# A Practical Approach of Web System Testing

Javier Jesús Gutierrez, María José Escalona, Manuel Mejías and Jesús Torres

Department of Computer Languages and Systems. University of Seville, Spain. (javierj, escalona, risoto, jtorres)@lsi.us.es

## Introduction

The process of testing software system is gaining more importance every day [6]. Software applications are growing in size and complexity quickly. It makes more necessary to dispose techniques to assure quality of the systems and that the result satisfied initial specifications [1].

Assure the quality of the system is very important in web engineering. First web systems, at the beginning of nineties, had a simple design based on static HTML pages. Nowadays, web systems are 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, in order to obtain a multi-user and multi-platform application. The design, maintenance and test of the modern web applications have many challenges to developers and software engineers [7][13].

Internet and web systems also bring to developers a new and innovative way to build software. Internet allows millions of users to access to an application [7]. Thus, problems in a web application can affect to millions of users, cause many costs to the business [13] and destroy a commercial image. For all these reasons, quality assurance and software testing acquire a vital importance in the web system development.

This work introduces theory and practice about the generation and implementation of system test cases in web applications. This work presents a complete vision of system testing showing how to put in practice the ideas exposed. Section 2 defines the process of software testing and studies in depth system testing process and how it can be applied to web development. Section 3 shows a practical case of generation and implementation of system test cases. Finally, section 4 resumes conclusions and future works.

## System Testing over Web Systems

This section describes the process of software testing, studies the process of system testing in depth, from the point of view of web engineering and, finally, describes briefly a proposal to generate system test cases in a systematic way. This approach will be applied in the practical case described in section 3.

### An Overview of Software Testing Process

The process of software testing cannot be done at the end of the construction of the system, neither expecting to test the whole system ant one time [2]. The process of software testing has to be divided into sub-process. Every sub-process indicates which elements must be tested and the moment to perform every test. The sub-process might be applied when first elements under test are available. One possible division in sub-process widely accepted is showed in figure 1 [9].

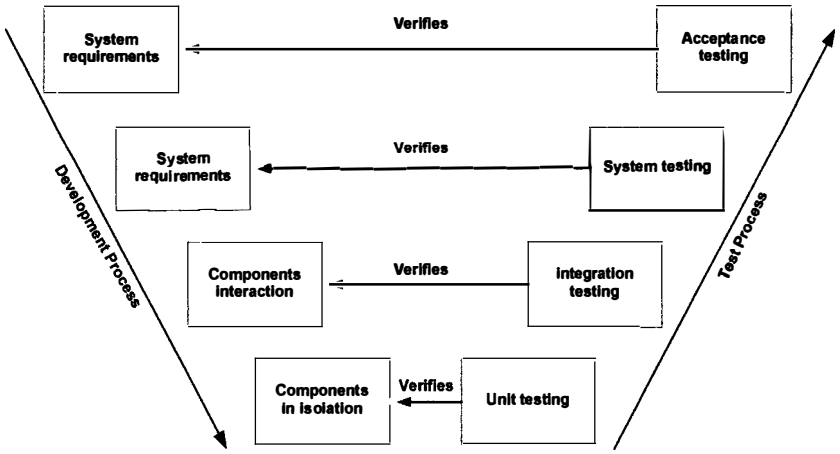


**Fig. 1.** Software testing process

Unit testing is done during building of software system [8]. Their objectives are the verification of the design and the functionality of every component in the system. Integration testing is done during the building of software system. Their objectives are the verification of the union among system components through their interfaces and their functionality. System testing is done after building of software system. System testing answer

the question: is the entire system working to deliver the user goals? [17]. Acceptance testing is done after software system implantation. Their objectives are the verification that system covers all requirements expected and satisfies the needs of the users.

In figure 1 another kind of testing, regression testing, is lost. It was not included because they are not performed during the development of the system. Regression testing are applied during the maintenance of the system to accurate that the changes does not introduces unexpected errors in unmodified elements.

Testing process described in figure 1 might be applied to all types of software system: desktop applications, client-server applications, mobile applications, etc. In the specific case of web systems, it is needed to arrange specific techniques, like separate client-side and server-side components, and specific tools, like HTML validation tool, to web applications that allows to apply this process.

Next section describes in depth the process of the software system testing and how it must be applied in web projects.

## System Test Process

Unit and integration testing guarantees an error-free code, or, at least, that the code has not the most important or common errors, due of the fact that it is impossible to test the whole code in depth [2]. However, unit and integration testing are not enough to assure the quality of the system. It is possible to have an error-free code that does not satisfy the expectative or the needed of the system final users. This one makes necessary a system test phase. This phase will be performed after unit and integration test phases, and when first system requirements are completely implemented.

An important rule that can be applied in all phases of software testing process is that testing must begin as soon as possible [12]. Due the cost of time and resources needed to correct an error increases at same time than time between the apparition and the detection of that error, it is vital to detect all errors as soon as possible. In system test cases, system is verified like a black box. Thus, system tester have to wait until the system is built, or, at least, until some requirements are fully implemented. An implemented requirement means that all elements needed to perform that requirements from the user point of view, like user interfaces, persistent layers, databases, etc, are implemented. However it is possible to advance definition and design of system test cases to early development phases.

In [4] a group of representative proposals of early testing are described, analyzed and compared.

The process of generation of system test cases from functional requirements consists in build a use model of the system from its requirements, and, later, to generate a set of input values, a set of events or interactions among system and users or actors, and the expected results [5]. The evolution of methodological proposals to drive this process has been focused in how to build and represent the use model.

There are works, like [11], that propose new kinds of requirements specific for web development, like actors' requirements, adaptability requirements or navigational requirements. However, functional requirements are still playing a very important role in web systems, but must be complemented with other types of requirements like navigational or information requirements.

Functional requirements should be independent of the implemented platform or the architecture of the system in early development phases. Thus functional requirements must not include any reference to any platform, like web or standalone platform. Due this fact, methodologies to generate system test cases from requirements are also applied in web systems. An example of integration of a process to generate system cases into a web development methodology can be found in [9]. Test cases are, also, independence of the complexity of web interfaces.

Next section describes a possible approach to generate automatically system test cases. This process is applied in the practical example in section 3.

## A Proposal to Generate System Test Cases from Requirements

Descriptions of functional requirements are the main artifact to the design of system test cases [5]. Nowadays, there are a big number of proposals to systematize this process. A complete survey about these proposals is presented in [4].

This section introduces briefly the fundaments of a systematic proposal to generation of system test cases from use cases. This proposal has been development from the conclusions of comparative studios and analyzes of several existing proposals, like [4]. The generation of test cases is showed in figure 2 with an activity diagram [14]. It starts with the enumeration and the description of the **observable results**. An observable result is anything that can be automatically or manual checked, like a system screen, a new stored, modified or deleted record, a received message by other computer or server, etc. In conclusion,

In second step, all possible **execution paths** are identified. An execution path is a description of the interactions among system and one or more actors to obtain one of the results identified in previous activity.

After identifying all the execution paths, all needed values for each path must be identified. A valid value is a data or precondition that applied to an execution path allows obtaining the expected result.

In the last step, all **redundant execution paths** must be removed. An execution path is redundant when there is other equal path or when it is included into other path, with the same values, and the same results are obtained.

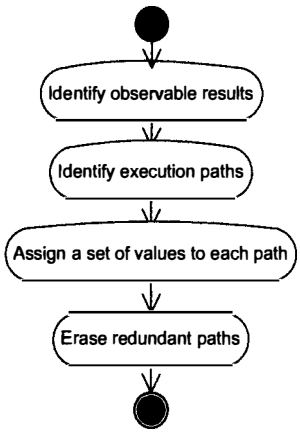


**Fig. 2.** Activity diagram to generate system test cases

This proposal can be applied in parallel with identification and definition of requirements. In the next section, we are going to put in practise this proposal in order to generate a set of system test cases for a simple example.

## A Case Study

In order to explain in a detail way the approach presented in the previous section, in this section we are going to apply it in a real project.

## System Description

This section describes a simplified web application that we use like a simple example in next sections to apply the presented approach. The objective of this web system is to manage the information about customers in a business. Concretely, the selected example is the functional requirement: "System must allow that one registered user inserts new customers into customers database". It indicates that the system has to allow to add new costumers.

This requirement is too basic and ambiguous to be directly implemented. Thus, the requirement has been refined and splitters in two use cases. The first use case describes the process to access into the system and the second one offers the way to insert a new customer. Both use cases are graphically showed in the UML use case diagram [14] in figure 3 and described textually in table 2 and 3.

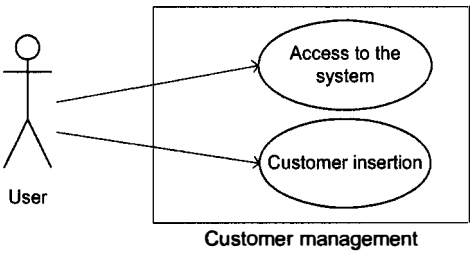


**Fig. 3.** Use cases to insert new customers

In order to describe deeply each requirements. We will use a patterns, that is a special template, described by the proposals NDT (Navigational Development Techniques) [11]. NDT is a methodological proposal to drive the requirements and analysis phases in a web system development. NDT also includes a support tool called NDT-Tool [10]. For instance, the pattern for the firs use cases is presented in table 2.

Using a similar pattern for the other use case, the requirements can be implemented in a web application. In our wor, Technologies used in the implementation were HTML and JavaScript, to define the user interfaces, and PHP to define the business logic and data access. This web application is composed of two main forms: the first one controls the access to the system, and the other one inserts new customers. Figures 4 shows the first one.

**Table 1.** Textual description for the use case "Access to the system"

| FR-01 | Access to the system | |
|---|---|---|
| Description | The system has to manage the access to the system and verify the identity of each doctor. For that, it has to play like it is describe in this use case. | |
| Normal execution | Step | Action |
| | 1 | An user tries to access to the system |
| | 2 | System asks for identifier and password. |
| | 3 | User gives the system this information. |
| | 4 | If this information is correct, the system allows the user the access, and its continue with use case FR-02. |
| Post-condition | None | |
| Exceptions | Step | Action |
| | 3 | If identifies does not exist and the number of attempts is less than 3, system shows a message telling that user name does not exists and asks it again. |
| | 3 | If password does not match with the user password and the number of attempts is less than 3, system shows a message telling that password is invalid and asks it again. |
| | 3 | If identifier does not exist or password does not match and the number of attempts is 3 or bigger, system shows a message and denied access to the system. |

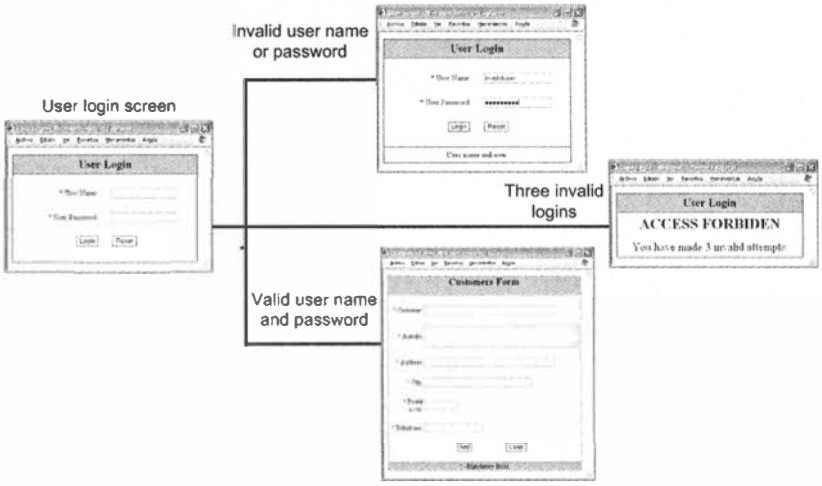


**Fig. 4.** Access to system screens

Unit and integration testing have already been successfully performed. Next section shows how to generate a set of test cases from the requirements of this application.

## System Test Cases Generation

Previous section has showed a proposal to generate system test case from functional requirements expressed like use cases. In this section, that proposal will be applied to generate test cases from use cases described. These test cases will verify the success implementation of the requirement described in table 1 into the web application.

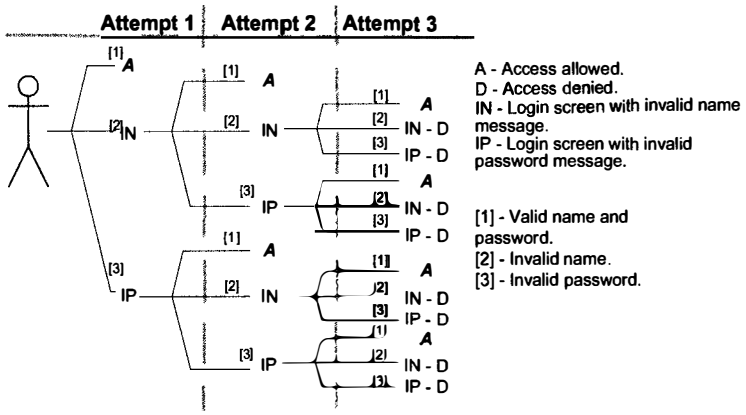


**Fig. 5.** Execution paths from "Access to system" use case

The simplest way to find execution paths is to explore all combinations among steps described in a use case. Figures 5, for instance, shows execution paths that cover all combinations of steps. They include the normal execution sequence and the alternative execution sequences. From the "Access to system" use case, twenty-one execution paths are generated.

It is more difficult to study the number of possible combinations in the "Insert customer" use case. In theory, an infinite number of executions paths are possible, for instance, "do not writing a mandatory field". In our example, we are going to suppose that this scenario can appear just one time, in order words, a mandatory field could be empty just the first time. Applying this supposition, from the "Insert customer" use case, , seven execution paths are generated.

Calculating all possible combinations between both use cases, 21 x 7 = 147 execution paths are generated. This number is too high in order to implement one test from each possible execution path. We will choose a representative subset of paths only to be implemented as system test cases. It is out of the scope of this work to show the criterions to select the adequate paths to be implemented as test cases. Several algorithms for select paths can be found in [15] and [16]. Some heuristics and guides can be applied in order to sure the quality of the election. Next paragraphs describe the chosen paths.

The objective of system test cases is to verify the success insertion of a new customer. The only way to access to insertion customer form is from user login screen. So, the entire test includes a valid login, in other words, a success path through the use case in table 2.

**Table 3.** Executions paths

| | Execution path description | Observable result. |
|---|---|---|
| 1 | Write a valid name and password.<br>Write a valid customer. | Customer inserted screen. |
| 2 | Write a valid name and password.<br>Write a valid customer without a mandatory field.<br>Write the mandatory field. | Customer inserted screen. |
| 3 | Write a valid name and password.<br>Write an existing customer. | Customer error screen. |

**Table 4.** Test values

| Path | Values set | Concrete values. |
|---|---|---|
| 1 2, 3 | Valid login. | Name: validname<br>Password: validpassword. |
| 1, 3 | Valid customer. | Customer : customer_name<br>Activity : customer_activity<br>Address : customer_address<br>City : customer_city<br>Postal code : customer_postalcode<br>Telephone : customer_telephone |
| 2 | Valid customer without a mandatory field. | Customer : customer_name<br>Activity : customer_activity<br>Address : <empty><br>City : customer_city<br>Postal code : customer_postalcode<br>Telephone : customer_telephone |

Database server is an external component to web system, but it is also need to perform the requirement. Thus, to accurate that any error is provoked just only into the system and not into external components, all test cases assume that database server components is always running and always process successfully the operations requested.

Execution paths selected to be implemented as system test cases are showed and described in table 3 and table 4.

To study in depth all possible combinations of every use case separately from other uses cases is useful in order to verify the implementation of every use case in isolation. However, at the time to verify a requirement composed of several use cases, it is better to choose a subset that verifies the whole functionality of that requirement.

### *Test Case Implementation*

This section shows how to implement the test generated in section 3.2. to be executed over the example web application. We have searched for an open-source tool that facilities the implementation task. The characteristics searched in the tool were: possibility to describe the operations performed in every test case and possibility to compare the results with the expected results.

There are two kind of testing tools. Tools that record and replay a sequence of actions and tools that offers an API to write code that simulates a user interaction.

API tools are more flexible, minimize the maintenance of test cases and allow easily testing web system that returns big, dynamic or complex web pages. We have chosen HttpUnit [3] from all available tools. It is out of the scope of this work to explain how to implement a test with HttpUnit.

The code implements the first execution path in table 4 will have the next steps: 1. Connect to web server, 2. Ask for login page. 3. Verify that received page is login page. 4. Write a valid user and password and press submit button. 5. Verify that received page is customer form. 6. Write a new valid customer and press submit button. 7. Verify that received page is ok page.

## Conclusions

This work has described the process of software testing, applying over web applications. This work has also showed the needs that become necessary to perform system testing. A methodological proposal to generate system test cases from functional requirements has been briefly described.

System test cases are written from functional requirements. Thus, system test cases are independent of the type of system or architecture under development. All proposals to generate system test cases from requirements might be applied to web systems. Design of software testing can also start as soon as first requirements are available. Planning and design of software test cases in early development phases performs and additional validation over the system requirements, allowing to detect errors, omissions, incongruences and even overspecification or, in other words, too much requirements. Correcting errors detected in early development phases are easy and economic because the cost of correct errors increases in the same way that time between apparition and detecting increases [4].

All the ideas exposed in first sections of this work, have been applied in a practical case of study. In this case of study, a set of system test cases have been generated and implemented from a real web application. This example also shows how is possible to use different technologies and languages in the development of web applications without integration problems. Concretely, languages used in web application example have been: JavaScript, HTML and PHP, and language of the tool to test the application has been Java.

System testing can be completed with another types of tests, like performance, reliability [17] and navigability [18] tests.

An investigation line open is to study the integration of the generation of system test cases into a web development process. First ideas can be found in [9]. Another line of investigation is to automate the generation process and integrate it into a CASE tool like [10]. Construction of web system involves many types of non-functional requirements very important, like navigation requirements [11]. Another line of investigation open is to develop another generation process to be applied over non-functional requirements.

## References


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

[2] Binder Rober V (1999) Testing Object-Oriented Systems. Addison Wesley

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

[4] Gutiérrez, JJ, Escalona MJ et-al. (2004) Comparative Analysis Of Methodological Proposes to Systematic Generation Of System Test Cases From System Requisites. SE'04 Workshop, Paris France

[5] Jacobs F (2004) Automatic Generation of Test Cases From Use Cases. ICSTEST'04. Bilbao Spain

[6] Pankaj J (2002) Software Project Management in Practice. Addison Wesley USA
[7] Offutt J et-al. (2004) Web Application Bypass Testing. 15th IEEE International Symposium on Software Reliability Engineering ISSRE. Saint-Malo France
[8] Link J, Frohlich P (2003) Testing in Java: How Tests Drive the Code. Morgan Kaufmann Publishers USA
[9] Escalona MJ et-al. (2004) Testing Methods Applied In Web Requirement Engineering with NDT. IADIS WWW/Internet 2.004. 353-360
[10] Escalona MJ et-al. (2003) NDT-Tool: A Case Tool to Deal with Requirements in Web Information Systems. ICWE'03. Oviedo Spain
[11] Escalona MJ (2004) Models and Techniques to Specify and Analyze Navigation in Software Systems. Ph. European Thesis. Department of Computer Languages and Systems. University of Seville. Seville Spain www.lsi.us.es/~escalona/files/reports /tesis. rar
[12] Magro B, Garbajosa J et-al. (2004) Automated Support for Requirements and Validation Tests as Development Drivers. Proc of the 3rd workshop on System Testing and Validation. pp 9-18. Paris France
[13] Wu Y, Offutt J, Du X (2004) Modelling and Testing of Dynamic Aspects of Web Applications. Submitted for journal publication
[14] 2003. OMG Unified Modelling Language Specification 2.0
[15] Nebut C et-al. (2003) Requirements by Contract Allow Automated System Testing. Proc of the 14th International symposium of Software Reliability Engineering (ISSRE'03). Denver Colorado EEUU
[16] Nebut C et-al. (2004) A Requirement-Based Approach to Test Product Families. LNCS. pp 198-210
[17] Cohen F (2004) Java Testing and Design. Prentice Hall USA
[18] Ricca F, Tonella P (2001) Analysis and Testing of Web Applications. 23rd International Conference on Software Engineering. Toronto Canada