Internet Application Testing

Prijic, Aleksandar; Jovic, Darko; and Milutinovic, Veljko

Abstract — ***In the introductory part, this paper defines the general environment for this research and defines the terms of interest. Then we define the research problem: How to find a proper testing solution that will guarantee debugging errors, which will give us reliable software product; this problem is important because it enables us to recognize some errors, which we are not conscious of. Existing solutions to the problem are briefly surveyed in the third part, and their drawbacks are underlined; each surveyed piece of research is analysed according to the same template. Then, the essence of the proposed solution is presented: As our proposal, we have developed a specific test framework, which as a result, brings significant improvement and speedup of the process of online game system functional testing. The main idea is to run tests from test environment JUnit. It is explained, on a concrete situation, which tests to perform, so we can be sure that the system works properly. The conclusion is from the point of view of performance/complexity ratio.***

1. Introduction

BEFORE we start to discuss about concrete situations, as a primary thing, we have to define the term *testing*, itself. Definitions which most precisely describe this term are the following: "… process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component..." [1], "... involves any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results. " [2], "... process of executing a program or system with the intent of finding errors." [3] Considering that, testing does not provide the software without errors; the main goal of the testing is to locate them. On the other hand, it has to secure that the system works properly, even if there are errors. It is necessary that the person that executes testing properly simulates interaction between software and its environment, which includes identification and simulation of the interface used by the system and all the inputs that can appear.

Manuscript received January 30, 2006. This work was supported in part by the Finsoft Company, London, United Kingdom.

Prijic, Aleksandar, Jovic, Darko, and Milutinovic, Veljko are with the Faculty of Electrical Engineering, University of Belgrade, Serbia.

In the case of the internet applications, the first step is to recognize the iterative nature of the cycle, which would make the testing itself a lot easier. However, this usually brings changing demands during process, resulting in many problems (e.g. testing functionalities that have been changed or that do not exist at all). „The most dangerous errors that can appear are systematic. They are caused because of wrong assumptions that author can predict." [4]. Also, one of the main factors for successful testing is knowing internet explorer, operating systems, data bases, script languages, web servers, and transaction protocols. As it is already mentioned, it is required to set the appropriate environment, so this system will look alike the environment in which the application really works. [2] The most crucial factor is good planning. First it has to be made the detailed plan, which has to be broken in the functional components. After this step, these components are being prioritized and tested according its priorities.

2. Problem Statement

At this very moment, Internet is having its explosion, and the process of testing has to follow it up. One solution for this problem is to increase number of people which are executing testing, while the other is to make that process an automatic. This solution is becoming the strategic need and the critical factor for the process of software evolution. During the test development strategy, it is necessary to minimize the influence of changes in applications and, also, in the tools used for testing. It is necessary to implement the unique test environment, which will improve and develop itself according each new application applied (e.g. acclimatization for different versions of software).

So we can even think about unique test environment, we have to consider a few facts: [3]

-Automation has to be the central task, not the secondary problem. Test design environment and writing code require plenty of time and effort. This is not a project that someone can do it in his free time. This implies a complete software project, which has to be documented, inspected and tested.

-Test environment and test design are separate parts. Test environment is executable environment for automatic tests, while test design shows how to test the particular function or characteristic of the application.

-Test environment has to be separated from the application. Although applications are quite simple, components of which they consist of are

not, and that is the reason why environment for test automation has to satisfy these components. Using this fact, all critical components of the application are separated from the application itself, so the environment can be used for other applications as well as for that one. Data which are characteristic for the application itself are being sent to test environment using variables.

-Test environment has to be easy for evolution, as well as for maintenance. System should be modular; each part should be independent, so it can be deployed particularly, which results in simple maintenance. Also, system should be properly documented and defined (standards and templates for documentation).

-Test strategy vocabulary and test design should not depend on test environment. Test strategy should define low level vocabulary, which will be used for all applications and should be independent from test environment. On the other hand, test design for a certain application should define high level vocabulary, which is specific only for that application and, also, independent from test environment. This fact should improve manually tests, as well as automatic. Vocabulary should be understandable for an ordinary person.

-Test strategy and test design should hide its environment complicacy from the persons which execute testing.

Having these facts on our mind, we tried to evolve the optimal solution.

3. EXISTING SOLUTION AND THEIR CRITICISM

There are a great number of commercially available environments for test automation, which are based on key words (*commercial keyword driven frameworks*). They are mostly used as bridge between applications, which are being tested, and tools for the automatic tests, which the company is developing. These kinds of products require some modifications so they can respond to the test demands. Some of the known tools are:

Puffin Automated Framework – environment for test automation, completely implemented on the programme language Phyton. This project has open code. At the beginning it was used for testing the internet applications, but it was timely evolved, so now it can also be used for supervising the system and for automatic data transmission. [6] Whole environment is based on four types of objects:

- Action – executable object, which can receive one or more inputs and product one or more outputs. Its results can be checked according to the expected standards of the certain type.
- Action token – represents the value used as an input for some action, or the value generated as an output after executing an action. These objects can be associated with data system or data base. They can be used for execution control or condition.
- Task – group of actions which all have to manage completely so the task succeeds. It can be arranged that some task can be executed only if the other task is completely performed, or that failure of the task determines the whole project.
- Plan – structured execution scheme, which consists of set of tasks. In the plan it can be defined which tasks execute and do they execute depending of the certain condition or until the task is complete.

Apache JMeter – tool designed for functional testing of applications and for measuring performances. [7, 8] It is completely implemented in Java. It was first designed for testing internet applications, but timely spread with other functions. This tool can test performance of static and dynamic resources. It can be used to simulate server, network or object load. Using this property, we can test persistence of the system, or analyze performances in the different environments. The interface consists of two elements:

- TestPlan – contains tests which have to be executed.
- WorkBench – implements the work space where the tests are being created. This part can be broken in a as many sections and subsections as needed.

4. PROPOSED SOLUTION

Games can be divided in following types:

- *“One click” games (FOSC - Fixed Odds Single Click)* – user bets on particular selection. Game random generates number and depending on the result, user is winning or losing.
- *High-Low games* –session is performing; user bets money and plays the game. Session enables transactions execute after the end of the game.
- *Lottery games* – there are a several players at the same time. When game starts all players raise their bets. System plays an important role; it has to place values and make all transactions, quickly and correctly.

Although each game has its specialties, there is a certain element collection, common for all games.

First we have to explain the system architecture, for which we proposed test framework. After system architecture, we will introduce you to our solution for online game testing problem.

Elementary structure of the system (Figure 1) consists of:

Games Room Server (GRS) is the central segment of the system. It is realized in the programme language Java, apropos Java Servlets and Java Server Pages technologies. Used data base is the Oracle. The games communicate with server using XML messages. All games are created in the Flash technology.

They can be animated from some of the useful explorers.

GamesWeb (GW) is an internet application used as an interface between Flash games and Games Room Server. This application enables users to actuate games from internet explorers.

GamesManager (GM) serves as backing up for games**.** Backing of the game comprises watching the game and the whole system, configuring prices, generating reports for game analysis, and coordinating in case system fails. GM is an internet application, which is available only for the client. Accessing this application is only by combination of username and password, so as with IP address license.

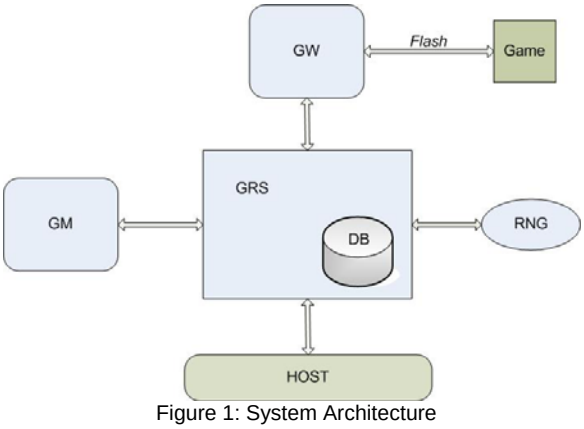

Figure 1: System Architecture

Random Numbers Generator (RNG) is elementary segment of the system. It is very important that this segment works properly. Quality of this part has direct influence on quality of the system, and its speed influences on maximum number of concurrent users. It can be implemented software (Java) as well as hardware.

Host system is already known system client, which has to enable financial transactions. Actually, it is a connection between client and electronic financial business system. It has to enable operations with client bills, registering clients, canceling transactions, etc. All client bills, including all necessary information, are stored in the system.

4.1 System Description on Concrete Example

Game testing system is realized on programming language Java. Thereat, backbone for running tests is test environment JUnit.

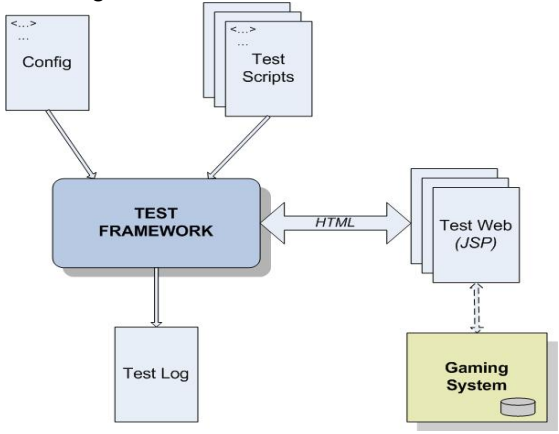

Figure 2: Test Framework

Figure 2 represents proposed components for test environment, which is being used for online game system (Figure 1) testing. It consists of main application – *Test Framework,* which inputs are *Config Files* and *Test Scripts*, and the output is *Test Log*, which holds information about the communication between *Test Framework* and gaming system, as well as the errors which occurred while the testing is being performed.

Communication between game and server is performed through XML messages. All tests are, also, written in a simple language based on XML, which has been developed for the purposes of system testing. Every XML tag has a corresponding Java class. During test execution, test framework reads a test and, according to decrypted objects, takes certain actions, calls adequate classes, thus bringing the system to adequate state.

Basic segments of tests are processes, which represent concrete actions that have to be executed. Each process establishes a connection to appropriate part of the system, after which specific action is being performed.

Processes can be grouped in three segments:

- *<initialization/>* - serve for system initialization prior to tests execution.
- *<destroy/>* - performed at the end of test execution and they do some final actions (e.g. releasing resources).
- *<tests/>* - perform concrete test actions; simulate game playing and check parameters.

System upholds control structures (e.g. exists adequate XML operator for *while* and *if* loops). That enables writing of powerful scenarios of tests for simulating game playing and testing game characteristics. Also, it is important to mention that there is a support for simulating concurrent playing of a number of players (XML marker exists, which enables execution of several tests at the same time). This is of great importance during performance testing.

Test Web (JSP) serves as an interface between test application and a system which is being tested. These pages receive HTTP requests and perform corresponding actions (logging in the system, getting session, setting parameters, setting random numbers, playing game…).

In order to speed up a process of testing, an application has been developed, which enables one to choose all automatic or performance tests that need to be run, and with just one click send them all into processing queue. During test execution it is possible to see number of tests performed, number of tests still waiting in the queue, number of errors etc. Also, every test has a corresponding log file in which one can see complete process of test execution, for the analyzing purposes.

Automatic Tests simulate playing the game. They should, during the game running, cover all bet options within the game and all values of the bets. This is not always possible, but should be

tried. Thereat, during the play, some of the important parameters should be checked inside of tests (bill account, bet, profit, paid sum, etc.).

Performance Tests are used for testing game performance. The main idea is not to check game parameters, but to simulate playing a great number of parties, by the great number of players (i.e. game is tested with 50 thousands played parties - 25 players, where each player played 2000 parties.).

Beside automatic and performance tests, it is important to mention existence of *manual tests*. Their main purpose is not to check game parameters, but to bring system into desired state, after which tester performs manual inspection of specific game functionality.

4.2 JUnit

JUnit is an environment for test writting, based on programme language Java. It is an instance of an architecture *XUnit* for test environment. It enables tester to write a set of automated tests which initialize, execute and inspect the code being tested. The tests are mutually independent and their sequence is not important. As a result, a programmer gets a report with total number of successfully and unsuccessfully performed tests.

4.2.1 Design

Environment design is based on two patterns, *Command* and *Composite*.

Command object is *TestCase*. Each class that comprehends test methods should be subclass of class TestCase. This class can define number of public testXXX() methods. If someone wants to check expected or current test result, he should call method assert(), or any of its modifications.

Subclasses of class TestCase which consist of more testXXX() methods can use setUp() and tearDown() methods for initialization and exemption each object which is being tested. These objects constitute test fixture. Each test works in context of its own fixture calling methods setUp() and tearDown(), before and after, respectively, each test method. With this action, it is prohibited side effects between test executions.

TestCase instances can be associated in *TestSuite* hierarchy, which automatically call all testXXX() methods from each TestCase instance. TestSuite is a combination of other tests, whether TestCase or TestSuite instances. This attribute enables to collect all tests, so they can be uniformly and automatically executed (there is only one result – success or failure).

4.2.2 Test Execution

Running tests is available either individually or within TestSuite object. In case of running TestSuite, automatically it runs all minor TestCase or TestSuite instances. Running TestCase object, it automatically runs all its public testXXX() methods.

JUnit provides either text or graphic user interface. Both intefaces show how many tests are running, errors, if they occured, and status report. The main characteristic of both interfaces is their simplicity.

For running tests using text interface, a command is used:

⇒ java junit.textui.TestRunner
ShoppingCartTest.

If all tests pass, text interface represent the message “OK”. Otherwise, an error message appears.

For running tests using graphic interface, it is used command:

⇒ java junit.swingui.TestRunner
hoppingCartTest.

Graphic interface shows *Swing* window with green line, if all tests pass, or with red line, in case some test fail.

4.3 Functional Testing Game “Fortune 36”

In this chapter, it is described the process of functional testing using concrete example, game “Fortune 36”. Appearance of a game is shown on the picture (Figure 3).

During functional testing, following phases have to be passed by:

1. Defining and introducing a set of test cases and writing adequate document (*test case document*).
2. Writing test cases according the document (preceding step).
3. Running and executing tests.
4. Writing test reports.

First two phases require most effort and time, and these phases have direct influence on correct and rapid passing through the rest of the testing. At the beginning, it is necessary to introduce the game itself, its characteristics and available options. This step is very important, because according this, test cases are noticed properly. Test cases should cover all game characteristics, with an accent on the limited situations, where an error can occur. That is why test cases should be carefully picked, which, sometimes, represents problem.

For testing the game “Fortune 36”, the following test cases have been noticed:

- *User Interface Test* – checks the buttons in the game; do all the buttons work properly. It does not matter if an account of the user is wrong calculated, as long as all the buttons do their job correctly.
- *Sound Test* – checks do music and sound effects, at the time they should appear, work correctly.
- *Game Play – Single Win* – a scenario of playing game in case of one winning. Special attend is on correct update of client’s bill (sum of entered and wined money). It checks weather these data are properly registered in application GameManager.

- *Game Play – Single Loss* – this scenario is similar to the previous one, except this scenario is for loosing money.

Figure 3: Game "Fortune 36"

- *Game Play – Several Bets, Only One Win + History Test* – tests situation when more dices are entered on more numbers (option where only one bet wins). Special attend is on correct update of client's bill (sum of entered and wined money). Also, it checks are the previous played games remembered within option History.
- *Game Play – Several Bets, Only One Loss* - this scenario is similar to the previous one, except this scenario is for situation where only one bet loses.
- *Game Play with Balance 1 Lost* – this is the situation when a client's bill account is 1, and client plays game with bet 1, and loses. Game, after this situation, has to notify the player that he has no more money on the account, and to forbid him to play any more.
- *Game Play with Bet Out of Balance* – tests the situation where player tries to bet the dice, which has bigger value than he has on his account. Game should obstruct this activity and send player adequate message.
- *Balance 0* – scenario where a player tries to run the game, even if his account is 0. Game should notify the player and forbid him to play.
- *Manual Payout* – for each game, for security reasons, should be defined limit for winning, after which the payment should be manually. This test tests does the game recognize this situation and, also, does it inhibit direct payment which is bigger that the limit is.
- *Changed Balance* – this is the situation when player bets a certain sum and before he confirm it someone aside changes his account (the account is now lower than the betted one). Game must recognize this situation and must not accept this bet.
- *Min/Max Stake* – for each game it is possible to set minimum and maximum bet value. This test tests situation when player tries to bet a sum, which is lower that the minimum value or bigger than the maximum value.
- *Games Manager Test* – tests work of application GamesManager; does the setting of a certain parameters inside this application influence the game itself.
- *Fun Mode Test* – each game can be played because of fun, not because of money. This scenario checks performing the game within this regime.
- *Automatic Tests* – these tests simulate playing the game.
- *Performance Profiling Test* – tests game performance (game "Fortune 36" is tested with 50 thousands played parties; i.e. 25 players, where each player plays 2000 parties).

After the document is over, for each test case, the test is being written. Structure of the tests is described in chapter 4.1. *Apache Ant* is being used for their execution. During the execution of the tests, it generates certain messages, which help to follow the flow of execution. If the test recognizes an error, it is being noticed in the report.

5. CONCLUSION

Number and complicacy of internet applications are being exponentially growing, and demands for their quality is being bigger each day. This paper represents process of software testing, with an accent on internet applications. Paper tries to explain basic aspects of application testing in the conditions, such as internet environment and privilege of the automatic testing. After theory of the automatic tests, it is given description of tools, which enables one to see how the process works, and it can also help choosing tools and evolution of new systems.

Testing on line games system represents good environment example, which is being created developing existing environments and their acclimatization on our demands and application specialties.

Software testing, particularly internet application testing, is extremely important process, which has to be attended special caution in case of serious and concurrent software product.

REFERENCES

[1] ANSI/IEEE Standard 729, 1983.
[2] Hetzel, W., "The Complete Guide to Software Testing," *QED Information Sciences Inc*., 1984.
[3] Myers, G. J., "The Art of Software Testing," *Wiley*, 1979.
[4] Frederick Brooks Jr. "The Mythical Man-Month,"
[5] David Janzen, Hossein Saiedian: "Test-Driven Development: Concepts, Taxonomy, and Future Direction," IEEE Computer, September 2005.
[6] Weissingerk, K., "Web Application testing with Puffin: Puffin testing framework, Part 1," http://www-128.ibm.com/developerworks/library/os-puffin.html
[7] http://jakarta.apache.org/
[8] Hansen, K., "Load Testing your Applications with Apache JMeter," http://javaboutique.internet.com/tutorials/JMeter/