted test-cases
iii. Prioritization criteria

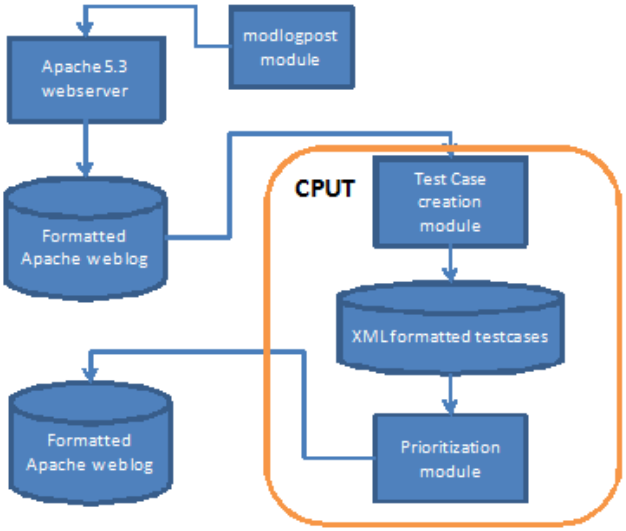

Fig. A.1 CPUT Tool overview

Figure A.1 gives an overview of CPUT functions. The logger for Apache is implemented as a module in C. The remaining components of CPUT - the test-case creation engine, the prioritization engine, and the user interface - are implemented in Java. A user of the tool first deploys the module in Apache to enable the logging of user sessions. The user then loads the usage logs into CPUT which parses the log to create XML-format test-cases. The XML test-cases can then be prioritized using a particular test prioritization method.

i. New Logger module for Apache Web Server

The logger for Apache was implemented in C as a module. The module was generically designed to deploy on Apache that is running on both Windows and Linux platforms. The module logs the HTTP GET and POST requests. The HTTP GET requests are typically logged by default in most web servers. HTTP POST requests generally transmit form data as part of the HTTP request body, instead of being appended to the URL. Therefore, additional methods were necessary to gather the date associated with an HTTP POST request. This module should be included with other Apache modules and can be

enabled by setting the Apache server's configuration file. Version 1.0 of this module logs the request data in the following format:

[Date]]# IP Address]# Method]# URL]# Cookie Id]# Referrer]# POSTDATA

ii. Conversion of web server usage logs into XML formatted test-cases

The test-case generation utilizes previously used heuristics to convert a usage log into test-cases. Specifically, the cookie information, the IP address, and the time stamp of each request are used to assign a request with a test-case. The usage log and the test-cases are stored in a PostGreSQL database. Figure A.2 shows the CPUT screen when a user specifies options to load the log _le into the database. Storing the logs and test-cases in a database allows for efficient storage and retrieval. The test-cases from the database table are then converted into test-cases in XML format. Figure A.3 shows the CPUT screen with all the XML test-cases parsed from the log _le. An XML format was chosen because of the extensible and easily parsing nature of XML. Figure A.4 shows a sample test-case. The important tags include: test suite denotes the test suite, session id represents the unique ID of a test-case within the test suite, and URL represents a page that the test-case accesses and has an associated request with a request type of GET or POST. Within a request, a baseURL includes the specific page that is accessed and parameters (denoted as param) that have names for parameters and values that are assigned to the parameter.

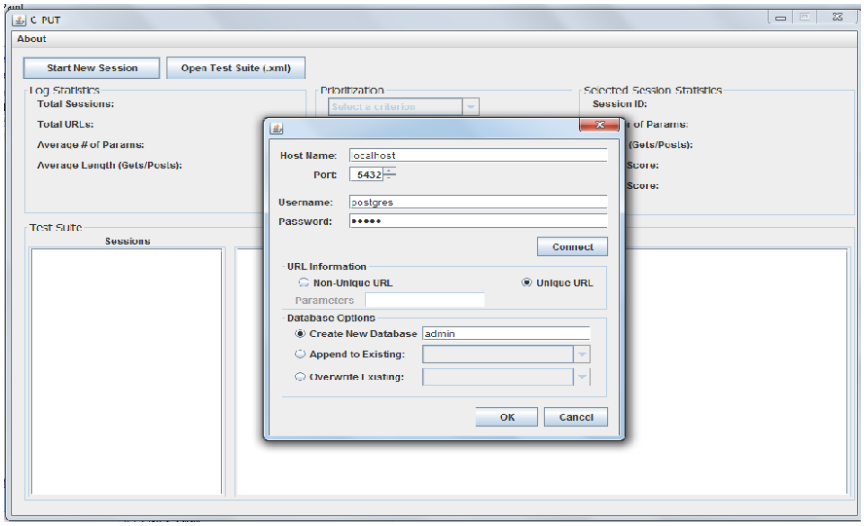

Fig. A.2 CPUT screen with options to load log file into database.

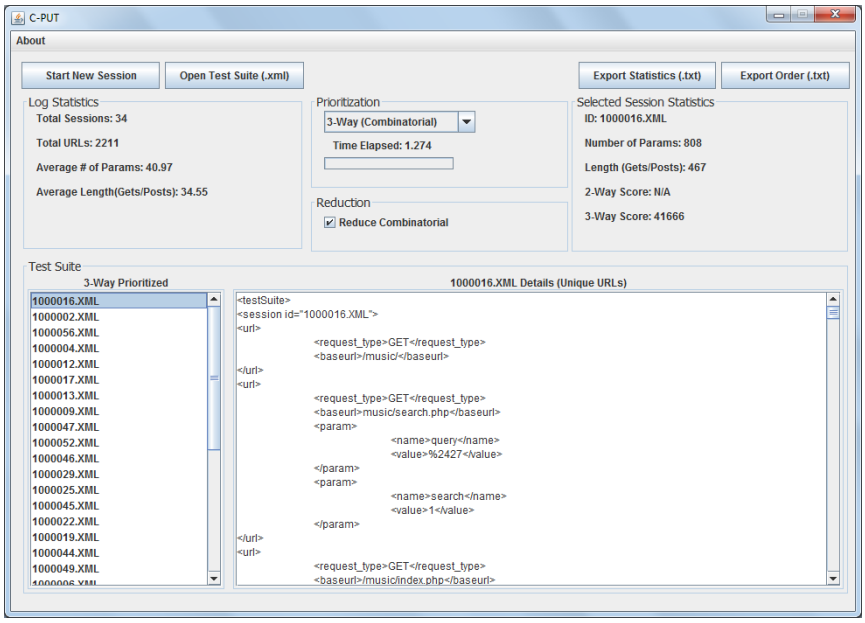

Fig. A.3 CPUT: Screenshot

iii. Prioritization criteria

This allows a user to prioritize their test suite. The options include:

- Length (Gets/Posts)
- Number of parameters
- 2-way combinatorial (disabled for test suite files larger than 15 MB)
- 3-way combinatorial (disabled for test suite files larger than 12 MB)
- Random
- Frequency-MFPS (Most frequently present sequence)
- Frequency-APS (All present sequence)
- Frequency-WF (Weighted Frequency)

APPENDIX B

TEST SUITES

The following figures show a test-case from the test suite used in the experiment described in this paper. The test-case has 23 GETS/POSTS, 21 parameters, a 2-way score of 9 and a 3-way score of 89.

```
<testSuite>
<session id="1000019.XML">
<url>
        <request_type>GET</request_type>
        <baseurl>/music/</baseurl>
</url>
<url>
        <request_type>GET</request_type>
        <baseurl>/music/library/common.js</baseurl>
</url>
<url>
        <request_type>GET</request_type>
        <baseurl>/music/include/shop.css</baseurl>
</url>
<url>
        <request_type>GET</request_type>
        <baseurl>music/search.php</baseurl>
        <param>
                <name>query</name>
                <value>mandolin</value>
        </param>
        <param>
                <name>search</name>
                <value>1</value>
        </param>
</url>
<url>
        <request_type>GET</request_type>
        <baseurl>/music/index.php</baseurl>
</url>
```

Fig. B.1 Sample test-case (part 1) from test suite described in Section 4.2

```
<url>
        <request_type>GET</request_type>
        <baseurl>music/search.php</baseurl>
        <param>
                <name>query</name>
                <value>mandolin</value>
        </param>
        <param>
                <name>search</name>
                <value>1</value>
        </param>
</url>
<url>
        <request_type>GET</request_type>
        <baseurl>music/index.php</baseurl>
        <param>
                <name>c</name>
                <value>28</value>
        </param>
</url>
<url>
        <request_type>GET</request_type>
        <baseurl>music/index.php</baseurl>
        <param>
                <name>c</name>
                <value>29</value>
        </param>
</url>
<url>
        <request_type>GET</request_type>
        <baseurl>music/index.php</baseurl>
        <param>
                <name>c</name>
                <value>30</value>
        </param>
</url>
```

Fig. B.2 Sample test-case (part 2) from test suite described in Section 4.2

```
<url>
	<request_type>GET</request_type>
	<baseurl>music/search.php</baseurl>
	<param>
		<name>query</name>
		<value>logan</value>
	</param>
	<param>
		<name>search</name>
		<value>1</value>
	</param>
</url>
<url>
	<request_type>GET</request_type>
	<baseurl>/music/index.php</baseurl>
</url>
<url>
	<request_type>GET</request_type>
	<baseurl>music/search.php</baseurl>
	<param>
		<name>query</name>
		<value>dhaak</value>
	</param>
	<param>
		<name>search</name>
		<value>1</value>
	</param>
</url>
<url>
	<request_type>GET</request_type>
	<baseurl>/music/index.php</baseurl>
</url>
<url>
	<request_type>GET</request_type>
	<baseurl>music/index.php</baseurl>
	<param>
		<name>c</name>
		<value>13</value>
	</param>
</url>
```

Fig. B.3 Sample test-case (part 3) from test suite described in Section 4.2

```
<url>
        <request_type>GET</request_type>
        <baseurl>music/index.php</baseurl>
        <param>
                <name>c</name>
                <value>13</value>
        </param>
        <param>
                <name>p</name>
                <value>5</value>
        </param>
</url>
<url>
        <request_type>GET</request_type>
        <baseurl>music/index.php</baseurl>
        <param>
                <name>c</name>
                <value>8</value>
        </param>
</url>
<url>
        <request_type>GET</request_type>
        <baseurl>music/index.php</baseurl>
        <param>
                <name>c</name>
                <value>33</value>
        </param>
</url>
<url>
        <request_type>GET</request_type>
        <baseurl>music/index.php</baseurl>
        <param>
                <name>c</name>
                <value>9</value>
        </param>
</url>
```

Fig. B.4 Sample test-case (part 4) from test suite described in Section 4.2

```
<url>
        <request_type>GET</request_type>
        <baseurl>music/index.php</baseurl>
        <param>
                <name>c</name>
                <value>10</value>
        </param>
</url>
<url>
        <request_type>GET</request_type>
        <baseurl>music/search.php</baseurl>
        <param>
                <name>search</name>
                <value>1</value>
        </param>
</url>
<url>
        <request_type>GET</request_type>
        <baseurl>/music/index.php</baseurl>
</url>
<url>
        <request_type>GET</request_type>
        <baseurl>music/search.php</baseurl>
        <param>
                <name>query</name>
                <value>kkk</value>
        </param>
        <param>
                <name>search</name>
                <value>1</value>
        </param>
</url>
<url>
        <request_type>GET</request_type>
        <baseurl>/music/index.php</baseurl>
</url>
</session>
</testSuite>
```

Fig. B.5 Sample test-case (part 5) from test suite described in Section 4.2

APPENDIX C

CODE COVERAGE

Following tables summarize the code coverage in terms of lines of code and percent covered for the entire test suite used for the experiment.

TestCase ID	%Code Covered	No. of Lines covered
1000000	1.18%	550
1000001	12.88%	604
1000002	17.70%	1671
1000003	13.25%	2339
1000004	5.83%	546
1000005	12.31%	1189
1000006	11.35%	775
1000007	39.02%	2091
1000008	13.70%	1658
1000009	55.29%	775
1000010	16.95%	2238
1000011	22.52%	1024
1000012	29.04%	1741
1000013	21.75%	1038
1000014	9.18%	550
1000015	10.08%	604
1000016	27.88%	1671
1000017	29.12%	2339
1000018	9.11%	546
1000019	19.84%	1189
1000020	12.93%	828
1000021	16.78%	740
1000022	37.32%	1192
1000023	34.88%	1093
1000024	27.66%	3588
1000025	12.93%	1758
1000026	37.34%	828
1000027	17.08%	740
1000028	29.05%	1192

Table C.1 Summary (Part 1) of Percent Code Coverage and Number of Lines Covered by Each Individual Test-case

TestCase ID	%Code Covered	No. of Lines covered
1000029	17.32%	1093
1000030	22.76%	3588
1000031	16.58%	1758
1000032	23.59%	2458
1000033	18.15%	1094
1000034	15.28%	820
1000035	22.86%	1473
1000036	11.90%	1397
1000037	26.56%	856
1000038	15.88%	1202
1000039	54.20%	794
1000040	39.66%	1758
1000041	16.95%	2458
1000042	32.52%	1094
1000043	21.04%	820
1000044	16.68%	1473
1000045	48.65%	1397
1000046	13.16%	856
1000047	31.61%	1202
1000048	26.04%	794
1000049	16.80%	1007
1000050	26.66%	828
1000051	50.57%	740
1000052	16.58%	1192
1000053	23.59%	1093
1000054	18.15%	3588
1000055	15.28%	1758
1000056	22.86%	2458
1000057	11.90%	1094
1000058	26.56%	820
1000059	16.28%	1473
1000060	26.56%	1397
1000061	12.48%	856
1000062	17.12%	1202
1000063	29.23%	794
1000064	18.17%	1089
1000065	25.34%	1519
1000066	13.16%	789
1000067	26.59%	1594
1000068	11.90%	713

Table C.2 Summary (Part 2) of Percent Code Coverage and Number of Lines Covered by Each Individual Test-case

TestCase ID	%Code Covered	No. of Lines covered
1000069	11.93%	715
1000070	16.03%	961
1000071	17.83%	1069
1000072	18.25%	1094
1000073	23.14%	1387
1000074	12.95%	776
1000075	16.68%	1000
1000076	48.65%	2916
1000077	13.16%	789
1000078	31.61%	1895
1000079	26.04%	1561
1000080	17.10%	1025
1000081	13.00%	779
1000082	12.23%	733
1000083	39.14%	2346
1000084	10.11%	606
1000085	44.23%	2651
1000086	12.38%	742
1000087	13.73%	823
1000088	15.67%	939
1000089	44.91%	2692
1000090	13.70%	821
1000091	14.51%	870
1000092	32.33%	1938
1000093	47.81%	2866
1000094	22.91%	1373
1000095	13.80%	827
1000096	13.98%	838
1000097	14.30%	857
1000098	14.46%	867
1000099	11.90%	713
1000100	12.50%	749
1000101	32.18%	1929
1000102	42.88%	2570
1000103	13.70%	821
1000104	13.25%	794

Table C.3 Summary (Part 3) of Percent Code Coverage and Number of Lines Covered by Each Individual Test-case

TestCase ID	%Code Covered	No. of Lines covered
1000105	25.83%	1548
1000106	12.31%	738
1000107	15.35%	920
1000108	39.02%	2339
1000109	13.70%	821
1000110	55.29%	3314
1000111	20.19%	1210
1000112	38.89%	2331
1000113	39.32%	2357
1000114	17.50%	1049
1000115	18.80%	1127
1000116	12.51%	750
1000117	20.72%	1242
1000118	28.16%	1688
1000119	17.12%	1026
1000120	31.00%	1858
1000121	21.96%	1316
1000122	13.70%	821
1000123	18.42%	1104
1000124	14.95%	1024
1000125	8.63%	1741
1000126	28.50%	1038
1000127	7.74%	550
1000128	9.14%	604
1000129	33.18%	1671
1000130	37.79%	2339
1000131	15.88%	546
1000132	54.20%	1189
1000133	39.66%	828
1000134	16.95%	2331
1000135	32.52%	2357
1000136	21.04%	1049
1000137	16.68%	1127
1000138	48.65%	750
1000139	13.16%	1242
1000140	31.61%	1688

Table C.4 Summary (Part 4) of Percent Code Coverage and Number of Lines Covered by Each Individual Test-case

TestCase ID	%Code Covered	No. of Lines covered
1000136	21.04%	1049
1000137	16.68%	1127
1000138	48.65%	750
1000139	13.16%	1242
1000140	31.61%	1688
1000141	26.04%	1026
1000142	17.10%	1858
1000143	13.00%	1316
1000144	12.23%	821
1000145	39.14%	1104
1000146	10.11%	1024
1000147	31.00%	1741
1000148	21.96%	1038
1000149	13.70%	550
1000150	18.42%	604
1000151	14.95%	1104
1000152	8.63%	1024
1000153	28.50%	1741
1000154	7.74%	1038
1000155	9.14%	550
1000156	33.18%	604
1000157	37.79%	1671
1000158	15.88%	2339
1000159	54.20%	546
1000160	12.23%	1189
1000161	39.14%	828
1000162	10.11%	2331
1000163	31.00%	2357
1000164	21.96%	1049

Table C.5 Summary (Part 5) of Percent Code Coverage and Number of Lines Covered by Each Individual Test-case

APPENDIX D

SEEDED FAULT SUMMARY

The 68 seeded faults used for the experiment described in this paper are broken up by fault category and user type. In addition, the logic fault category is broken down into five sub-categories.

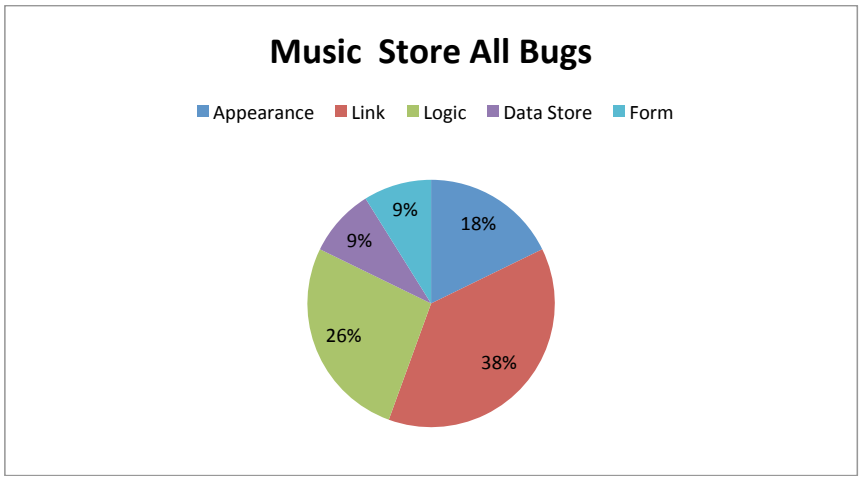

Fig. D.1 Fault category distribution for 68 seeded faults

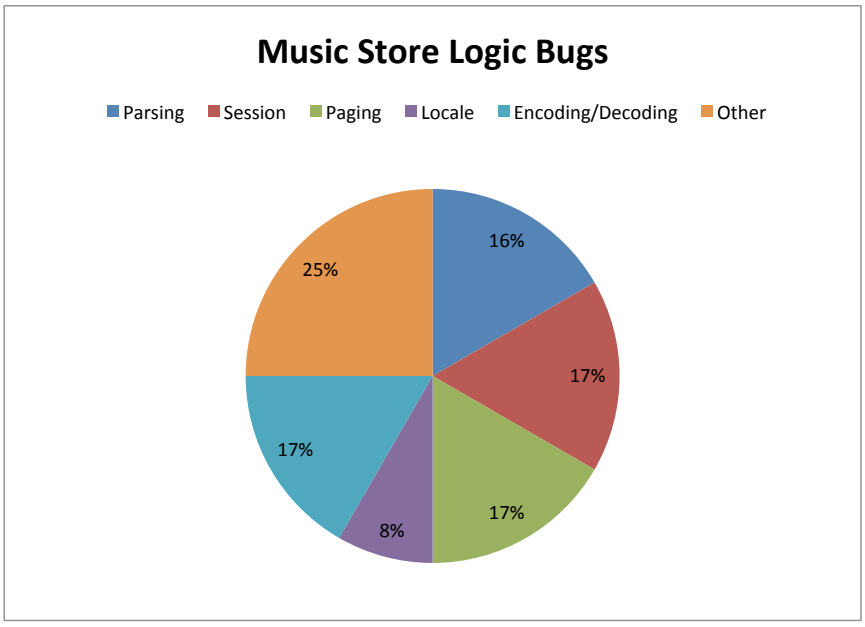

Fig. D.2 User type distribution for 68 seeded faults

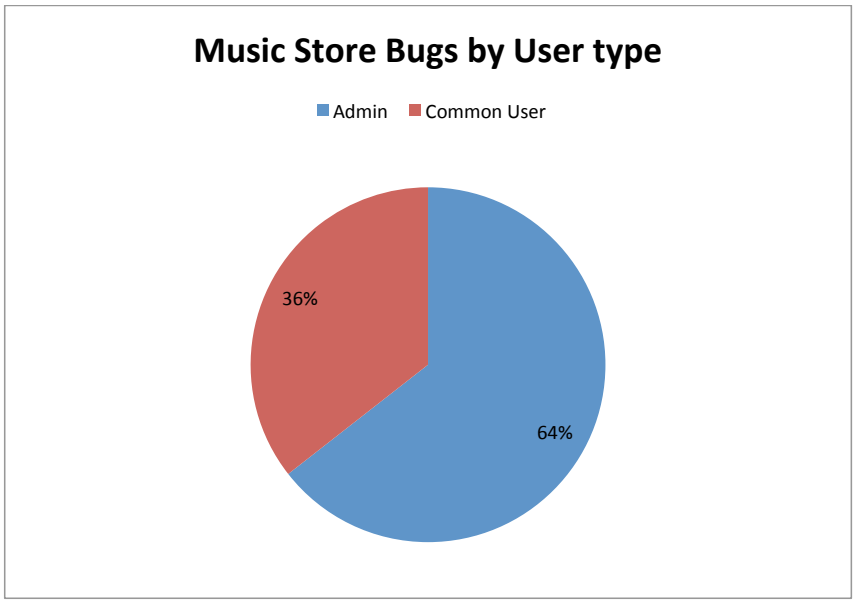

Fig. D.3 Logic fault sub-category distribution for 30 seeded logic faults

APPENDIX G

LIST OF FAULTS/BUGS

Music Store Fault Statistics

Category	Sub Category	Fault Description	Failure Description	User Type	Number of faults	Fault file details	**Music Store Versions**	**Comments**
Appearance Fault		Faults which affect the web page display.	Failure details which affect the web page display.	Admin and Common User	8			
	/	improper directory listing as image source	The picture of common user home page is not showing properly	User	1	Music.jpg	Music 1.1	Bug added
	/	improper directory listing as image source	The picture of admin home page is not showing properly	Admin	1		Music 1.2	Bug added
	/	Improper CSS for admin pages	Left navigation bar of admin home page is not showing properly	Admin	1		Music 1.3	Bug added
	/	Improper CSS for user pages	The navigation bars of the home page are not	User	1		Music 1.4	Bug added

			showing properly					
	/	HTML is not coded properly for search function	The search bar in home page is not showing properly. Search/ Submit button is missing	User	1	Top.php Line 40	Music 1.5	Bug added
	/	Top.php is included in every user pages	The search bar is coming each and every page	User	1	Car.php, Checkout.php, Success.php	Music 1.6	Bug added
	/	Improper directory listing for user logo	The logo in user home page is not showing	User	1	Top.php	Music 1.7	Bug solved
	/	Improper directory listing for admin logo	The logo in admin home page is not showing	Admin	1	Main.php	Music 1.8	Bug solved
Link Fault		Faults in the application code that changes the Page pointed to by an URL.	Failure details in the application code that changes the Page pointed to by an URL.	Admin and Common User	17			
	/	Wrong hyperlink set	The category link of admin	Admin	1	Template.php Line 31	Music 1.9	Bug added

			page is pointing to the general home page of music store.					
	/	Wrong hyperlink set	The category link of admin page is pointing to product page of admin home page of music store.	Admin	1	Template.php Line 31	Music 1.10	Bug added
	/	Wrong hyperlink set	The category link of admin page is pointing to order page of admin home page of music store.	Admin	1	Template.php Line 31	Music 1.11	Bug added
	/	Wrong Hyper link set	The product link of admin page is pointing to a wrong page.	Admin	1	Template.php Line 32	Music 1.12	Bug added
	/	Wrong Hyper link set	The order link of admin page is pointing to a wrong page.	Admin	1	Template.php Line 33	Music 1.13	Bug added
	/	Wrong Hyper link set	The shop config.	Admin	1	Template.php Line 34	Music 1.14	Bug added

			link of admin page is not working.					
	/	Wrong Hyper link set	The user link of admin page is not working.	Admin	1	Template.php Line 35	Music 1.15	Bug added
	/	Wrong Hyper link set	The logout link of admin page is not working.	Admin	1	Template.php Line 37	Music 1.16	Bug added
	/	Wrong Hyper link set	The modify search/ search link of user page is not working.	User	1	search_ header.html Line 9	Music 1.17	Bug added
	/	Wrong Hyper link set	Go to shopping cart link in user page is not working.	User	1	Minicart.php Line 45	Music 1.18	Bug added
	/	Wrong Hyper link set, onclick function is not working	Proceed to checkout link in user checkout page is not working	User	1	Cart.php Line 107	Music 1.19	Bug added
	/	Wrong Hyper link set	All category link in user home page is	User	1	leftNav.php Line 13	Music 1.20	Bug added

			not working					
	/	Wrong Hyper link set	Add to shopping cart link in user home page is not working.	User	1	productDetail.php Line 22	Music 1.21	Bug added
	/	Wrong Hyper link set	The product name/image link in user shopping cart page is not working.	User	1	productList.php Line 46	Music 1.22	Bug added
	/	Wrong Hyper link set, onclick function wrongly used	Continue shopping link in user shopping cart is not working.	User	1	Cart.php Line 103	Music 1.25	Bug added
	/	Wrong Hyper link set	The delete user link in admin home page is not working.	Admin	1	List.php Line 39	Music 1.29	Bug added
	/	Wrong Hyper link set	The change password link in admin home page is not working.	Admin	1	List.php Line 38	Music 1.30	Bug added
Logic Fault		Faults in the	Failure details	Admin and	12			

		applicat ion code that implem ent Busines s logic and control flow.	in the applicat ion code that implem ents Busines s logic and control flow.	Commo n User				
	Other	Onclick delete function wrongly coded	The delete button in user shoppin g cart is not working .	User	1	cart-function s.php Line 103-113	Music 1.23	Bug add ed
	Other	Onclick updatec art function wrongly coded	The update button in user shoppin g cart is not working .	User	1	cart-function s.php Line 131-160	Music 1.24	Bug solv ed
	Paging	Wrong paging logic	Displayi ng all the search results is not proper. Only first 10 records are being displaye d.	User	1	searchf uncs.ph p Line 600-612	Music 1.26	Bug add ed
	Encoding / decoding	Md5 is done while changin g passwor d but while login md5 hashing is not	Once the passwor d is change d user will not be able to login with the new passwor	Admin	1	change Pass.ph p Line 17	Music 1.27	Bug add ed

		done to check the password matches or not	d in admin page. Because while checking user in function.php the password is not encrypted.					
	Encoding / decoding		The add user link in admin home page is adding new user but the newly created user is not able to login.	Admin	1	process User.php Line 32-35	Music 1.28	Bug added
	Other	on click event is blank	The add user button link in admin home page is not working.	Admin	1	List.php Line 49	Music 1.31	Bug added
	Paging	Paging variable value is kept static, it is not dynamically populating	the show all order link in admin home page is showing only one order per page	Admin	1	Functions.php Line 281-301	Music 1.32	Bug added
	Locale	Date format is not used	The detail order link in	Admin	1	Detail.php Line 58-65	Music 1.36	Bug solved

		while displaying dates	admin home page is not showing date in proper format.					
	Parsing	PHP variables and codes are wrongly enclosed in HTML .	The email link in admin shop config. page is not properly HTML parsed.	Admin	1	Main.php Line 59	Music 1.35	Bug added
	Parsing	PHP variables and codes are wrongly enclosed in HTML .	The search button in the user home page is not showing "search " rather it is showing the PHP variable .	User	1	Top.php	Music 1.39	Bug added
	Session	login session is not being store	Admin pages are not being properly navigated for admin login.	Admin	1	Functions.php Line 56-72	Music 1.40	Bug added
Form Faults		Faults in the application code that controls , Modifies and displays	Failure details in the application code that controls , Modifies and	Admin	4			

		name-value pairs in forms.	displays name-value pairs in forms.					
	1	Function Onchange is not correctly coded	The view orders option in admin home page is not working properly.	Admin	1	List.php Line 46	Music 1.33	Bug added
	2	Dropdown list is wrongly populated by for loop	The currency options cannot be selected and showing wrongly in admin shop config page.	Admin	1	main.php Line 29-45	Music 1.34	Bug added
	3	Address and name variables are swapped while storing corresponding values	The address and shop name values are swapped in admin shop config page.	Admin	1	process Config.php Line 34-37 Main.php Line 47-51	Music 1.37	Bug added
	4	Telephone and email variables are swapped while storing corresponding values	The telephone and email in admin shop config page are swapped.	Admin	1	Main.php Line 55-59 process Config.php Line 34-37	Music 1.38	Bug added

Data Store Fault		Faults in the application code that Manipulates data in any kind of data store.	Failure details in the application code that Manipulates data in any kind of data store.	Admin	4			
	1	Insert query is wrong.	Incorrect SQL statement for add category in admin category page.	Admin	1	process Category.php Line 46-48	Music 1.42	Bug added
	2	Price and quantity variables swapped their places in insert query	For the add product link the price and quantity is swapped while being stored.	Admin	1	process Product.php Line 47-49	Music 1.43	Bug added
	3	Insert query is missing password value from user	Change password is setting the user password as blank.	Admin	1	process User.php Line 52-55	Music 1.44	Bug added
3-way fault		Fault details in application code that affects 3-way interaction of the web-	Failure details in application code that affects 3-way interaction of the web-	Admin and common user	8			

		pages	pages					
	1	Variable value for currency is randomly set.	In admin page currency was selected as USD, but in user page currency is not coming as USD	User	1	processConfig.php Line 34-62	Music 1.46	Bug added
	2	Continue shopping function has been changed accordingly so that it can delete all the items in the cart in a particular user session.	After add product going to cart page shows an option to continue shopping – pressing that button deleting all the existing items in the shopping cart	User	1	Cart-functions.php Line 116 Cart.php Line 103	Music 1.47	Bug added
	3	In Cart.php step value is changed to directly 2 to skip the page for shipment information. Corresponding changes	Proceed to checkout is directly going to checkout confirmation page – but modify shipment/payment informa	User	1	CheckoutConfirmation.php Line 9-24, 200 Cart.php Line 107	Music 1.48	Bug added

		has been done in Checkoutconfirmation.php	tion from that page is not happening					
	4	Variable value storing payment method changed from cashondelivery to paypal and accordingly corresponding functions also changed	Selecting PayPal as payment method is taking cash-on-delivery as payment method	User	1	checkoutConfirmation.php Line 19-39	Music 1.49	Bug added
	5	Mail sending check removed and corresponding changes has been done in process Config.php	In admin page email notification was deactivated in admin shop-config, then also email notification was sent when an order has been issued by an user	Admin	1	success.php Line 15-21 process Config.php	Music 1.50	Bug added
	6	Variable that contain	In admin page	Admin	1	Minicart.php Line 43-	Music 1.45	Bug added

		s the value of shipping cost in the user pages has been changed randomly	shipment cost is there but in user page shipment cost is not coming same.			45 Cart.php Line 67-69 checkConfirmation.php Line 92-94		
	7	Created an extra column for enable/ disable and for each newly added category it is initially disabled for users to see.	In admin page category is added but in user page the same category is not being showed	Admin	1	category-functions.php Line 75-78, 128-131 process Category.php Line 46-48, 110-112	Music 1.51	Bug added
	8	Created an extra column for enable/ disable and for each newly added product it is initially disabled for users to see.	In admin page product is added but in user page the same product is not being showed	Admin	1	productList.php Line 13-16 product-functions.php Line 18-20 process Product.php Line 48-49, 138-141	Music 1.52	Bug added
2-way fault		Fault in application code that affects 2-way interaction of the	Failures in application code that affects 2-way interaction of	Admin	17			

		web-pages	the web-pages					
	1	Change the value of variable product Id to 19	Product modify for a particular product is modifying different product	Admin	1	process Product .php in admin/ product . Line 137-138	Music 1.53	Bug added
	2	Delete random product if the product Id is 10	Product deletion for a particular product is deleting different product	Admin	1	process Product .php in admin/ product . Line 157-163	Music 1.54	Bug added
	3	catId value changed to 17 from 16.	Add product under a specific category is adding the product under different category	Admin	1	process Product .php in admin/ product . Line 47-48	Music 1.55	Bug added
	4	No call for image upload function in addproduct function	Image upload during add product is not working	Admin	1	process Product .php in admin/ product . Line 43-46	Music 1.56	Bug added
	5	No call for image upload function in modify	Image upload during modify product is not working	Admin	1	process Product .php in admin/ product . Line	Music 1.57	Bug added

		product function				118-121		
	6	Change the value of variable catId to 21	Modify a specific category is always modifying a different category	Admin	1	process Category.php in admin/category. Line 110-113	Music 1.58	Bug added
	7	Delete random category cat_Id is 7	Deleting a specific category is always deleting another category	Admin	1	process Category.php in admin/category. Line 128-133	Music 1.59	Bug added
	8	No call for image upload function in addcategory function	Image upload for add new category is not working – no image can be uploaded during new category addition	Admin	1	process Category.php in admin/category. Line 44	Music 1.60	Bug added
	9	No call for image upload function in modifycategory function	Image upload for modifying a specific old category is not working – no image can be uploaded for modific	Admin	1	process Category.php in admin/category. Line 98	Music 1.61	Bug added

			ation					
	10	Insert query is not executed	Add user is not adding new user in admin home page under user tab	Admin	1	processUser.php in admin/user. Line 56	Music 1.62	Bug added
	11	Delete query is not executed	Delete user is not deleting existing user	Admin	1	processUser.php in admin/user. Line 92	Music 1.63	Bug added
	12	Hyperlink is removed	Under order tab in admin home page selecting orders is not showing the details of that order	Admin	1	List.php in admin/user Line 79-80	Music 1.64	Bug added
	13	Update query is not executed	Under order tab after selecting a specific order if admin tries to change the order status the order status is not being changed	Admin	1	procesOrder.php in admin/user Line 35	Music 1.65	Bug added
	14	Wrong hyperlink and	Change password link in	Admin	1	User.js in admin/li	Music 1.66	Bug added

		modified delete function without confirmation	admin page is deleting the user			brary Line 27-30 List.php in admin/ user Line 38		
	15	Wrong hyperlink is set	Delete user is taking admin to change password page and not deleting the user	Admin	1	List.php in admin/ user Line 39	Music 1.67	Bug added
	16	Wrong function called in the hyperlink	Delete product in admin is going to modify product.	Admin	1	List.php in admin/ product Line 81	Music 1.68	Bug added
	17	Wrong function called in the hyperlink	Delete category in admin is going to modify category.	Admin	1	List.php in admin/category Line 66	Music 1.69	Bug added