University of Nevada, Reno

A UML-Based Approach for Testing Web Applications

A professional paper submitted in the partial fulfillment
of the requirements for the degree of
Master of Science
with major in Computer Science

By
Manish Nilawar

Dr. Sergiu Dascalu, advisor

August 2003

Abstract

Web applications have evolved from small web site add-ons to large multi-tiered applications. Many web applications have an impressive number of concurrent users distributed all over the world. Building web applications is serious business, as they are becoming progressively more intricate as well as mission critical. Systematically modeling and testing these applications can help manage their growing complexity. In this paper, we describe an approach for engineering web applications with UML and, as key part of this approach, focus on testing such applications using a strategy we denote 'combo testing'. Combo testing involves the use of a new modeling construct that we propose, 'web application compound' (or, in short, 'web compound'), and allows thorough verification of functional requirements of web applications via two types of testing: use case-based testing and web compounds unit testing. As an example, we have built a UML model and implemented the prototype for Web Records, a software tool that serves the event management requirements of an organization through a database driven architecture with a web front end. Through Web Records we illustrate our approach and show how developers can exploit UML and its extension mechanisms in modeling and testing web applications.

Keywords: web applications, UML extensions, modeling, web compounds, combo testing.

Acknowledgments

First and foremost, I would like to express my deepest appreciation to my advisor Dr. Sergiu Dascalu for his extraordinary level of support and guidance.

I am very grateful to Dr. Harris and Dr. Meerschaert for their valuable time and suggestions and for serving on my committee.

I would also like to thank George McKinlay and Dr. Warner for helping me learn this great technology, WebObjects.

On a personal note, I would like to thank my family for their support and encouragement throughout my education.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Background and Related Work

Web applications are typically software programs that operate on the Internet, interacting with the user through an Internet browser. Web applications require the presence of Web server in simple configurations and multiple servers in more complex settings. Such applications are more precisely named *web-based applications*. Similar applications, which may operate independent of any servers and rely on operating system services to perform their functions, are termed *web-enabled applications*. These days, with the integration of technologies used for the development of such applications, there is a thin line of separation between *web-based* and *web-enabled applications*, so in this paper we collectively refer to both of them as *web applications*.

Although there are a number of differences between web applications and traditional software, some of the testing strategies for traditional software applications can be adapted or modified [1]. Most web applications tend to be object-oriented in nature [2]. Up to now most of the web testing approaches have concentrated on the client-side validation and other static aspects of the web application such as server-side validation using Javascript validation tools and HTML validators. Research into functionality testing of web applications is a recent trend [3].

For instance, Ricca and Tonella [4] proposed a model for laying out analysis and testing strategies for web applications. Their strategy is primary built around static webpage analysis and some preliminary dynamic analysis. As more and more information processing is being done dynamically using web applications, the focus should be shifted on the dynamic behavioral aspects of the applications. Lee and Offutt [5] have described a mutation based testing approach for XML-based web applications. The key idea behind their approach is consistency of data exchange among different components via XML. Nevertheless, functionality checking of the web application is not focused in their work. Conallen [6] has suggested comprehensive modeling strategies using UML [7, 8, 9] extensions and establishing a mapping between standard modeling constructs and the artifacts of a web application. Yet, testing was not considered as part of the work other than some basic web page analysis. Kung et al [10, 11] have developed a model to represent web sites as a graph, and provide some preliminary definitions for developing tests based on the graph in terms of web page traversals. It does not take into consideration dynamic behavioral aspects based on server-side code.

From the above, it can be seen that testing web applications is a research of recent date, in particular UML-based approaches for testing such applications are still in their early stages and significant work seems to be left for researchers and developers. The primary motivation for our research has been to work in this rather uncharted area of UML-based web testing approach, which has a significant practical application and can lead to considerably high benefits.

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding. Testing presents an interesting anomaly for the software engineer. During earlier definition and development phases, the engineer attempts to build software from an abstract concept to a tangible implementation. Now comes testing. The engineer creates a series of test cases that are intended to demolish the software that has been built. In fact, testing is one step in software engineering process that could be viewed as rather destructive than constructive. Yet, its importance and utility can never be underestimated.

An engineered product can be tested in one of the two ways:

1. Knowing the specified functions that the product has been designed to perform, tests can be conducted to demonstrate that each function is fully operational.
2. Knowing the internal workings of a product, tests can be conducted to ensure that "all the gears mesh", that is, that the internal operation of the product performs according to specification and all internal components have been adequately exercised [19].

The first approach is called *black box testing* the second *white box testing*. The formal definitions are as below [19]:

- *Black box testing*. Black box testing enables the software engineer to derive sets of input conditions that will fully exercise all functional requirements for the program.
- *White box testing*. White box testing is a test case design method that uses the control structure of the procedural design to derive test cases.

In this paper we focus on *black box testing* which generally is more suitable and more necessary for web applications.

1.2 Overview of the Proposed Approach

The proposed approach uses UML for modeling and testing of web applications. In particular, testing has received a special focus. The modeling steps proposed have been adapted from the software engineering methodological guidelines of [6]. For the purpose of modeling, we have introduced the concept of *webcomps* (short form for *web compounds* or *web application compounds*). For testing, we suggest a systematic allocation of use cases to *webcomps*. We also propose a *combo testing* approach, which involves two components:

- Use-case driven testing
- *Webcomps* unit testing

An illustration of this approach has been demonstrated using a web application called Web Records whose initial working prototype has been implemented.

The remainder of the paper is organized as follows. Section 2 presents a closer look at the architecture and functions of Web Records. Section 3 discusses the UML-based modeling approach. Section 4 concentrates on the *combo* testing approach with detailed description of both its component methods. Section 5 describes the directions of future work. Finally, Section 6 summarizes the contributions and presents conclusions of the paper.

2 Web Records: Architecture and Functions

To illustrate our approach we have developed a web application denoted Web Records. Web Records is a database-driven web application that helps an organization in managing the data gathering needs such as registration forms and paper submissions for any event like conference, seminar or training workshop.

Figure 1 depicts the overall architecture of Web Records. The application is well supported on all browsers that conform to HTML 3.2. The Web Server uses Common Gateway Interface (CGI) and the HTTP adaptor connects the application to the Internet by acting as an intermediary between the application instances and Web Server. The Web Records application itself responds to the requests and responds usually in the form of dynamically generated Web pages.

Web Records uses an intuitive, easy to use, graphical user interface. It is meant to be an online event management solution that addresses the needs of an organization hosting the event. Administrators, participants, and reviewers need only a web browser to utilize the application. Remote access from any location is possible by anyone with an Internet connection and a web browser, regardless of platform. The application shall allow users to register with organization and then sign up for any events being hosted by the organization. Unregistered users shall view and browse through the details of all the available events but not register for any of those events. Access to all other information within the application is restricted to registered users only. Administrators can log in to the system to check the status of participation for each event. Detailed statistics regarding the attendance of participants such as region-wise analysis can also be viewed. Reports to view the list of participants for each event can be generated on the fly. Web records can also serve the needs of a special event like seminar or conference. The application can accept the submissions from participants in Word or PDF format through a very simple web interface. The submissions are stored in a database and retrieved when queried for. Reviewers of submissions can also login to the same system and get access to view and comment each submission.

Other similar products [12, 13] that supply similar functionality are more sophisticated than Web Records, but this project is aimed to be a simple implementation and sow the seeds for future development.

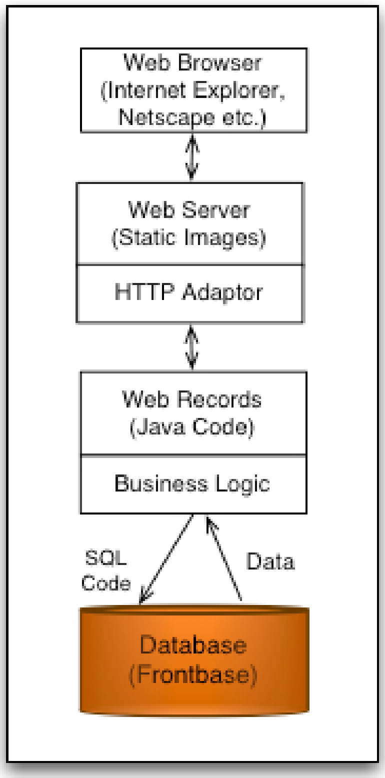


Figure 1 Web Records Architecture

3 UML-Based Modeling

3.1 Concepts

Web applications are becoming progressively more intricate as well as mission critical. Modeling can help to manage this growing complexity. The current standard for modeling software-intensive systems is the Unified Modeling Language (UML). In order to utilize a single modeling notation for an entire system that includes Web-specific components and traditional middle-tier components, UML must be extended.

UML provides several extension mechanisms to allow modelers to make some common extensions without having to modify the underlying modeling language. In doing so, the modeler should carefully weight the benefits and costs before using extensions especially when existing mechanisms will work reasonably well.

Following are the three types of extensibility mechanisms provided with UML [7]:

1. *Stereotypes*. Allow the creation of new building blocks from the existing ones. Extend the language by allowing the addition of new, problem-specific model elements.
2. *Tagged values*. Attach new information to an existing modeling element. Extend the language by addition of new properties.
3. *Constraints*. Extend the semantics of UML building blocks by adding new rules or modifying the existing ones.

Here we introduce and discuss an extension to the UML, making use of its formal extension mechanism [7, 8, 14]. The extension has been designed so that Web-specific components can be integrated with the remainder of the system's model. It is meant to exhibit a level of abstraction and detail suitable for application designers and implementers. A *stereotype* allows you to define a new semantic meaning for a modeling element. *Tagged values* are key values that can be associated with a modeling element. They allow the modeler to attach any value to a modeling element. Web pages, whether scripted or compiled, map one-to-one to components in UML. Components are physical and replaceable parts of the system. The *implementation view* of the model, also known as the *component view*, describes the components of the system along with their relationships [14].

The project Web Records has adopted UML modeling techniques for analysis and design. The notion of *web application compounds or web compounds,* in short *webcomps,* has been introduced here and used for the modeling of this project. Figure 2 depicts the composition of such a *webcomp*. *Webcomps* help in abstracting aspects of navigation as well as business logic representation. A *webcomp* represents logical grouping of parts of a web application that together represent a given functionality. These may include client side HTML code as well as server side code like Java, ASP, and so forth. Packages in UML also imply similar grouping of smaller subsystems, but *webcomps* are specifically composed of the same types of elements with similar structure, i.e. HTML (client-side code) and Java (server-side code). Packages can be more heterogeneous and *webcomps* and packagesdistinguish from each other because of their distinctive structure and functionality.

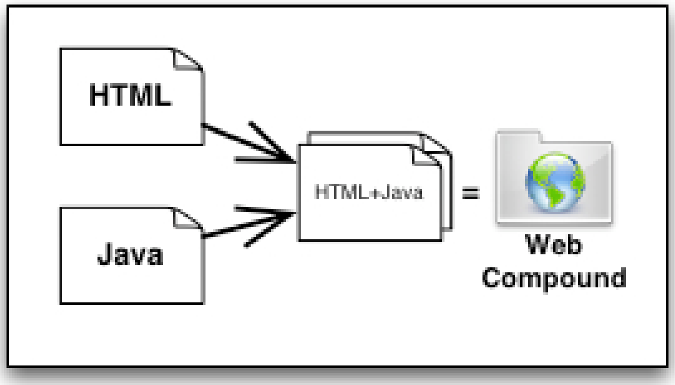


Figure 2 Composition of a Typical *Webcomp*

As shown in [6, 15], stereotypes can be used to represent connections between *webcomps* through hyperlinks, requests to server code and so forth. Tagged values can be used to represent parameters passed across these *webcomps*. Figure 3 shows an example of how a web application can be modeled using the notion of *webcomps*. If the user clicks on the hyperlink "Forgot Password," the users email ID is passed as a tagged value between the `Main` *webcomp* and `Forgot Password` *webcomp* and the hyperlink has been abstracted using the *link* stereotype. So, stereotypes can be used as a mechanism for representing navigation and tagged values for passing data among *webcomps*.

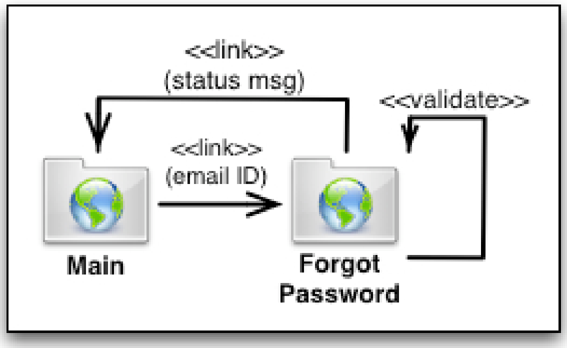


Figure 3 *Webcomp* Model

3.2 Deliverables

Specific deliverables, based on [14] and following the guidelines suggested by Dr. Dascalu in the software engineering courses he taught in the spring of 2003, are grouped in the following documents.

1. *Software Requirements Specification* (SRS) with the following contents:

    - Introduction
    - General description
    - Requirements specification
        i. Requirements that will be satisfied by Web Records
        ii. Requirements that may be satisfied by Web Records
    - Use case modeling
        i. Use case diagrams
        ii. Detailed use cases
        iii. Scenarios
        iv. Requirements traceability matrix
    - Glossary

2. *Design Document* (DD) with the following contents:

   - Introduction
   - Overview
   - High-level design: class diagram and program structure with database schema
   - Low-level design: selected items of detailed design including state diagram and component diagram and pseudo code descriptions
   - User interface design

3. *Source code packages* (SCP) including:

   - Source code for all program units
   - Testing specifications, details on these being described later in the paper

3.3 Use Case Modeling

According to [16], on the basis of design the Web Records application can be classified as a *thin web client*, where standard facilities of the Web browser are being exploited for user interaction, but all the business logic is being executed on the server. This use case diagram shown in Figure 4 depicts the functionality of Web Records. In our proposed approach the use case diagram of the application needs to be expanded to derive the *webcomps* model. As it can be seen in Figure 4 three actors interact with the system to perform the required functions. Use cases have been derived on the basis of the functional requirements of the application.

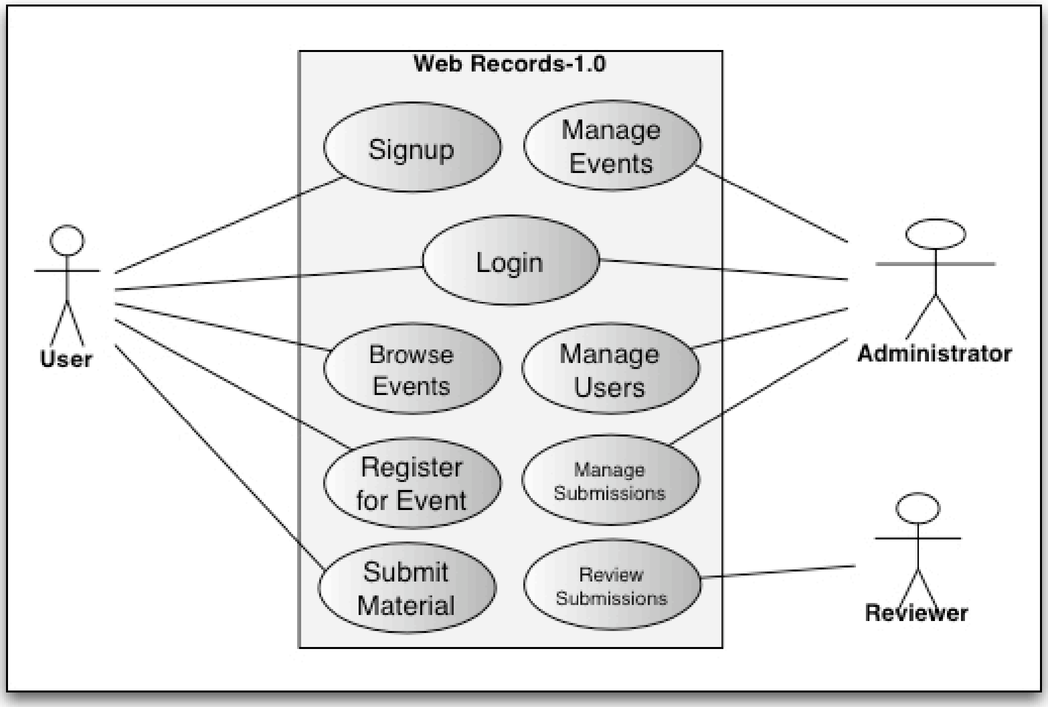


Figure 4 Use Case Diagram of Web Records

3.4 Elaboration of Web Compounds

Webcomps take the modeling of the application to a level of abstraction that comes closer to the implementation. The purpose is to focus on the underlying conceptual characteristics associated with the development approach without worrying about how the representations look like [17]. The *webcomps model* also addresses the key elements of web applications, the *interactions* [3]. This can be demonstrated using a navigation map (site map), as shown in Figure 5.

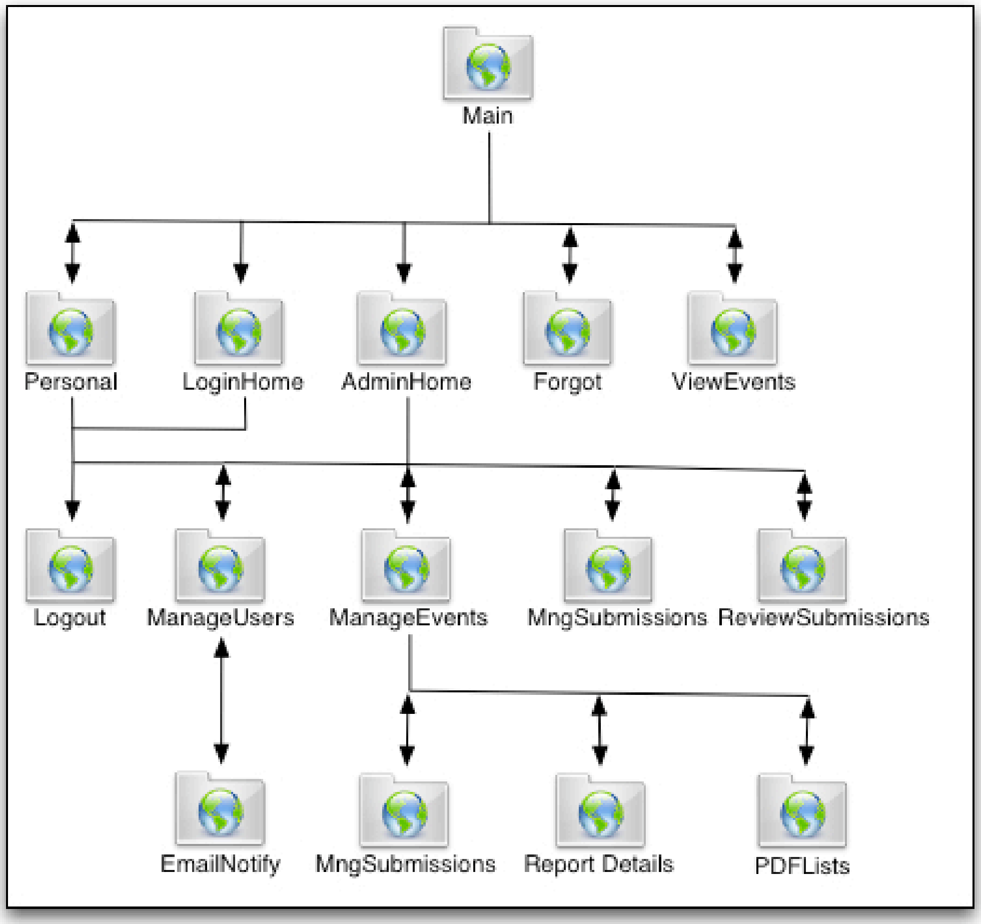


Figure 5 *Webcomps* Navigation Map

The next modeling step after use cases and *webcomps* are elaborated is to achieve a mapping between these two. Using a systematic process, we can map each use case to one or more *webcomps* that will be used to implement the functionality of the use case. This mapping is important because it will be used during *combo testing* to thoroughly verify how the system implements its required functionality. The mapping of use cases to *webcomps* is shown in Figures 6 and 7.

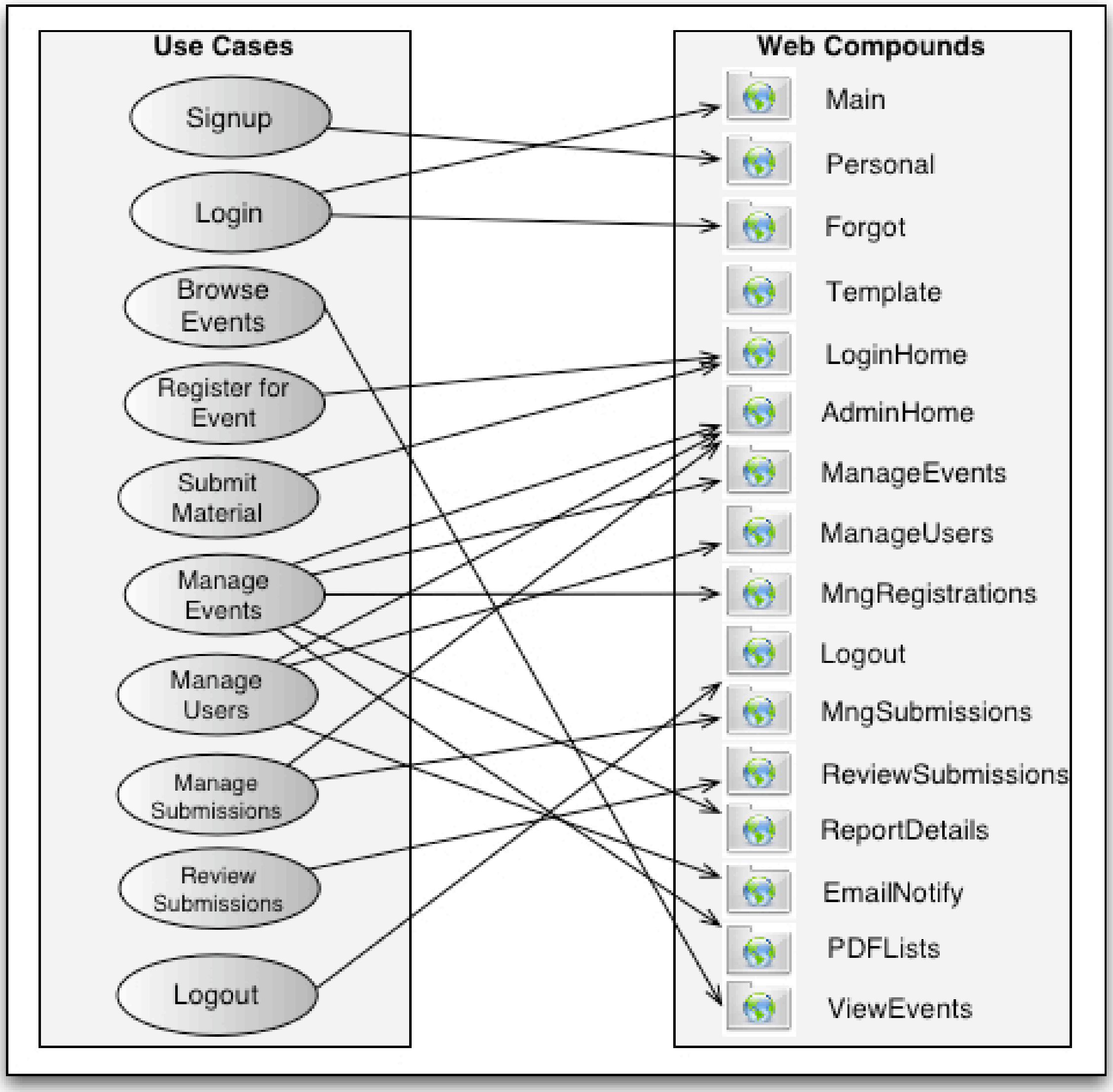


Figure 6 Mapping Use Cases to *Webcomps*

Figure 6 provides a visual correspondence between the two types of modeling constructs and Table I shows the same in a tabular form. It should be noted that in Figure 6 the *webcomp* Template is only used for display purposes and does not deliver any functionality, by itself hence it is not mapped to any use case.

Table I: Mapping Table for Use Cases and *Webcomps*

Use Case	Webcomps
Signup	Personal
Login	Main, Forgot
Browse Events	ViewEvents
Register for Event	LoginHome
Submit Material	LoginHome
Manage Events	ManageEvents,MngRegistrations, ReportDetails, PDFLists
Manage Users	ManageUsers, EmailNotify
Manage Submissions	MngSubmissions
Review Submissions	ReviewSubmissions
Logout	Logout

4 Testing

4.1 Combo Solution for Testing

The most natural way of verifying a system is just to operate it in some representative situations and verify whether it behaves as expected [18]. Testing is a process of executing a program with the intent of finding an error. If testing is conducted successfully, it also demonstrates that the software functions appear to be working according to the specification and the performance requirements are met. In addition data collected during the testing process provides a good indication of software reliability and some indication of the software quality as a whole [19]. Testing, particularly software testing can have a different meaning for different people. To be more specific, we concentrate on *verification testing,* i.e. proving that Web Records has the required functionality based on the *Software Requirements Specification* (SRS) document. The approach involves *use cases based testing* and *webcomps unit testing* and treat the Web Records application itself as a *black box*.

4.2 Use Case Based Testing

4.2.1 Use Cases Towards Testing

A use case is a top-level category of system functionality, (i.e. Login, Register for Event, etc.)[20]. A use case has a graphical representation and a text description. The diagram identifies all the actors (outside of the system) involved in the function, and the text description provides an indication of how the use case is initiated and then evolves. The collection of *use case diagrams* provides a 'context' diagram of system interfaces. Each use case constitutes a complete list of events initiated by an actor and specifies the interaction that takes place between an actor and the system. In a use case the system is viewed as opaque, where only the inputs, outputs, and functionality matter [21, 22].

The purpose of a use case may include [23]:

- Promoting communication
- Understanding requirements
- Focusing on "what" rather than "how"
- Providing prototype test cases

Use cases can serve to elaborate the scenarios that ensure the functionality required can be supported and can be used to discover the objects (in the class diagram) that will construct a system that satisfies all functional requirements. Conceptually, we can view the functionality as a set of processes that run horizontally through the system, and the objects as sub-system components that stand vertically. Each functionality does not use every object, but each object may be used by many functional requirements. This transition from a functional to an object point-of-view is accomplished with use cases and scenarios.

4.2.2 Scenarios: The Link between Use cases and Test cases

A *scenario* is an instance of a use case, or a complete "path" through the use case [3]. End users of the completed system can go down many paths as they execute the functionality specified in the use case. Basically, scenarios give the use cases the ability to generate multiple flows of events. Each use case has a *primary scenario* and multiple *secondary scenarios*. The primary scenario can have its flow of events and each of the secondary scenarios can have its own flow of events, as shown in Figure 7. The primary scenario's flow of events should cover what "normally" happens when the use case is performed. The secondary scenario's flows of events cover behavior of an optional or exceptional character relative to normal behavior, and also variations of the normal behavior.

Figure 8 shows an example of a scenario for the "Register for Event" use case from Web Records. This use case is used as a prototype module for the purpose of implementing the test strategy on Web Records.

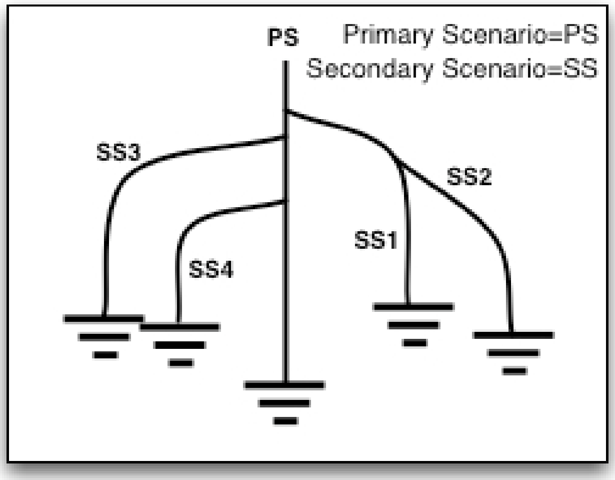
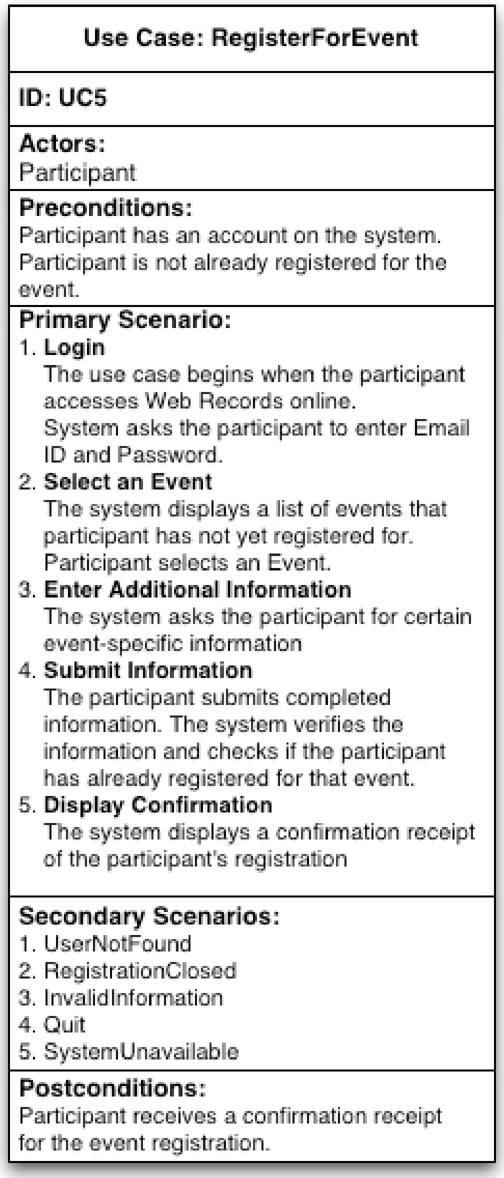


Figure 7 Multiple Flow of Events in a Use Case

The flow of events for primary scenario yields a successful registration for the participant. Each secondary scenario has its own flow of events. So, a significant amount of detail goes into fully specifying a use case. The *precondition* specifies the required state of the system prior to the start of the use case. This can be used for a similar purpose in the test case. The *postcondition* is the state of the system after the actor interaction. This may be used to check pass/fail criteria during testing.

4.2.3 Generating Test Cases

A test case is a set of test inputs, execution conditions, and expected results developed for a particular objective, for example to exercise a particular program path or verify compliance with a specific requirement, for example. The purpose of a test case is to identify and communicate conditions that will be implemented in the test. Test cases are necessary to verify successful and acceptable implementation of the product requirements (use cases).



Use Case: RegisterForEvent

ID: UC5

Actors:
Participant

Preconditions:
Participant has an account on the system.
Participant is not already registered for the event.

Primary Scenario:
1. **Login**
   The use case begins when the participant accesses Web Records online.
   System asks the participant to enter Email ID and Password.
2. **Select an Event**
   The system displays a list of events that participant has not yet registered for.
   Participant selects an Event.
3. **Enter Additional Information**
   The system asks the participant for certain event-specific information
4. **Submit Information**
   The participant submits completed information. The system verifies the information and checks if the participant has already registered for that event.
5. **Display Confirmation**
   The system displays a confirmation receipt of the participant's registration

Secondary Scenarios:
1. UserNotFound
2. RegistrationClosed
3. InvalidInformation
4. Quit
5. SystemUnavailable

Postconditions:
Participant receives a confirmation receipt for the event registration.

Figure 8 Scenario for "`Register for Event`" Use Case

The four-step process for generating test cases described below has been adapted from [24]. To this process we have added an initial step, which involves the development of a prioritization mechanism. Also, we have suggested some additional guidelines for generating scenarios.

Step 1: Prioritize use cases based on the *requirements traceability matrix*.

Step 2: Generate tentatively sufficient use case scenarios for each of the use cases.

Step 3: For each scenario, identify at least one test case and the conditions that will make it execute.

Step 4: For each test case, identify the data values with which to test.

Step 1: Prioritize Use Cases

As mentioned before, most web applications are developed under severe time-to-market pressures. Development periods may range from one to three months for a vast majority of such applications with relatively small team of developers. In such situations, addressing the user's priorities can be very crucial. Regardless of the rigorousness of the design process, one thing holds true: the frequency with which each function of the system is used, will reflect the relative importance of the specific system features. The familiar use-case model of system requirements can play a part in computing the *requirements traceability matrix*, thus guiding the selection of test cases for the system. We suggest a prioritization scheme as shown in Table II where the *requirements traceability matrix* of Web Records indicates the relative importance of each use case of the system.

Each requirement is assigned a priority weight P_w . We use a simple scheme of high (P_w=2) and low (P_w=1) priority for each requirement. The score for each use case is calculated by summing up all of its priority weights. Use case with greatest score has the highest priority and so forth. So, in this UC2 has the highest priority (score=10). More sophisticated rules can be set for determining use case priorities in complex systems. Essentially it is left to the developers to decide on the number of levels of priority that should be used in a given application.

Table II: Requirements Traceability Matrix

	P_w	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9
R1	2	X	X							
R2	2		X							
R3	2	X								
R4	2		X	X						
R5	2				X					
R6	1		X				X			
R7	1		X					X		
R8	1		X							X
R9	1	X			X	X				
R10	1		X						X	
SCORE		5	10	2	3	1	1	1	1	1

Step 2: Generate Use Case Scenarios

For each use case starting from higher to lower priority, generate a tentatively sufficient set of scenarios. Having a detailed description of each scenario such as the input variables involved, the preconditions and postconditions can be very helpful in the later stages of test case generation [24]. Table III shows the tentatively sufficient set of scenarios for the Register for Event use case. For this approach to work for each use case, we propose that there must be a primary scenario and at least one secondary scenario, which in some cases could be used to represent abnormal behavior of the application. In fact, more secondary scenarios mean more comprehensive modeling and consequently more thorough testing. However, the above minimum requirement in terms of the number of scenarios is intended to allow, flexibility for developers and testers and to ensure that each use case is sufficiently well covered.

Table III: Scenario Set for "Register for Event" Use Case

Scenario ID	Scenario Name
Scenario 1	SuccessfulRegistration
Scenario 2	UserNotFound
Scenario 3	InvalidInformation
Scenario 4	UserQuits
Scenario 5	SystemUnavailable
Scenario 6	Registration closed
Scenario 7	DuplicateRegistration

Step 3: Identify Test Cases

After the tentatively sufficient set of scenarios for the use case has been identified, the next step is to identify the test cases. Analyzing the use case and its scenarios allows this. The preconditions and flow of events for each scenario can be exploited very well to identify the input variables and the constraints that bring the system to a specific state represented by the postconditions. A matrix format is useful in clearly documenting the test cases for each scenario. The representation shown in Table IV demonstrates the test case matrix for `Register for Event` use case of Web Records. In the header row, first column contains Test case ID, the next has Scenario description, and the following column contains a series of input variables and constraints. The last column shows the expected results of the test cases.

The test case matrix represents a framework for testing without involving any specific data values. The V indicates *valid*, I indicates *invalid* and N/A indicates *not applicable*. This matrix is an intermediate step and provides a good way to document the conditions that are being tested. Notice that TC4 and TC5 have identical Vs and Is in all the columns. This means that some more conditions need to be added to it to specifically address these scenarios [24].

Step 4: Identify Data Values

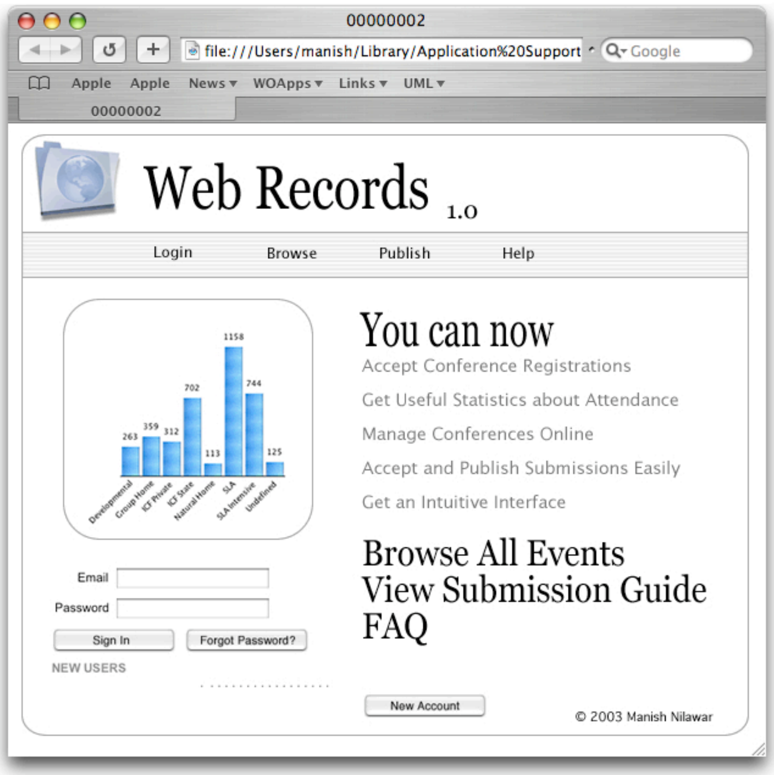
Once all test cases have been identified they should be completed, reviewed and validated to ensure accuracy and identify redundant or missing test cases. If this is ensured then the next step is to plug in the data values for each of the Vs and Is. This can be done using a variety of test data as shown in Table V. Also, before the application is deployed and ready for outside world, it should be tested using real data to see if some other issues like performance prop up. Generating test data is in itself an extensive subject of discussion, which exceeds the scope of this paper [24]. The look and feel of Web Records is shown in Figure 9 and 10. Figure 9 depicts the main login screen and Figure 10 shows the screen where the events available for registration are listed.

Table IV: Test Case Matrix for the `Register for Event` Use Case

Test Case ID	Scenario	Email ID	Password	Not Already Registered	Event Specific Information	Event Registration Open	Expected Results
TC 1	SuccessfulRegistration	V	V	V	V	V	Display confirmation receipt
TC 2	UserNotFound	I	N/A	N/A	N/A	N/A	Error message: Back to login screen
TC 3	InvalidInformation	V	V	N/A	I	N/A	Error message: Re-enter information
TC 4	UserQuits	V	V	N/A	N/A	N/A	Back to login
TC 5	SystemUnavailable	V	V	N/A	N/A	N/A	Error message
TC 6	RegistrationClosed	V	V	N/A	N/A	I	Error message
TC 7	DuplicateRegistration	V	V	I	N/A	N/A	Error message

Table V: Test Case Matrix with Data Values

Test Case ID	Scenario	Email ID	Passwd	Not Already Registered	Event Specific Information	Event Registration Open	Expected Results
TC 1	SuccessfulRegistration	Nilawar@ mac.com	Matrix	Yes	Yes	Yes	Display confirmation receipt
TC 2	UserNotFound	unknown@ mac.com	N/A	N/A	N/A	N/A	Error message: Back to login screen
TC 3	InvalidInformation	Nilawar@ mac.com	Matrix	N/A	Invalid	N/A	Error message: Re-enter information
TC 4	UserQuits	Nilawar@ mac.com	Matrix	N/A	N/A	N/A	Back to login
TC 5	SystemUnavailable	Nilawar@ mac.com	Matrix	N/A	N/A	N/A	Error message
TC 6	RegistrationClosed	Nilawar@ mac.com	Matrix	N/A	N/A	Full	Error message
TC 7	DuplicateRegistration	Nilawar@ mac.com	Matrix	No	N/A	N/A	Error message

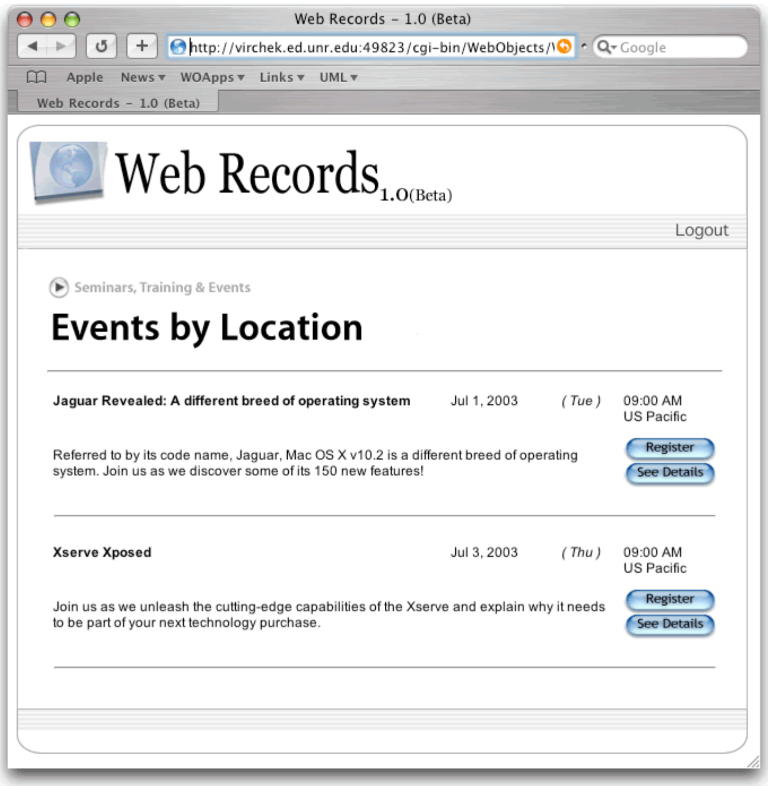


You can now
Accept Conference Registrations
Get Useful Statistics about Attendance
Manage Conferences Online
Accept and Publish Submissions Easily
Get an Intuitive Interface

Browse All Events
View Submission Guide
FAQ

© 2003 Manish Nilawar

Figure 9 Web Records: Main Page



Events by Location

Jaguar Revealed: A different breed of operating system Jul 1, 2003 (Tue) 09:00 AM US Pacific

Referred to by its code name, Jaguar, Mac OS X v10.2 is a different breed of operating system. Join us as we discover some of its 150 new features!

Xserve Xposed Jul 3, 2003 (Thu) 09:00 AM US Pacific

Join us as we unleash the cutting-edge capabilities of the Xserve and explain why it needs to be part of your next technology purchase.

Figure 10 Web Records: `Register for Event Page`

4.3 *Webcomps* Unit Testing

4.3.1 The Approach

The input to the *webcomp* is a formal specification in XML syntax, which specifies the functionality parameters. More precisely, it specifies the input and the expected outputs of each test case. Based on this XML description the test cases are executed from within the *webcomp*. As the result of execution, the generated output is written into a report file as a status code [25, 26].

4.3.2 Test Specification

By formally specifying the web application's intended functionality in test specification, tests can be performed by comparing the actual functionality with the intended functionality. Any web application functionality that can be specified can be tested. A test specification is a hierarchy of test suites and test cases.

- *Test Suite:* A specification consists of one or more test suites. Each test suite contains a set of test cases for one functionality of the *webcomp*.

- *Test Case:* Each test case consists of a pair of request specification and response specification.

4.3.3 Request and Response Specification

- *RequestSpec*. The *RequestSpec* specification depicts a pattern of HTTP requests, which consists of the URL of the request.

- *ResponseSpec*. The *ResponseSpec* specification depicts the assertions on the HTTP response generated from the HTTP request in the same test step.

```
<testsuite>
        <testcase name=”check1”>
                <request url=”http://virchek.ed.unr.edu:57332/cgibin/WebObjects/WebRecords.woa/wo/1”/>
                <response code=”Pass”>
                <match op=”contains” regexp=”false” select=”/html/body” value=”Manage Events”/>
                <OR>
                <match op=”contains” regexp=”false” select=”/html/body” value=”Manage Users”/>
                <OR>
                <match op=”contains” regexp=”false” select=”/html/body” value=”Manage Registrations”/>
                <OR>
                <match op=”contains” regexp=”false” select=”/html/body” value=”View PDF Lists”/>
                <OR>
                <match op=”contains” regexp=”false” select=”/html/body” value=”Logged Out”/>
                </response>
        </testcase>
        <testcase name=”check2”>
                <request url=”http://virchek.ed.unr.edu:57332/cgibin/WebObjects/WebRecords.woa/wo/1”/>
                <response code=”Fail”>
                <match op=”contains” regexp=”false” select=”/html/body” value=”NSException”/>
                </response>
        </testcase>
</testsuite>
```

Figure 11 XML Test Specification File

Figure 11 shows an example of a test specification. The test case requests a web page with initial URL “http://virchek.ed.unr.edu:57332/cgibin/WebObjects/WebRecords.woa/wo/1” Here, the *select “/html/body”* means the match argument is searching the response document. With the above match argument, it specifies that the response page should contain either

Manage Events or Manage Users and so forth. If a valid match is found the URL and the response code *Pass* is written into the report file. If the next test case is found to be valid, and the response page contains the literal string NSException, a status code *Fail* is written into the report file.

4.3.4 Implementation within a *Webcomp*

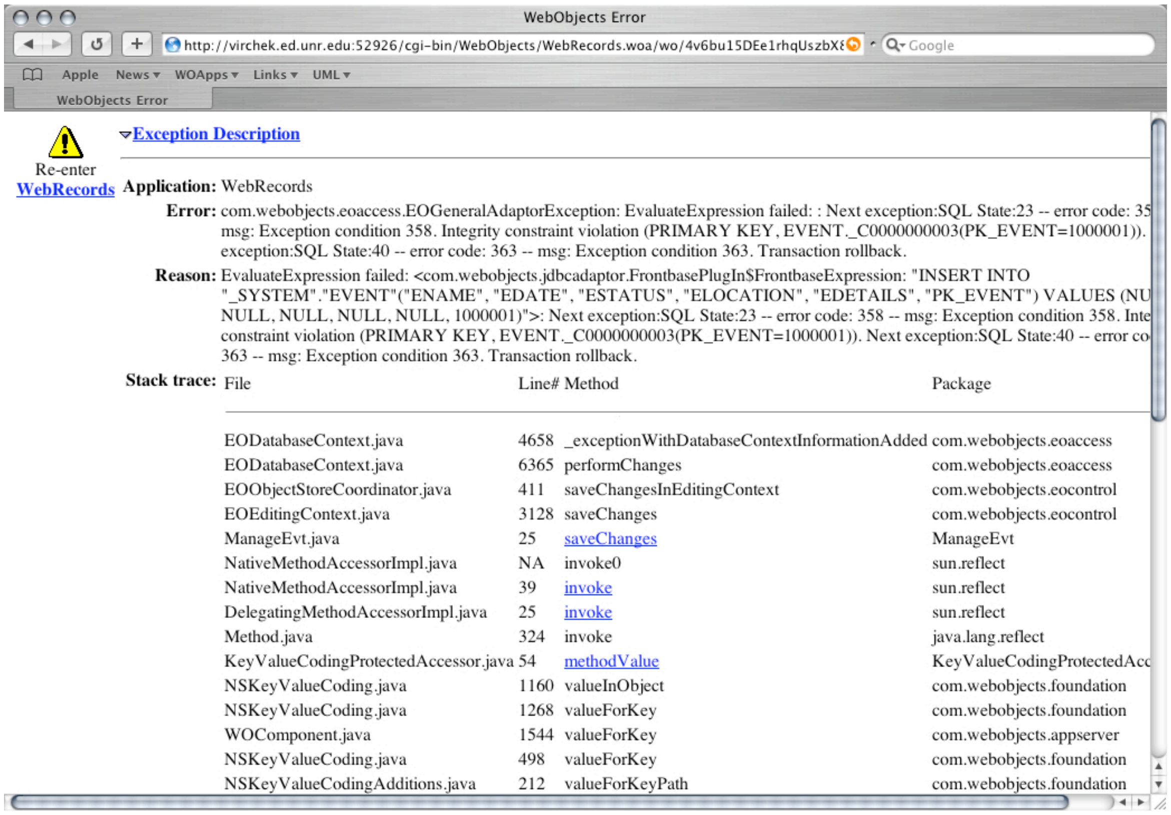
The test specification XML file is stored within the Resources folder of the project such that it is accessible from within any *webcomp* of the project. Built into WebObjects are two HTTP methods used for reading HTTPRequest and parsing the HTTPResponse.

- *takevaluesfromrequest()*. As the request arrives, the first thing the application on the server does is extract the data user has submitted. The data is then passed to the appropriate *webcomp*. The overall effect of the takevaluesfromrequest() method is therefore to synchronize all form values submitted from the browser with their corresponding Java instance variables in the server application
- *appendtoresponse()*. This method is responsible for generating the response to the client. The response is generated in the form of WOResponse object, which is later converted to HTTPResponse to be shown in the browser by the WebObjects adapter.

The input of Web Records is an XML file containing test specifications. Tests are implemented within each *webcomp's* Java code, which uses the Java API for XML Processing (JAXP).

4.3.5 Results of Testing

Several exceptions were caught in the first stroke of execution of the login *webcomp*. Some conditions of exceptions were also artificially generated to verify the results. An example of a screen that depicts an exception in the application is shown in Figure 12. The corresponding test report generated is shown in Figure 13. The test report summarizes a list of request URLs and the corresponding response status codes.



```
Application: WebRecords
      Error: com.webobjects.eoaccess.EOGeneralAdaptorException: EvaluateExpression failed: : Next exception:SQL State:23 -- error code: 35
             msg: Exception condition 358. Integrity constraint violation (PRIMARY KEY, EVENT._C0000000003(PK_EVENT=1000001)).
             exception:SQL State:40 -- error code: 363 -- msg: Exception condition 363. Transaction rollback.
     Reason: EvaluateExpression failed: <com.webobjects.jdbcadaptor.FrontbasePlugIn$FrontbaseExpression: "INSERT INTO
             "_SYSTEM"."EVENT"("ENAME", "EDATE", "ESTATUS", "ELOCATION", "EDETAILS", "PK_EVENT") VALUES (NU
             NULL, NULL, NULL, NULL, 1000001)">: Next exception:SQL State:23 -- error code: 358 -- msg: Exception condition 358. Inte
             constraint violation (PRIMARY KEY, EVENT._C0000000003(PK_EVENT=1000001)). Next exception:SQL State:40 -- error co
             363 -- msg: Exception condition 363. Transaction rollback.
Stack trace: File                                 Line# Method                                   Package

             EODatabaseContext.java               4658  _exceptionWithDatabaseContextInformationAdded com.webobjects.eoaccess
             EODatabaseContext.java               6365  performChanges                           com.webobjects.eoaccess
             EOObjectStoreCoordinator.java        411   saveChangesInEditingContext              com.webobjects.eocontrol
             EOEditingContext.java                3128  saveChanges                              com.webobjects.eocontrol
             ManageEvt.java                       25    saveChanges                              ManageEvt
             NativeMethodAccessorImpl.java        NA    invoke0                                  sun.reflect
             NativeMethodAccessorImpl.java        39    invoke                                   sun.reflect
             DelegatingMethodAccessorImpl.java    25    invoke                                   sun.reflect
             Method.java                          324   invoke                                   java.lang.reflect
             KeyValueCodingProtectedAccessor.java 54    methodValue                              KeyValueCodingProtectedAcc
             NSKeyValueCoding.java                1160  valueInObject                            com.webobjects.foundation
             NSKeyValueCoding.java                1268  valueForKey                              com.webobjects.foundation
             WOComponent.java                     1544  valueForKey                              com.webobjects.appserver
             NSKeyValueCoding.java                498   valueForKey                              com.webobjects.foundation
             NSKeyValueCodingAdditions.java       212   valueForKeyPath                          com.webobjects.foundation
```

Figure 12 Application Exception Screen

```
url="http://virchek.ed.unr.edu:57332/cgibin/WebObjects/WebRecords.woa/wo/1, Pass

url="http://virchek.ed.unr.edu:57332/cgibin/WebObjects/WebRecords.woa/wo/abfe6e/4, Fail
```

Figure 13 Sample Test Report

5. Future Work

Use cases are used to specify the required functionality of an object-oriented system. Test cases that are derived from use cases take advantage of the existing specification to ensure good functional test coverage of the system. We plan several significant research efforts using the proposed *combo testing* strategy. An important step will be to develop algorithms that automatically generate the prioritized use cases, rather than going through a manual process of assigning priorities. An important aspect of web applications is the close tie-up between the navigation and functionality of its web pages. This leads to certain white box testing criteria derived from [27] which include *page testing* and *hyperlink testing*. Page testing is to check if every page in the web application is visited at least once in some test case and hyperlink testing to check if every hyperlink from each page has been clicked at least once [16]. It would be interesting to explore the possibilities of using the *webcomps* model to develop testing criteria for page testing and hyperlink testing. The goal of using knowledge from these different areas is to come up with a sophisticated testing framework that addresses a variety of aspects of the application, both functional and non-functional.

In practical terms, future research topics may include:

- Deriving white box testing criteria from *webcomp* model.
- Using the XML test specification to include arguments for testing performance parameters like page load time.
- Automation of the XML test case generation process through an algorithm.
- Using *webcomp* model to exploit the navigational aspects of a web page to perform basis path testing using the concepts of graph theory.

6. Conclusions

Web applications need a different approach in both the modeling and the testing phases of software development. Our approach has been focused towards functionality testing of web applications, which involves the dynamic behavioral aspects of the application more than the static webpage content. The testing method is tightly bound to the requirements, thereby concentrating on immediate benchmarks rather than getting distracted by add-on features. Our approach has proposed a framework for systematic testing of web applications, aimed to add rigor to an activity that traditionally relies on the art and skills of the testers.

To summarize, our approach has the following main contributions:

a. Proposal of the *webcomp* element for UML-based specification of testing web applications. This new modeling element relies on the adapted use of standard UML extension mechanisms. With this modeling element and based on traditional navigation maps, a *webcomp model* for the application has been proposed.

b. Elaboration (mapping) of use cases to *webcomps*.

c. Adaptation of a four step process for generating test cases based on use cases. To the process described in [24] we have added the mechanism to prioritize the use cases before generating test case matrices and have suggested guidelines for creating scenarios.

d. *Combo approach* for testing consisting of *use-case driven testing* and *webcomp unit testing* and covering thoroughly the functionality of the application.

e. Development of the Web Records application, which has been used as basis for exercising our approach.

On the other hand, we are aware that there are many possible improvements for the current proposal. In particular use cases have their own limitations, which need to be addressed [28, 29]. Also, there is room for expanding the types of tests included in the *combo* strategy and there will be a need for further exercising the approach on more complex web applications. However, we believe that the suggested UML-based modeling and testing methodology provides a practical and efficient solution for web applications.

7. References

[1] H. Zhu, P. Hall, and J. May, "Software Unit Test Coverage and Adequacy," *ACM Computing Surveys,* pp. 366-427, December 1997.

[2] M. Chen and M. Kao, "Testing Object-Oriented Programs - An Integrated Approach," *Proceedings of the Tenth International Symposium on Software Reliability Engineering,* pp. 1-4, 1999.

[3] Y. Wu and J. Offutt, "Modeling and Testing Web-based Applications," GMU ISE Technical ISE- TR- 02-08, 2002.

[4] F. Ricca and P. Tonella, "Analysis and Testing of Web Applications", *Proceedings of the 23rd International Conference on Software Engineering (ICSE 2001),* pp. 25-34, 2001.

[5] S. Lee and J. Offutt, "Generating Test Cases for XML-based Web Applications," *Proceedings of 12th International Symposium on Software Reliability Engineering,* pp. 200-209, 2001.

[6] J. Conallen, *Building Web Applications with UML,* Second Edition, Addison Wesley, 2002.

[7] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley, 1998.

[8] OMG's UML Resource Page, accessed May 12, 2003, http://www.omg.org/uml.

[9] Rational UML Resource Center, accessed May 05, 2003, http://www.rational.com/uml.

[10] D. Kung, C. Liu and P. Hsia, "An Object-Oriented Web Test Model for Testing Web Applications," *Proceedings of IEEE 12th Annual International Computer Software and Application Conference,* pp. 3-5, 2000.

[11] C. Liu, D. Kung, P. Hsia and C. Hsu, "Structure Testing of Web Applications," *Proceedings of 11th Annual International Symposium on Software Reliability Engineering,* pp. 84-96, 2000.

[12] www.regonline.com accessed May 20, 2003.

[13] http://www.123signup.com accessed May 20, 2003.

[14] J. Arlow and I. Neustadt, *UML and the Unified Process:Practical Object-Oriented Analysis and Design,* Addison-Wesley, 2001.

[15] J. Conallen, "Modeling Web Application Architectures with UML," *Comm. of the ACM*, vol. 42, no. 10, pp. 63-70, 1999.

[16] F. Ricca and P. Tonella, "Testing Processes of Web Appplications," Annals of Software Engineering, vol. 14, no. 1-4, pp. 93-114, 2002.

[17] C. Atkinson, C. Bunse, H. Grob and T. Kuhne, "Towards a General Component Model for Web-Based Applications," *Annals of SoftwareEngineering*, vol. 13, no. 1-4, pp. 35-69, 2002.

[18] C. Ghezzi, M. Jazayeri and D. Mandrioli, *Fundamentals of Software Engineering*, 2nd Edition, Prentice Hall, New Jersey, 2003.

[19] R. Pressman, *Software Engineering: A Practitioner's Approach*, 4th Edition, McGrawHill, 1997.

[20] D. Wood and J. Reis, "Use Case Derived Test Cases," http://www.stickyminds.com/, accessed May 11, 2003.

[21] Ivar Jacobson, *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison- Wesley, 1994.

[22] Martin Fowler, *UML Distilled*, Addison-Wesley, 1997.

[23] M. Jarke, "Requirements Tracing," *Communications of the ACM* , vol 41, no. 12, pp. 32-36, 1998.

[24] J. Heumann, "Generating Test Cases from Use Cases," The Rational Edge, Rational Software, 2001.

[25] X. Jia and H. Liu, "Rigorous and Automatic Testing of Web Applications," *Proceedings of the 6th IASTED International Conference on Software Engineering and Applications (SEA 2002),* November, 2002.

[26] E. Hieatt and R. Mee, "Going Faster: Testing The Web Application," *IEEE Software*, vol. 19, no. 2, 2002.

[27] B. Beizer, *Software Testing Techniques*, Second Edition, International Thomson Computer Press, 1990.

[28] B. Berger, "The Dangers of Use Cases Employed as Test Cases," http://www.testassured.com/docs/Dangers.htm, accessed April 22, 2003.

[29] A. Simons, "Use Cases Considered Harmful," *Proceedings of Technology of Object-Oriented Languages and Systems,* 1999.

[30] P. Allen and J. Bambara, "Modeling Web Applications," http://www.informit.com/, accessed April 2003.