Testing web applications in practice

Javier Jesús Gutiérrez, Maria José Escalona, Manuel Mejías, Jesús Torres
Department of Computer Languages and Systems.
University of Seville.
{javierj, escalona, risoto, jtorres}@lsi.us.es

ABSTRACT

Software testing process is gaining importance at same time that size and complexity of software are growing. The specifics characteristics of web applications, like client-server architecture, heterogeneous languages and technologies or massive concurrent access, makes hard adapting classic software testing practices and strategies to web applications. This work exposes an overview of testing processes and techniques applied to web development. This work also describes a practical example testing a simple web application using only open-source tools.

1. INTRODUCTION

Internet gives to developers a new and innovative way to build software. Internet also allows the access of millions of user to a web application [4]. Thus, problems in a web application can affect to millions of users, cause many costs to business [2] and destroy a commercial image.

The Business Internet Group of San Francisco undertook a study to capture, assess and quantify the integrity of web applications on 41 specific web sites. According to the report, of those 41 sites, 28 sites contained web application failures [14]. Thus, software testing acquires a vital importance in web application development.

First web applications had a simple design based on static HTML pages. Nowadays, web applications are much more complex and interactive with dynamic information and customized user interfaces. Design, support and test modern web applications have many challenges to developers and software engineers.

This work is organized as follow. Section 1 defines some basic concepts used in this work, shows the basic aspects of a client-server web application and introduces software testing process. Section 2 describes a simple web application used as example in this work. Section 3 describes how to make unit testing over a web application. Section 4 describes how to make integration testing over a web application. Section 5 resumes conclusions and future work.

1.1. Definitions

A web page is all kind of information that can be displayed into a browser window [2]. A web page uses to be composed by HTML code, generated statically or dynamically, or by client-side executable components, like Macromedia Flash modules or Java Applets.

A web site is a group of web pages, where the information of every page is semantically related and syntactically related by links among them. User access to a web site is made by HTTP requests. Client-side user interface uses to be a browser program running over a personal computer (PC, Mac, etc.).

A web application is built applying heterogeneous technologies like client-side scripting languages included into HTML, client-side components like Java applets, server-side scripting languages like PHP or PERL, server-side components like Java Servlets, web services, databases servers, etc. All these heterogeneous technologies have to work together, obtaining a multi-user, multiplatform application.

1.2. Client-Server Architecture in web applications

Functioning of a web application is similar to classic client-server application with thin client, as showed in Figure 1. When a user writes a web address in a browser, this one acts like a client, requesting a file to a server accessible by Internet. Server processes the request and sends the file. Client, at the time to receive the file, process its content and shows them.

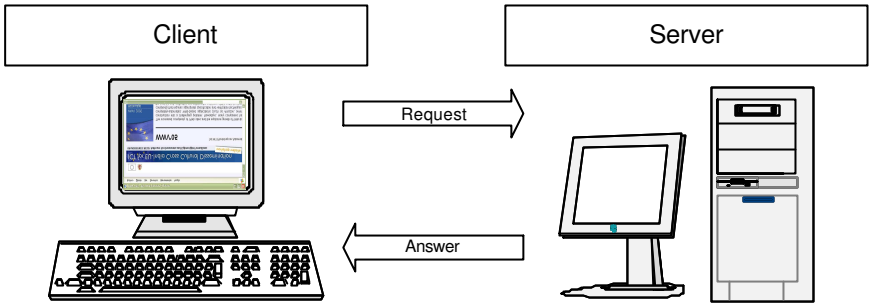

Figure 1. Static web application

First web applications were composed only by static web pages. They have not the possibility to modify their content depending by date, user, or number of requests. The user always received the same file with the same information into it. Actually, as showed in Figure 2, it is possible to build web pages dynamically, changing their information depending of many factors.

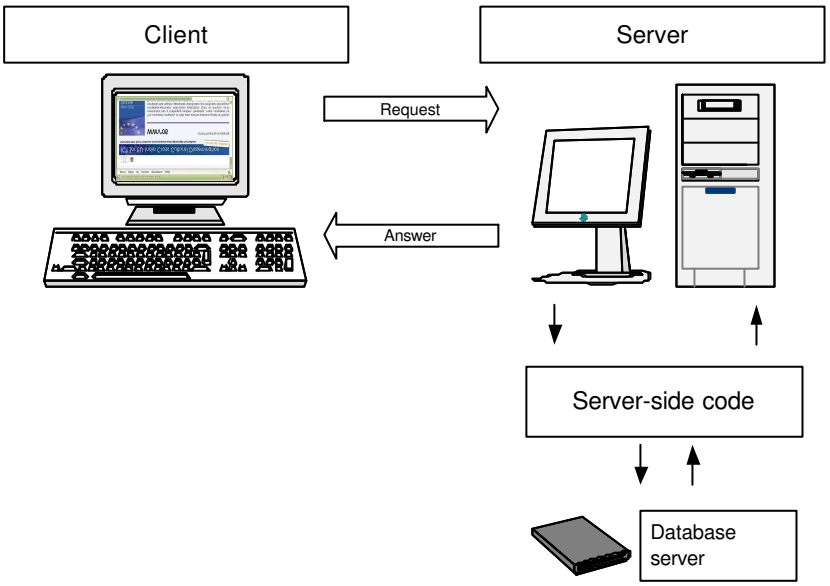

Figure 2. Dynamic web application..

Client-Server architecture in Figure 2 is the same that Client-Server architecture in Figure 1. Main different is that the server in figure 2 has two elements, one dedicated to receive requests and answer to them, and other dedicated to execute web application and generate HTML dynamically.

1.3 An overview of software testing process

Nowadays, test process has became in a vital task in development process of all kind of software systems [3]. It is needed to make a classification [1] of testing, the moment to apply them and their objectives before expose how to apply test process to web applications. Table 1 shows this classification.

Kinds of tests	Moment to apply	Description
Unit testing.	During building of software system.	Unit testing verifies design and functionality of every component of the system.
Integration testing.	During building of software system.	Integration testing verifies the union among system components through their interfaces and their functionality.
System testing.	After building of software system.	System testing verifies in depth the functionality of the system, as a black box, checking that all requirements have been implemented in the correctly.
Implantation testing.	During production environment implantation.	Implantation testing verifies the correct function of the system in the real production environment.
Acceptance testing.	After software system implantation.	Acceptance testing verifies that system has all requirements expected and satisfies the needs of the user.
Regression testing.	During modify Software system.	Regression testing verifies that changes in code do not generate unexpected errors.

Table 1. Testing classification.

Unit and integration testing verifies components of the system. System, implantation and acceptance testing verifies the entire system as a black box, from different points of view. This word is focused in unit and integration test only.

1.4. Related work

There are several works describing how to test a web application. For example, Liu [16] considers each web application component as an object and generates test cases based on data flow between those objects. Ricca [15] proposes a model based on the Unified Modeling Language (UML), to enable web application evolution analysis and test case generation. Wu [17] defines a generic analysis model that characterizes both static and dynamic aspects of web based applications. This technique is based on identifying atomic elements of dynamic web pages that have static structure and dynamic contents. Elbaum [18] explores the notion that user session data gathered as users operate web applications can be successfully employed in the testing of those applications.

This paper does not proposal a new web test model but a set of techniques to test web component. These techniques can be used in any web test model proposal to implemented test cases. These techniques can be applied to test both client-side and server-side components and they are useful to put test model in practice.

2. PRACTICAL CASE

This section describes a simple web application to insert customers in a database. In client-side, application is composed of three web pages: a form to insert customers, a web page with a message if insertion was possible and another web page with a message if insertion was not possible. In server-side, application is composed of a MySQL [8] database server and a PHP [9] insertion script. Figure 3 shows these components.

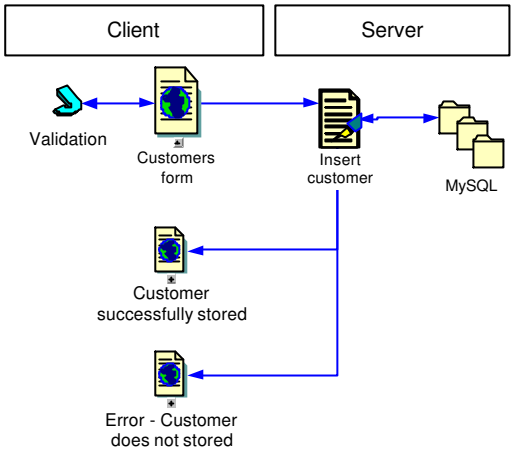

Figure 3. Insert customer web application components.

Figure 4 shows captures of HTML web pages of the application.

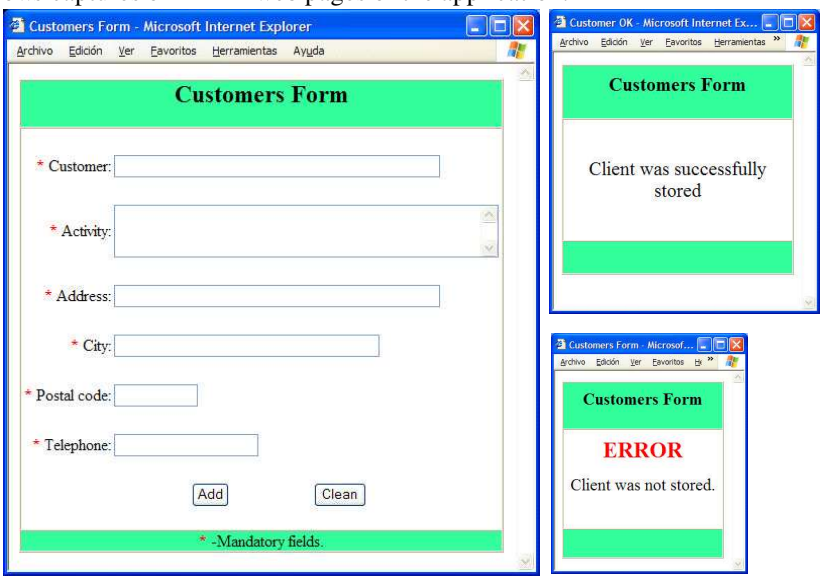

Figure 4. Insert customer form.

Validation code, written in JavaScript and included into HTML form, will verify that none obligatory field will be empty.

This application stores customers in a MySQL table. SQL code to create customers table are showed in Figure 5.

```
CREATE TABLE Customers(Id BIGINT(20)
  UNSIGNED NOT NULL AUTO_INCREMENT
  PRIMARY KEY,
Entity VARCHAR(50) NOT NUL        L,
Activity VARCHAR(250) NOT NULL,
Address VARCHAR(50) NOT NULL,
City VARCHAR(50) NOT NULL,
ZIP_Code VARCHAR(10) NOT NULL,
Telephone VARCHAR(50) NOT NULL,
Contact_person VARCHAR(50),
Contact_phone VARCHAR(10),
Observations VARCHAR(250) );
```

Figure 5. Customer SQL code.

To test this application, we will write, at first time, a set of unit tests to verify client-side components and server-side components. At second time, we will write an integration test to verify the correct working of customer form and insertion script together.

3. UNIT TESTING

Objective of unit testing is to verify the functionality of every component in isolation. To do this, we are going to divide components in two sets: client-side components (web pages, JavaScript code, etc.) and server-side components (server-scripts, databases, etc.) [1]. Each component set will have its own testing strategy and tools. Division in our example web application is showed in Figure 6.

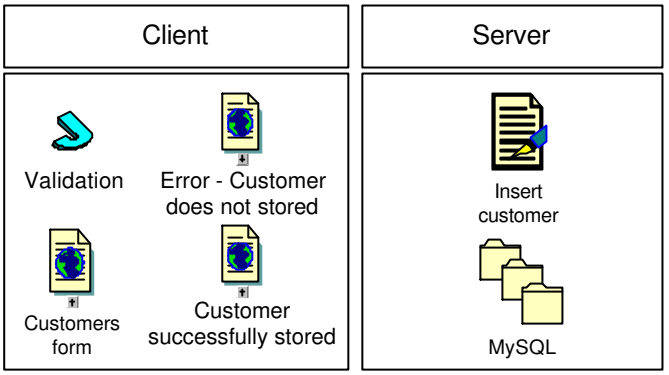

Figure 6. Client side and server side components.

Client-side components are downloaded and executed in client web browser. Server-side components are executed in server and their results are sent to client. Section 3.1 describes techniques and tools to test server-side components. Section 3.2 describes techniques and tools to test client-side components.

3.1. Server-side testing

The strategy to test server code is similar to strategy to develop unit testing in not-web applications. Main idea is to write a code fragment to execute the code under test with a set of test values and compares it result with expected results. In our sample web application we are going to write a test script in PHP.

There are many open-source tools to make easy and automatist this process. In general, all tools implements JUnit architecture [11]. A list of xUnit tools can be found in [7]. We have selected PEAR PHPUnit [13], among all PHP xUnit available.

The example web application has a function to insert a customer with one expected parameter with a table with all information about customer. This function returns a value that indicates if the client was or was not inserted. The prototype of this function is shown in Figure 7. Test case will invoke function in figure 7 with a test customer and will check result returned.

```
// $result is TRUE if customer was added into database
// and FALSE if does not.
function insertCustomer($customer)
{
      //...
      return $result;
}
```

Figure 7. InsertCustomer function prototype.

The unit test is divided in two actions. These actions are described in Table 4.

Step	Action	Verification
1	To call function “insertCustomer” with a test customer.	To verify that function result is TRUE.
2	To search test customer inserted in step 1 in customer database.	To verify that customer exists in database and its values are equals those values of test customer.

Table 4. Test steps.

The PHP scripts which implements this unit test is showed in Figure 8. First line includes PHPUnit libraries which offer functions similar to JUnit.

```
<?
  include_once('./PHPUnit/PHPUnit.php');
  include_once('./     InsertCustomerFunction       .php');

  class InsertCustomerTest extends PHPUnit_TestCase {

    var $  test_cust  omer ;
    function testInsert      Customer  () {
      $this   ->PHPUnit_TestCase("testInsertarUnCliente");
    }
    function setUp() {
      $this   ->test_customer   ['customer'] = "    test_customer    ";
      $this   ->test_customer   ['activity'] = "    test_activity    ";
      $this   ->test_customer   ['address'] = "    test_address    ";
      $this   ->test_customer   ['city '] = "  test_city   ";
      $this   ->test_customer   ['ZIP_Code  '] = "00000";
        $this ->test_customer   ['telephone  '] = "000  -00-00-00";
    }
    function testInsert      ACustomer  () {
      $  result = insert  Customer  ($this ->test_customer    );
      $this   ->assertTrue($result);
    }
    function test    CustomerInserted    (){
      $conn = mysql_connect("localhost", "", "")
      mysql_select_db("C       ustomer  ")
      $sql = "SELECT * FROM       Customer  WHERE  'custome  r'='".
            $this    ->test_customer   ['customer'   ]."'";
        $result  = mysql_query($sql)
      $  Customer   = mysql_fetch_array($resul) ;
      $this   ->assertEquals($this     ->test_customer   ['customer'   ],
                        $       Customer  ['customer'   ], " Different   customers   .");
      //   ...

      mysql_free_result($result);
    }
}
  echo "<HTML> <BODY> <HR> <BR>          Insert Customer Test      . <BR>";
  $suite  = new PHPUnit_TestSuite("InsertCustomerTest ");
  $result = PHPUnit::run($suite);
  echo $result      -> toStrin  g();
  echo "<HR>";
?>
```

Figure 8. InsertCustomer function unit test.

If “insertCustomer” function has no error, test script will write an output like figure 9. That output indicates that test was success.

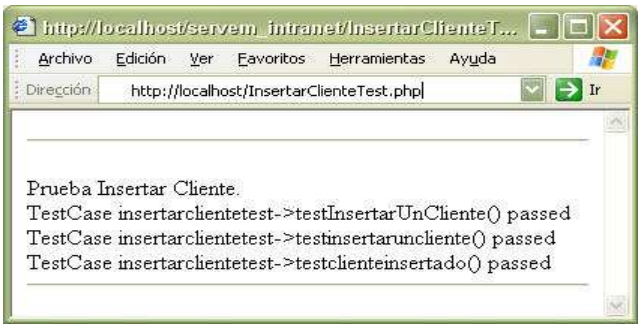

Figure 9. Successfully test.

3.2. Client-side testing

Objectives of dient-side components testing are to verify that HTML is correct and compliments standards [10] and to verify dynamic components into web pages. For example, we will verify that HTML satisfied HTML 4.01 Transitional standard and the JavaScript validation code in our example web application. It is important to fulfil HTML standards to guarantee that a web page is correctly visualized in different browsers.

Other unit test that can apply to client-side components are to verify that web pages are correctly visualized in different browsers, verify user interface usability, etc.

3.2.1. HTML web pages testing

A HTML web page contains the information that will be displayed and a set of tags that indicates how that information has to be displayed. Thus, we have to test that every web page in our web application example satisfied HTML standards proposed by W3C consortium. A HTML validation tools is available at W3C consortium web site [6]. We have used that tool to verify our pages. Results are resumed in Figure 10.

Validation form has an option to upload a web page to validate. In our example web application, when customer form is validated, some errors appeared:

Line 107, column 38: document type does not allow element "BODY" here
<body bgcolor="#FFFFFF" text="#000000" >
Line 108, column 75: there is no attribute "BORDERCOLOR"
... cellpadding="0" align="left" bordercolor= "#0066FF">

First error is because of before <body> tag must be </head> tag.
"Bordercolor" attrib is obsolete and does not complaint version 4.0 of HTML specification [10]. Style sheets must be used to define color.
Once corrected two errors, validation tool shows next message: "This Page Is Valid HTML 4.01 Transitional!"
The other two web pages have no errors.

Figure 10. Insert customers HTML form test.

To avoid testing of each web page in isolation and uploading pages one by one, we can use an option to write and URL and let application to verify all HTML pages in that URL. There are also, applications that connect with W3C web site.

3.2.2. Testing JavaScript code

The form to insert customers includes JavaScript code to avoid blank fields. It is necessary to test this code to verify that its functionality is the expected one and it is able to detect all invalid combinations possible.

Originally, JavaScript code was included into HTML web form. This one makes hard to test it. So, JavaScript code was refactorized and moved into its own script file called “validate.js”. Illustration 11 shows a fragment of validation code.

```
function Validador(      CustomerForm    ) {

    if ( CustomerForm    .customer   .value == "") {
        alert(" Field   \"Customer  \" is obligatory     .");
        CustomerForm    .customer   .focus();
        return (false);
    }

    // ....

    return (true);
}
```

Ilustración 11.Validate.js fragment code.

Code showed in Illustration 11 notices if an obligatory field is empty. Code displays a message window, showed in Figure 12 and Figure 13, and sets form focus in empty field.

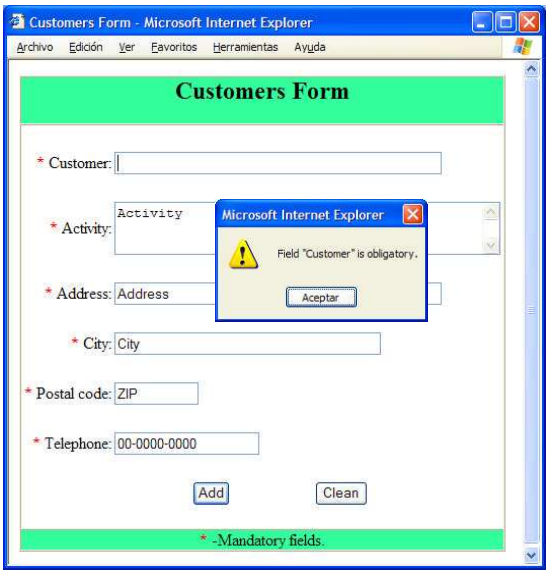

Figure 12. An error in customers form.

Figure 13. Alert window detail.

The first strategy to validate JavaScript code is to apply manual testing. Manual testing can be performed by a worker who will write all combinations possible and who will verifies that JavaScript code results are the expected results. This is the simplest solution, but worse too. Manual testing requires dedication in exclusive of one programmer, allows many errors because it is a very repetitive and bored task, and it is needed to make again all test every time customers form or JavaScript code changes.

Another strategy is to write a unit test case to test JavaScript code. Unit test case has to verify that functionality of “validate.js” is the expected functionality when applying a set of values, in the same way

that test written in section 3.1. To test "validate.js" in isolation, it is needed to write an object that represents form. A fragment of that code is showed in Figure 14.

```
function customer (value ) {
    this.value= value ;
    return(this);
}

function CustomerForm (e, a, d, l, c, t) {
     this. customer =new customer (e);
     this. activity =new activity (a);
     this. address =new address (d);
     this. city =new city (l);
     this.postal _code =new postal _code (c);
     this. telephone =new telephone (t);
     return (this);
}
```

Figure 14. JavaScript object representing customer form.

A test to verify validation code using the representation of the form is showed in Figure 15.

```
<SCRIPT LANGUAGE="JavaScript" SRC="validar.js"></SCRIPT>
<SCRIPT LANGUAGE="JavaScript" SRC="          customerForm    .js"></SCRIPT>
<SCRIPT LANGUAGE="JavaScript">
  var form= new      CustomerForm  ("","a", "d", "l", "c", "t");
  if (Validador(form) == false) {
    alert("   Test failed   .");
  } else {
    alert("   Test valid   .");
  }
</SCRIPT>
```

Figure 15. Test of form customer JavaScript code.

It will be possible to write similar tests based on test described in Figure 15, changing the position of blank field in customer form creation. Test in Figure 15 verifies that function return expected value. However, this strategy is not good because. Test does not verify that alert message has expected text or the form focus will be in the empty field. Even more, test stops after executing "alert" instruction, so manual verification is still needed. Another problem is that test has to be loaded into the browser to be executed. This strategy has no advantages from manual test strategy. We still need a programmer who changes the code, execute the test and verifies that alert message is expected message.

Other strategies to improve JavaScript testing are to use a JavaScript interpreter instead executing test in a web browser or to use a mouse-record tool. Improvement of JavaScript testability is a future work.

This same process can also be done to verify other client-side scripts, like VBScript.

4. INTEGRATION TESTING

Once verified each components of the system with unit tests, it is time to verify that those components are able to work right among them. This one is the goal of integration tests.

In a web application, components in client-side communicate with components in server-side using HTTP protocol, instead of classic messages to methods or functions. This allows a very low coupling but makes useless classic xUnit tools to develop integration tests.

There are two main techniques to develop integration testing in a web application to develop tests that operates application through it HTML web interface. One technique is using stored macros, and replaying them to make a test. Macros have to be record again when interface or application changes, so they are not the best option. Another technique is using an API to analyze HTTP communication. This API allows to write a program that sends HTTP requests and receives and analyzes their answers. This second technique is more flexible, and allow to test in depth web pages, but they spend more development time. We are going to write test based on API to test our web example application.

There are many open-source tools that offer APIs to write integration tools. We have chosen HttpUnit [6] to write an example. HttpUnit it is written in Java, but we will show that it is possible to use it to test a PHP web application with a HTML user interface.

4.1. Writing an integration test

HttpUnit can request a web page in the same way than a web client. HttpUnit offers an interface to ask if received page includes certain elements and to interact with the elements of the page, by example navigating through a link. This tool offers almost all functionality of a web browser, like cookies control, header analysis, GET and POSTS, etc.

In this point, we are going to write a test with HttpUnit. This test will be a Java class that request customer form and, later, verifies that the web page received has expected elements.

Test goes to verify:

1. Connection with server and customer form requesting.
2. Verify title of received page to test if it is the expected web page.
3. Verify if web received page contains customer form.
4. Verify if form includes all expected fields and assign them test values.
5. Verify if, when pressing add button, web server answers with expected page.

Figure 16 shows java code of this test.

```
import net.sourceforge.jwebunit.WebTestCase;
import com.meterware.httpunit.*;
import com.meterware.servletunit.*;
import java.util.*;
import junit.framework.*;

public class Test CustomerForm extends TestCase {
    public Test CustomerForm () {
        super(" Test CustomerForm ");
    }
    public void testInsert Customer ()
      throws Exception
    {
      WebConversation wc = new WebConversation();
      WebResponse resp = wc.getR esponse("http://localhost/CustomForm.htm");
     Assert.assertEquals(resp.getTitle().compareTo(" Customer Form "), 0);
      WebForm form = resp.getFormWithName(" CustomerForm ");
      Assert.assertNotNull(form);
      form.setParameter(" customer ", " test_cus tomer ");
        form.setParameter(" activity ", " test_activity ");
      form.setParameter(" address ", " test_address ");
      form.setParameter(" city ", " test_city ");
      form.setParameter("postal _code ", "00000");
        form.setParameter("tele phone ", "00 -00-00");
      WebRequest req = form.getRequest("Submit");
      resp = wc.getResponse(req);
      String output = resp.getText();
      Assert.assertEquals( output.indexOf("Error"), -1 );
    }
}
```

Figure 16. Customer form integration test.

It will be possible to add an additional validation to verify that testing customer inserted is really stored into database. This validation can check that insertion script works right when it is called from HTML form, not only when it is directly called.

5. CONCLUSIONS

This work shows how it is possible to build a web application and apply it different testing process with open-source tools only. All techniques exposed in this work can be easily applied to other web development platforms like ASP.NET or JSP and Servlets. Some of tests showed in this work are development with Java over a PHP web application, which demonstrates the interoperability among technologies and languages that web engineering offers.

All tools used in this work are free for use and download through Internet and, also, their source code is accessible, even in commercials tools like MySQL (although free for use depends of license of application).

We have seen with our example web application, that it is very important to separate different components of a web application, presentation components like HTML and code components like PHP scripts, to facility testing process. A good separation among components improves development process, testing process and maintainability.

Future lines of investigation from this work are to investigate new tools and strategies to test web user interfaces built with HTML and JavaScript, study with examples the application of this techniques and practices in other development platforms like .NET and other components, like Flash user interfaces, and study strategies to test HTML dynamically generated.

REFERENCES

[1] Ash, L. 2003. *The Web Testing Companion: The Insider's Guide to Efficient and Effective Tests*. John Wiley & Sons, Hoboken, USA.

[2] Ye Wu, Jeff Offutt, Xiaochen Du. 2004. *Modeling and Testing of Dynamic Aspects of Web Applicationsy*. Submitted for journal publication

[3] M.J. Escalona, M. Mejías, J.J. Gutiérrez, J. Torres. 2004. *Métodos de Testing Sobre La Ingeniería De Requisitos Web de NDT*. IADIS WWW/Internet 2.004. 353-360.

[4] Jeff Offutt et-al. 2004. *Web Application Bypass Testing*. ISSRE '04

[5] Métrica v3. http://www.csi.map.es/csi/metrica3/

[6] HttpUnit. http://httpunit.sourceforge.net/

[7] Herramientas xUnit. http://www.xprogramming.com/software.htm

[8] MySQL. http://www.mysql.com

[9] PHP. http://www.php.net/

[10] HTML 4.01 Specification. http://www.w3.org/TR/html4/

[11] Junit. http://junit.org/

[12]W3C HTML Validator. http://validator.w3.org/

[13] PEAR PHPUnit. http://pear.php.net/package/PHPUnit/

[14] BIGSF. Government Web Application Integrity. The Business Internet Group of San Francisco, 2003.

[15] F. Ricca, P. Tonella. Analysis and testing of web applications. In Proceedings of the 23rd international conference on Software Engineering. IEEE Computer Society Press, 2001.

[16] S. Chun, J. Outt. Generating test cases for XML-based web application. In Proceedings of the 12th International Symposium on Software Reliability Engineering, pages 200–209, Hong Kong, November 2001.

[17] Y. Wu, D. Pan, M.H. Chen. Techniques for testing component-based software. In Proceedings of the 7th IEEE International Conference on Engineering of Complex Computer Systems, pages 15–23, Skövde, Sweden, June 2001.

[18] M. J. Harrold, M. L. Soa. Selecting data flow integration testing. IEEE Software, 8(2):58–65, March 1991.