

# Comparison of Models for Testing Ruby on Rails Web Applications

Darra Ricks, Sarah Vessels  
University of Kentucky  
Fall 2009

## Abstract

*Web applications are prevalent and it is important that they be of high quality as businesses, schools, and public services rely upon them. Toward that end, we compared two approaches for testing web applications. The Atomic Section Model (ASM) and the Qian, Miao, Zeng (QMZ) model are both used as a means of generating test cases that traverse a web application. We applied the two models to a Ruby on Rails web application to compare defect detection efficacy. We found that both models performed equally well in terms of total faults detected, though neither model found all seeded and naturally occurring faults. Also, the ASM model detected one fault that the QMZ did not, and vice versa.*

## 1 Introduction

Web applications are subject to problems that do not affect desktop applications. In a web application, for example, a user can enter a URL previously seen while using the application and perhaps access content that should no longer be accessible. This sort of scenario is not possible in a desktop application wherein a user cannot directly change the state of the application except by going through features provided as part of the application. Since web applications are accessed through a third-party browser not of the web application developer's making, a developer of a web application cannot control all of a user's actions while using the web application. Though it may be possible to change the state of a desktop application by modifying the underlying operating system settings (e.g., the Windows registry), the fact that a web application is accessed entirely through another application adds an extra layer that can be misused or changed by a user. Because of the uniqueness of how web applications are used, testing them is more complicated.

In order to focus on issues unique to web applications, we apply two models for test generation: Atomic Section Model (ASM) [1] and the Qian, Miao, Zeng model (QMZ) [2]. We designed a study to compare their defect detection effectiveness when applied to a single Ruby on

Rails web application. We seeded faults across several fault categories and naturally occurring faults existed as well. The paper is organized as follows: Section 2 discusses other web application testing work. Section 3 presents the web application that we tested, describes the faults, and discusses the two test models. Section 4 describes the experimental design and threats to validity. Section 5 is an analysis and discussion of results, and Section 6 contains our conclusions and future work.

## 2 Related Work

Filippo Ricca and Paolo Tonella use a UML model to represent web applications at a high level [3]. Their testing is concerned in part with finding unreachable pages and ghost pages [3]; that is, pages that the user cannot access through other pages on the site and non-existent pages, respectively. Their testing technique seems much like the QMZ model wherein every page (a.k.a., screen) is visited at least once and every link on every page is accessed. Generation of test cases is done in the same way as ASM and QMZ in that test paths through the web application are followed, using varying input to elicit different responses.

Edward Heatt and Robert Mee describe the "classic Web application testing problem" [4] as being the difficulty in testing the client-side and the server-side halves of a web application at once. They suggest testing the server code first by testing functionality that can be accessed directly without use of a web browser. They then suggest testing the client side code, such as Javascript. As for testing the interaction of client and server, they suggest writing tests "that simulate the request/response Web environment" [4].

Sebastian Elbaum, Gregg Rothermel, Srikanth Karre, and Marc Fisher II proposed a new technique for testing web applications that uses a user's session data [5]. The results of their empirical study showed that making use of session data leads to fault detection not seen in other testing methods; that is, there was not much overlap of detected faults across testing techniques.

Because of this, they suggest that their technique “may be complementary to that of more formal white box testing approaches” [5].

### 3 Web Application, Faults, and Testing Models

This section describes the web application to be tested, the faults that we seeded, and the naturally occurring faults we discovered before applying tests created based on the models.

#### 3.1 Web Application Details

The web application to be tested was written by three undergraduate students using the Ruby on Rails framework<sup>1</sup>. It was written as part of a senior design class where the objective was to implement the card game Set<sup>2</sup>. Much of the game functionality was implemented using AJAX calls, though these were ignored in testing the web application due to the models’ limitations. It would be helpful for future work to find a model for web applications that supports Javascript and AJAX testing.

The web application requires user authentication before users can access a list of current games, their own game play statistics, and a settings page that also doubles as a user management page for administrators. There are three pages accessible to non-authenticated users: rules, login, and register.

One of the features available with a Ruby on Rails application is access to its rake stats command, which generated the statistics below in Tables 1 and 2.

Table 1, Rake Stats about Web Application

Name	Lines of Code	LOC per Method
Controllers	629	20
Helpers	48	22
Models	269	13
Libraries	276	2
Integration tests	44	0
Functional tests	234	0
Unit tests	123	0

Table 2, Rake Stats (continued)

Code LOC	Test LOC	Code to Test Ratio
1222	401	1:0.3

<sup>1</sup> <http://rubyonrails.org/>

<sup>2</sup> <http://www.setgame.com/>

As shown in Tables 1 and 2 above, the web application was not devoid of tests. It had unit tests that could be run via the Ruby on Rails test framework. These tests focused on individual method correctness, however, and not overall system behavior, which is targeted by the ASM and QMZ tests that we generated and applied.

#### 3.2 Faults

There were 15 faults seeded in the web application, and some naturally occurring faults were found in the process. One of the original developers of the web application seeded the faults, ensuring someone familiar with the system and the Ruby on Rails framework seeded them. Having an original developer seed the faults helped also in that she was familiar with parts that were problematic during the application’s development.

##### 3.2.1 Seeded Faults

Table 3 below contains a numeric identifier for each seeded fault as well as its description. Table 4 associates each seeded fault, based on its numeric identifier, with the fault type/category that seemed most applicable. The developer who seeded the faults was responsible for fault categorization since that person was most familiar with what each fault entailed.

Table 3, Seeded Fault Descriptions

#	Description
1	Simple divide-by-zero error possible when displaying Your Stats section.
2	Redirect a user to the registration page instead of the login page when they attempt to access restricted content and he/she is not logged in.
3	No longer marking check_id and check_post as hidden in LoginController, meaning a user can manually edit the URL and access these as pages, but they are supposed to be internal. This is specific to the Rails framework.
4	Do not update the user's settings immediately upon changing them, meaning a user will have to log out and back in to see the effect.
5	Do not check to see that a user's account has been approved before redirecting him/her to the game area upon logging in.
6	When a user logs out, his/her session will not be reset. He/she will be able to access privileged content.
7	Swapped column headers in list of Current Games such that two columns are not correctly identified.

8	'Un-make admin' link displays for administrators on admin page even if the user to whom the link refers is not currently an admin (and thus it makes no sense to try and un-admin him/her).
9	On the settings/admin page, changing the 'confirm password' input element's name so that the back-end will complain about a missing password when the form is submitted.
10	Show a regular text field in place of a password field in the user registration form.
11	List of Current Games on game/index should only include unfinished, playable games for the user, but instead will show all games associated with the user, even those that are complete and thus unplayable.
12	Page title will not differ on settings/admin page for admins versus regular users—always shows Administration.
13	Logged-in users will no longer be redirected to the game page if they manually go to the login page.
14	Modified email validation when a user registers such that it is more naïve: it will allow valid emails such as <a href="mailto:user@example.com">user@example.com</a> but also invalid emails such as <a href="mailto:user@example...com">user@example...com</a>
15	Non-logged in users can attempt to access the settings page but it will give an application error instead of just redirecting them.

Table 4, Seeded Fault Categorization

#	Type
1	"Checking (faulty or missing validation of data and values in the source code)" [6]
2	"Interface (addresses errors in communication between users, modules or device drivers)" [6]
3	"des.logic – Unanticipated case (faulty design logic)" [6]
4	"Assignment (addresses faults in the source code such as faulty initialization)" [6]
5	Checking
6	"Function (missing or wrong functionality, may require a formal design change)" [6]
7	Interface
8	Checking
9	Checking
10	"init – Wrong initialization; wrong type; definition clash" [6]—Specifically, use of the wrong method.
11	Function
12	Interface

13	Function
14	Checking
15	Interface

### 3.2.2 Summary of Seeded Faults

Table 5 below summarizes the seeded faults, showing all represented categories and the percentage of faults in each category.

Table 5, Seeded Fault Summary

Fault Category	#	%
Assignment	1	6.67%
Checking	5	33.33%
Function	3	20.00%
Interface	4	26.67%
Unanticipated case	1	6.67%
Wrong type	1	6.67%
<b>Grand Total</b>	<b>15</b>	<b>100.00%</b>

### 3.3 Naturally Occurring Faults

Table 6 below contains the description and fault type/category for each of the naturally occurring faults we found in the web application. As with seeded faults, a numeric identifier represents each naturally occurring fault.

Table 6, Naturally Occurring Fault Descriptions and Categorization

#	Description	Type
1	User can change game's status from "finished" to "in progress" by visiting the game play page.	Function
2	User only sees games associated with his/her user ID, instead of games by all users.	Function
3	Main navigation link to the Login page actually links to Games index.	Function
4	Games with a negative number of players can be created.	Checking
5	"We are Sorry but something went wrong error" screen when blank game name is given when creating a game.	Checking
6	When creating a game with 0 players, message such as "Number of players must be greater than zero" should be displayed instead of "That game is already full."	Interface
7	When unapproving users as an admin and no users are selected, message should be "No Users Selected to unapprove" instead of "No Users Selected to approve."	Interface

### 3.3.1 Summary of Naturally Occurring Faults

Table 7 below contains the same fault categories as the seeded faults, since no naturally occurring fault was detected outside of those categories, and the percentage of naturally occurring faults that were detected in each category.

Table 7, Naturally Occurring Fault Summary

Fault Category	#	%
Assignment	0	0.00%
Checking	2	28.57%
Function	3	42.86%
Interface	2	28.57%
Unanticipated case	0	0.00%
Wrong type	0	0.00%
<b>Grand Total</b>	<b>7</b>	<b>100.00%</b>

It supports the validity of our experiment that 26.67% of our seeded faults were Interface faults and 28.57% of the naturally occurring faults were Interface faults. Likewise, 33.33% of seeded faults were Checking while 28.57% of naturally occurring faults were Checking faults; this also helps support the validity of our experiment.

## 3.4 Atomic Section Model

The Atomic Section Model by Jeff Offutt and Ye Wu represents all possible user interface screens of a web application, just as control flow graphs represent all possible sequences of software program execution. ASM can be divided into two levels: the Component Interaction Model (CIM) and the Application Transition Graph (ATG) [1]. CIM is designed for the individual components of the web application while ATG is for the web application as a whole, in which the individual components come together and function as a system. Test cases were formed based on the ATG (see section 3.4.2 below).

Applying the ASM to the web application first meant determining atomic sections [1]. An atomic section is described as data that has the property that if any part of the data is displayed, all of it is displayed. The data cannot be broken up further. The simplest example of an atomic section is a static HTML page. Most dynamic web pages have several atomic sections, e.g., page header, content, and page footer.

One difficulty in determining atomic sections is that the ASM does not handle AJAX calls, but the web application to be analyzed uses AJAX in some screens. We considered only non-

AJAX sections for both models because of this limitation. Based on the atomic sections, an Application Transition Graph was created.

Test paths were designed to have complete atomic section coverage [1], which means every atomic section in every page was covered at least once by a test path. Input values for the various forms in the web application were created, including login names and passwords. These tests were run manually and the results recorded about the faults encountered. Because of the manual test runs, we were able to notice faults that would possibly be hard to detect with automated testing, such as mislabeled columns in a table.

### 3.4.1 Component Expressions

Component expressions [1] represent the exact structure of a page through its atomic sections. Atomic sections are denoted as  $P_x$  where  $x$  is either letters or a number.  $PA \bullet PB$  means atomic section PA is followed directly by PB.  $PA^*$  means PA is part of a loop and is repeated an unknown number of times.  $PA | PB$  means either PA is shown or PB is shown, but not both. Grouping is done with parentheses. Though it was not indicated in Offutt and Wu's paper [1], we made a decision about how to show NULL atomic sections. When an atomic section is either shown or not, with no alternative atomic section in its place, we used NULL in the OR condition. For example, consider  $PA | NULL$ , meaning PA might be shown, but if it is not, nothing is shown in its place.

All pages in the web application share a common layout. To indicate shared atomic sections that are part of the common layout, a letter is used as the indicator, e.g. PA instead of P1. Atomic sections PA through PJ, along with PHEAD and PFOOT, are shared on all pages, and their arrangement is the same. Numeric atomic sections are specific to each page.

As an example, here is the component expression for the games screen:

$$\text{PHEAD} \bullet (\text{PA} | \text{PB}) \bullet \text{PC} \bullet ((\text{PD} \bullet (\text{PE} | \text{PF})) | \text{PG}) \bullet (\text{PH} | \text{NULL}) \bullet (\text{PI} | \text{NULL}) \bullet (\text{PJ} | \text{NULL}) \bullet \text{P1} \bullet (\text{P2} | \text{NULL}) \bullet \text{P3} \bullet (\text{P4} | \text{NULL}) \bullet \text{P5} \bullet (\text{P6} | \text{NULL}) \bullet \text{P7} \bullet ((\text{P8} \bullet (\text{P9} \bullet (\text{P10} | \text{P11}^*)) \bullet \text{P12}) | \text{NULL}) \bullet \text{PFOOT}$$

### 3.4.2 Application Transition Graph

The ATG represents how all the separate components of the web application fit together. An arrow pointing from one component  $x$  to

another component  $y$  means that within an atomic section in  $x$ , there exists a link to  $y$ . An arrow from  $x$  to  $x$  means that there is a link in  $x$  that will simply load the same component  $x$  again. For example, all screens are potentially accessible from all other screens because site navigation is included in the application-wide layout. While all atomic sections are not typically shown when viewing a given screen, it is possible for a given atomic section to be accessible, e.g., in the event of a fault that makes a normally inaccessible atomic section accessible.

The ATG below represents major components (i.e., screens) and methods in the web application that are accessible as either links or the action of an HTML form, e.g., login/try\_register is the action of a form on the Register Screen.

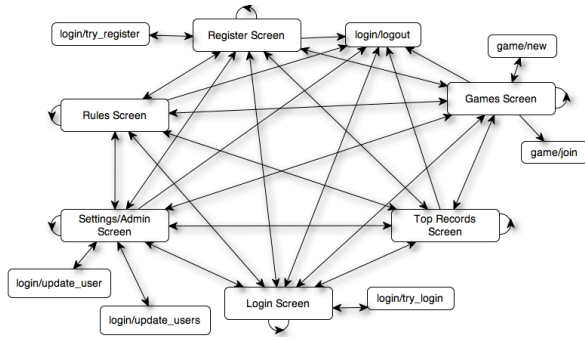


Figure 1, Application Transition Graph

### 3.4.3 CIM and Transition Graphs

The CIM of a particular screen includes the application’s start page, the screen’s atomic sections, the screen’s component expression, and the screen’s transitions (i.e., which other pages or URLs within the web application be accessed via that component). The start page for the entire web application was the login page, not shown here. The transition graph for the games screen is shown below in Figure 2. It includes the atomic sections and possible transitions that can be made from the games screen. Its component expression is shown above, in section 3.4.1. The games screen is where a user should be redirected upon logging in. It shows individual game play statistics, a Create Game form, and a list of games.

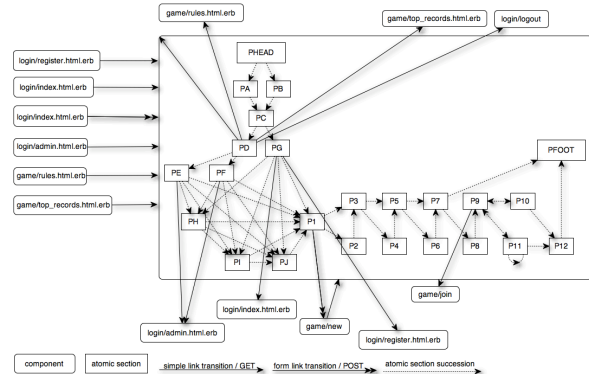


Figure 2, Games screen transition graph

## 3.5 Qian, Miao, Zeng Model

The Qian, Miao, Zeng (QMZ) model, like the Atomic Section Model, aims to model web applications for the purpose of testing them. Applying it involved the creation of a Page Flow Diagram (PFD) [2], which represents “the relationship among pages of a web application” [2]. A Page Test Tree (PTT) [2], based upon the PFD, is also necessary for generating shorter paths than those shown in the PFD. A Page Test Tree is a tree-like structure that is a simpler representation than that provided by the PFD of pages in the web application. Using the PFD2PTT algorithm [2], test paths were constructed for the web application.

Applying this model can involve dividing a sizable web application into smaller sub-web applications (subWAs) [2] because it is acknowledged that both the PFD and the PTT can become unwieldy for larger web applications. However, due to the small size of the web application being tested, it was not necessary to divide it into subWAs. See details about the web application below.

### 3.5.1 Application Flow Components

Table 8 below contains some sample Application Flow Components. The component name is a description of the flow component, the path name representation is an abbreviation for referring to the flow component in test paths, and the K Flow Number is a unique identifier for representing the flow component in the PTT.

Table 8, Application Flow Components

Component Name	Path Name Representation	K Flow Number
Registration Tab	Regst_tab	K2
Registration Button (Login Screen)	Regst_but	K3

Login Button	LI_but	K4
Login Tab	LI_tab	K5
Registration Button (Registration Screen)	Regst_but	K6

### 3.5.2 Test Paths Generated from PTT

The following are sample test paths generated from the PTT. In the path specification below, some path name representations (see Table 8 above) have an extension that specifies from which screen the particular object was clicked:

- \_RES – from Register screen
- \_LS – from Login screen

#### Register Screen

1. Start ⇒ Regst\_tab\_LS ⇒ Regst\_tab\_RES
2. Start ⇒ Regst\_tab\_LS ⇒ Regst\_but\_RES
3. Start ⇒ Regst\_tab\_LS ⇒ LI\_tab\_RES
4. Start ⇒ Regst\_but\_LS ⇒ Regst\_tab\_RES
5. Start ⇒ Regst\_but\_LS ⇒ Regst\_but\_RES

#### Login Refresh

1. Start ⇒ LI\_tab\_LS
2. Start ⇒ LI\_but\_LS

### 3.5.3 Sample Test Case

The following is a sample test case generated based on the QMZ model:

#### Start ⇒ Regst tab LS ⇒ Regst but RES

##### Registration 2\_1

1. Click Register tab.
2. Enter valid email, password, retype password.
3. Click Register button.
4. Verify application navigates back to Login page with message “Account request has been sent to Administrator.”

##### Registration N2

1. Click Register tab.
2. Do not enter any data in text fields.
3. Click Register button.
4. Verify error messages for blank text fields.

## 4 Experimental Design

These are the details of how the experiment was implemented.

## 4.1 Variable Selection

This is the description of which variables we selected for measurement and which variable we controlled.

### 4.1.1 Dependent Variables

The dependent variables will include the number of faults found, which faults are found (perhaps the ASM will find different faults than QMZ), and the types of faults found, e.g., interface, checking [6]. The goal is to measure defect detection effectiveness and the percent of seeded faults and of naturally occurring faults, of those known, that were found.

### 4.1.2 Independent Variables

The independent variable will be the web application testing method. We will apply both the Atomic Section Model to the web application as well as Qian, Miao, and Zeng's model. Other aspects of the experiment that will be controlled include the browser used to follow each model's test paths, as well as the features enabled in the browser. Both Javascript and CSS will be enabled in the browser, and we will execute the tests in the Firefox web browser<sup>3</sup> on separate computers.

## 4.2 Hypotheses

The null and alternate hypotheses are listed in Table 9 below.

Table 9, Hypotheses

Hypothesis	Description
H <sub>01</sub>	ASM will find the <i>same</i> number of faults total, across all fault types, as QMZ
H <sub>02</sub>	ASM will find the <i>same</i> specific faults as QMZ
H <sub>03</sub>	ASM will find the <i>same</i> number of faults in a given fault type as QMZ, but not necessarily the same number of faults across all fault types as QMZ
H <sub>a1</sub>	ASM will find <i>more</i> faults, across all fault types, than QMZ
H <sub>a2</sub>	ASM will find <i>more</i> faults in a given fault type than QMZ, but not necessarily more faults across all fault types than QMZ

<sup>3</sup><http://www.mozilla.com/en-US/firefox/personal.html>

## 4.3 Threats to Validity

This section discusses potential problems in our experimental design that might affect the validity of the results.

### 4.3.1 Conclusion Validity

The data collected (i.e., number of faults, types of faults, which specific faults were discovered) to draw our conclusions on whether ASM performs better than QMZ could be affected by the same factors that threaten the external validity of the experiment (see section 4.3.4 External Validity below.)

### 4.3.2 Internal Validity

The only intentional dependent variables are number of faults, which faults, and types of faults. There is only a single independent variable (i.e., the model being applied), so the only reason why the dependent variables would differ from one treatment to the other would be because of the independent variable. However, only one researcher was responsible for creating test cases, and both researchers ran test cases independently of each other with no overlap such that both ran the same test case. The former threat could mean the test cases were designed incorrectly without being noticed, the latter threat could mean if a test case were run incorrectly, there would be no redundancy to catch the mistake.

It might also be a threat to internal validity that, while both researchers used the Firefox web browser version 3.5.5, one researcher ran it in Windows XP while the other ran it in OS X.

Another threat to internal validity could be that we did not use all the methods of application coverage that were listed in the ASM model. We chose to use atomic section coverage [1] and invalid access coverage [1]. We did not apply prime path coverage [1] and invalid path coverage [1]. It is possible that other faults could have been detected with the ASM model had these different types of coverage been considered when designing test cases.

### 4.3.3 Construct Validity

The Atomic Section Model and the Qian, Miao, Zeng model may not have been applied correctly, being that we had not had any previous experience with either model. It is also possible that fault classification is a threat to validity since it is slightly subjective and only one researcher was familiar with Ruby on Rails development.

### 4.3.4 External Validity

The web application with which the experiment was conducted was built using the Ruby on Rails framework. It is uncertain whether the results found from this experiment will generalize to other web applications that were not written with this framework. Three undergraduate students, only one of which had any previous experience with the Ruby on Rails framework, developed the web application that was tested. Had professionals developed the application, fewer faults or different types of faults might have occurred.

Since AJAX and Javascript were ignored in the application of both models, this experiment will not generalize to web application testing where Javascript functionality is not ignored.

The domain of the web application also affects how this experiment will generalize to other web applications. For example, a bank's web application should definitely have fewer defects due to the severe problems that could occur due to its failure, and perhaps faults of different types, than a game web application. The number and type of tolerable faults for a particular domain may be better found with test paths from one model than the other.

The number of faults that we seeded could be a threat to validity as well as the types of faults that were seeded. We tried not to use contrived examples and read through the Subversion<sup>4</sup> change log for the web application to find real-life bugs that could be re-seeded. However, many of the real bugs found in the change log dealt with Javascript and AJAX which, as has already been stated, were ignored in this experiment.

## 5 Analysis and Discussion

The ASM and QMZ models generated different test cases. The counts for each are shown in Table 10 below.

Table 10, Test Cases per Model

Model	# Test Cases
Atomic Section Model	24
Qian, Miao, Zeng model	53

One reason for the great difference between test case count for the two models is in how test cases are generated for each. Since we used atomic section coverage for ASM, fewer test cases were necessary since each atomic section needed to be covered only once. The QMZ model

<sup>4</sup> <http://subversion.tigris.org/>

generated more test cases because it had many K flow components that could be used to leave a page, such as all the navigational links that appeared on each page, or different buttons in forms. Also, the QMZ test cases tended to be only a few steps long, whereas the ASM test cases had more steps per test case.

Table 11 below shows statistics on passing and failing test cases for each model.

Table 11, Passing and Failing Test Statistics

Model	# Pass	% Pass	# Fail	% Fail
ASM	9	37.5%	15	62.5%
QMZ	27	50.94%	26	49.06%

If a test case failed, that is indicative of a fault. Table 12 below shows which faults were detected by which models, based on which test cases failed.

Table 12, Detected Faults by Each Model

Model	Test Case	Faults Found
ASM	Create Game 1	S1
ASM	Create Game 2	S6, S7
ASM	Create Game 3	S1
ASM	Create Game 4	N4
ASM	Create Game 6	N5
ASM	Create Game 7	N5
ASM	Create Game 8	N5
ASM	Create Game 9	N6
ASM	Invalid Access Coverage	S2, S15
ASM	General Access 1	S6
ASM	General Access 3	S2, N3
ASM	Settings/Admin 2	S4, S9
ASM	Settings/Admin 3	S8
ASM	Register 1	S10
ASM	Settings/Admin 1	S12
QMZ	Registration 3_1	S2, S10, N3
QMZ	Registration N3	S2
QMZ	Login 6_1	S2
QMZ	Login 6_2	S2
QMZ	Login 6_3	S2
QMZ	Login N6	S2
QMZ	Create Game 8_1	S1
QMZ	Create Game 8_2	S1, N6
QMZ	Create Game 8_3	S1, N4
QMZ	Create Game N_8	S1, N5
QMZ	Current Game Link 9_1	S1
QMZ	Create Game 9_2	S1
QMZ	Create Game 9_3	S1, N6
QMZ	Create Game 9_4	S1, N4

QMZ	Create Game 9_5	S1, N5
QMZ	Tab Test 1 (regular user)	S6
QMZ	Tab Test 1 (admin user)	S1, S6
QMZ	Update Button Test 1	S4, S9
QMZ	Rules Tab Test 1	S1
QMZ	Top 20 1_2	S1
QMZ	Top 20 Logout	S6
QMZ	Rules Tab to Other Tabs Test 2 (admin user)	S1
QMZ	Admin Settings 1_1	S4, S9
QMZ	Admin Settings 1_2	S8
QMZ	Admin Unapprove N	N7
QMZ	Settings Test 1	S12

Table 13 below shows counts of which model found the most faults by fault category and overall.

Table 13, All Faults versus Faults Detected per Model

Model	Fault Category	All Faults	Faults Detected	% Detected
ASM	Assignment	1	1	100%
ASM	Checking	7	5	71.4%
ASM	Function	6	2	33.3%
ASM	Interface	6	5	83.3%
ASM	Unanticipated case	1	0	0%
ASM	Wrong type	1	1	100%
<b>ASM Total</b>		<b>22</b>	<b>14</b>	<b>63.6%</b>
QMZ	Assignment	1	1	100%
QMZ	Checking	7	5	71.4%
QMZ	Function	6	2	33.3%
QMZ	Interface	6	5	83.3%
QMZ	Unanticipated case	1	0	0%
QMZ	Wrong type	1	1	100%
<b>QMZ Total</b>		<b>22</b>	<b>14</b>	<b>63.6%</b>

The same number of faults was found overall by both models, providing support for null hypothesis  $H_{01}$ . The same faults were found or not found by both models in all categories except Interface: ASM found S15 but QMZ did not, and QMZ found N7 while ASM did not. These two faults are described as follows:

- S15: “Non-logged in users can attempt to access the settings page but it will give an application error instead of just redirecting them.”
- N7: “When unapproving users as an admin and no users are selected, message

should be 'No Users Selected to unapprove' instead of 'No Users Selected to approve'."

It makes sense that QMZ did not find S15 since QMZ relies entirely on flow components (e.g., links, form buttons) to navigate between pages, whereas ASM acknowledges that a user is not limited to these connections alone. A browser has history, and a user can easily navigate to a page he or she has visited in the past but should no longer be able to access, and the application must account for such use. ASM was able to catch S15 through its invalid access coverage.

## 5.1 Undetected Faults

There were some faults that neither model detected. This section discusses those faults and why they remained undetected.

### 5.1.1 Undetected Seeded Fault S3

It is described as "No longer marking `check_id` and `check_post` as hidden in LoginController, meaning a user can manually edit the URL and access these as pages, but they are supposed to be internal. This is specific to the Rails framework." This is a fault that makes visible two URLs to which the user should not have access. However, since neither of these URLs are linked anywhere within the web application's pages/screens, neither model was able to find them. This is indicative of a problem with both models: they rely entirely on the user interface to generate test paths. However, if a user were somehow able to discover URLs that were not linked in the interface but nevertheless accessible on the server, possible faults could be uncovered that were previously untested.

### 5.1.2 Undetected Seeded Fault S5

It is described as "Do not check to see that a user's account has been approved before redirecting him/her to the game area upon logging in." Both models could possibly have detected this fault, however we did not cycle through page flow components sufficiently in the QMZ model, or use more thorough test generation criteria in ASM, to reach this fault. It would be necessary for a user to register a new account and then return to the login page to attempt logging in. Our tests included registering a new account and then returning to the login page, but not attempting to log in, only ensuring that the login page loaded correctly.

### 5.1.3 Undetected Seeded Fault S11

It is described as "List of Current Games on game/index should only include unfinished, playable games for the user, but instead will show all games associated with the user, even those that are complete and thus unplayable." To discover this fault would have required more sample data loaded into the database. This fault could have been detected through our manual test execution since we would have noted that incorrect games were listed on the Games page for a given user.

### 5.1.4 Undetected Seeded Fault S13

It is described as "Logged-in users will no longer be redirected to the game page if they manually go to the login page." This fault was not detected because, once a user has logged in, the link back to the Login page does not appear. The QMZ model does not generate any test paths to pages unless a page flow component is present. With ASM and atomic section coverage, it was not necessary to alter the URL in the browser to return to the login page because the atomic sections of the login page had already been accessed. It is possible that in choosing another coverage criteria for ASM, this fault could be detected.

### 5.1.5 Undetected Seeded Fault S14

It is described as "Modified email validation when a user registers such that it is more naïve: it will allow valid emails such as `user@example.com` but also invalid emails such as `user@example...com`." This fault could have been detected with more thorough input data to existing test cases for both models.

### 5.1.6 Undetected Naturally Occurring Fault N1

It is described as: "User can change game's status from "finished" to "in progress" by visiting the game play page." It is possible both models could have generated a test path detecting this fault. For the QMZ model, one test path fell only slightly short of detecting this fault: it required the tester log in, verify the Games screen displayed correctly, click a link to access a game, and verify the Game Play screen displayed correctly. A single step further, requiring the user to click the Games tab, could have led to the detection of this fault. We tried to avoid doing anything further after reaching the Game Play screen because that screen uses a large amount of Javascript, and we did not want to risk using any

untestable Javascript functionality. Likewise with extended ASM coverage, this fault could have been detected.

#### *5.1.7 Undetected Naturally Occurring Fault N2*

It is described as “User only sees games associated with his/her user ID, instead of games by all users.” This fault could have been detected if more sample data had been loaded into the database, as with seeded fault S11 described above in section 5.1.3. It would have been noticed when the tester viewed the Games screen, which both models accessed.

## 6 Conclusions and Future Work

We have applied the Atomic Section Model and the Qian, Miao, Zeng model to a single Ruby on Rails web application and described the fault detection results. Both models performed adequately, detecting 63.6% of all seeded and known naturally occurring faults. ASM found one fault that QMZ did not, and vice versa, both in the Interface category. The null hypothesis could not be rejected since both models found the same number of faults overall and in each category.

Future work could include finding a second web application to which both ASM and QMZ could be applied. It would be helpful if the additional web application were not a Ruby on Rails application, so as to help generalize the results beyond Ruby on Rails or the Model-View-Controller paradigm that it uses.

A third web application testing method could also be applied to see if it performs better at fault detection than either ASM or QMZ. Perhaps the Elbaum et al. models that take into account user session could be tested.

The Atomic Section model could be reapplied using more extensive coverage criteria. This would possibly help to detect more faults, especially if more extensive sample data were loaded into the database such that more faults would be possible to detect.

If a testing model that accounts for Javascript functionality were found, it could be applied to the Javascript and AJAX used within this Ruby on Rails application. The Game Play screen could then be tested, though of course no comparison could be made with either ASM or QMZ since they do not take Javascript into account.

It would expedite the model application and test generation process if tools could be written, for example, to extract atomic sections from Ruby on Rails files. It would also be helpful to have a tool that extracts flow components from pages and then generates test paths for the QMZ model.

## 7 References

---

- 1 Offutt, Jeff and Wu, Ye, “Modeling Presentation Layers of Web Applications for Testing”, Springer’s Software and Modeling, DOI: 10.1007/s10270-009-0125-4.
- 2 Qian, Miao, and Zeng, “A Practical Web Testing Model for Web Application Testing”, Third International IEEE Conference on Signal-Image Technologies and Internet-Based System, DOI 10.1109/SITIS.2007.16.
- 3 Filippo Ricca and Paolo Tonella, “Analysis and testing of Web applications”, International Conference on Software Engineering, Proceedings of the 23<sup>rd</sup> International Conference on Software Engineering, 2001.
- 4 Edward Hieatt and Robert Mee, “Going Faster: Testing The Web Application”, IEEE Software, Volume 19, Issue 2, March 2002.
- 5 Sebastian Elbaum, Gregg Rothermel, Srikanth Karre, and Marc Fisher II, “Leveraging User-Session Data to Support Web Application Testing”, IEEE Transactions on Software Engineering, Volume 31, No. 3, March 2005.
- 6 Ploski, Rohr, et al., “Research Issues in Software Fault Categorization”, ACM SIGSOFT Software Engineering Notes, November 2007 Volume 32 Number 6.