UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

CURSO SEQÜENCIAL DE FORMAÇÃO COMPLEMENTAR EM ANÁLISE DE TESTES
TURMA 13

CITIESSE EXECUTION: A TOOL FOR GOOGLE SUBMISSION BASED ON MOTOROLA CASE

Authors

Artur José Miranda Júnior (ajmj@cin.ufpe.br)

Miscelânia Pedrosa de Araújo (mpa2@cin.ufpe.br)

Willamys Gomes Fonseca Araújo (wgfa@cin.ufpe.br)

Advisor

Prof. Roberto S. M. Barros (roberto@cin.ufpe.br)

Recife, May 2012

Abstract

This work shows the problem of test execution GMS (Google Mobile Service) for Motorola devices with Android. In performing the tests, there are some documents that must be followed: a workflow that illustrates in detail the implementation of the workflow, a document that describes how each test should be performed, and a worksheet for recording the results. These artifacts used in parallel require high effort and time to complete the tests. To solve this problem, the CITIESSE Execution tool was developed and will help the user perform the tests more rapidly and with a lower margin for errors, generating a file with the results of each test case performed, resulting in reduced effort and increased productivity for the tester.

Keywords: GMS, Automation, Test

Resumo

O presente trabalho aborda o problema da execução de um teste de GMS (Google Mobile Service) para dispositivos da Motorola com sistema Android. Na execução dos testes existem alguns documentos que devem ser seguidos: um workflow que ilustra detalhadamente a execução do fluxo de trabalho, um documento que descreve como cada teste deve ser realizado, e uma planilha para registro dos resultados. Estes artefatos utilizados em paralelo demandam alto esforço e tempo para completude dos testes. Para sua solução foi desenvolvida a ferramenta CITIESSE Execution, que servirá de ajuda para o usuário realizar os testes de forma mais rápida e com um índice menor de erros, além de gerar um arquivo com os resultados de cada caso de teste realizado, o que resultará na diminuição do esforço e uma maior produtividade para o testador.

Palavras-Chave: GMS, Automação, Teste

Acknowledgments

Thanks first to the One who was always by our side listening to us and giving us strength and faith to overcome all obstacles along the path. One who made us believe that we can, we are able. Thanks to our good and dear God.

We thank our families who have always been present and gave us all the support and understanding that we always need.

To the friends we have made over these nine months we have been doing this course, we welcome and thank you for being a second family to us.

We would like to thank CIn-BTC and Motorola for the great experience acquired during that period, both professionally and personally.

Finally, we would like to thank the Federal University of Pernambuco for the knowledge transmitted by our teachers through the course in Sequential Analysis Test, and especially our Advisor Roberto S. M. Barros, for believing in us.

Content

Figures List

Table List

Introduction

The *Compatibility Test Suite* (CTS) is a command mode tool to perform a series of test cases in Android. It is used to ensure compatibility of the devices and to report if the test results are valid or not valid. The CTS tests are performed frequently throughout the development process of a project. The CTS includes the following types of test cases: unit tests, that test atomic units of code inside the Android platform; functional tests, which test a combination of APIs together in a higher level of use cases; robustness tests, which test the durability of the system under stress; and performance tests, which aim to test the system performance against pre-defined scenarios.

This project will address the functional testing of the CTS called *Google Mobile Services* (GMS), which is a set of applications and services provided by Google android system, for a more comfortable use of mobile devices. This test is carried out to check the operation of applications and data for different operators.

The use of multiple documents written in natural language, the subjectivity of the worksheet to fill, and the lack of standardization of output artifacts make the execution time of test cases increase, and there is the possibility of different interpretations and consequently increased number of errors. The construction of the tool aims to unify these documents, reducing the subjectivity, the runtime and number of errors, so that the tester has a clear understanding and purpose in the execution of each test case.

1.1 Problems

As mentioned previously, the GMS test has the function to verify the operation of applications and services to different operators. This requires a few steps or activities.

The execution of GMS test has as result a spreadsheet that is populated by the tester responsible for its execution. The completion of this monitoring is done by following the steps in the GMS Workflow Execution flowchart, which contains all test cases, configuration steps, good practice and feedback. It also indicates which line must

be completed on the worksheet as well as the execution order, information that is provided via an id present in the test case in the flowchart.

The worksheet that is populated by the tester has all the results obtained with all tests executions. It consists of lines that correspond to the sequence used in the GMS Workflow Execution, which has an identifier (*id*), a brief description of each test case, and beside each one of them, the results are placed, as well as the screen images of the device being tested.

However, the flowchart and the spreadsheet do not have a description of what is needed in the execution of the test case, or what the expected result for each test case is. The spreadsheet requires only the information that the test case passed, failed or is not supported. Another need which is absent in both the worksheet and in the flowchart is the description of the process of getting some inside information from the device. Therefore, it is necessary that the tester executes some commands on the computer with the device connected to it. The spreadsheet provides only commands when it could be done in a more automated way.

In order to improve the understanding of the test cases there is another document that contains the description of the tests, as well as the expected results, but this document is not properly updated and even has a monitor for it.

However, analyzing the implementation process of the GMS, one realizes that it needs some additional documents to complete the test, but this intercalation between documents eventually leave the execution somewhat laborious and takes more time to complete.

1.2 Objectives

The following are the general and specific objectives of this work.

1.2.1 General Objectives

To develop an application to perform Google Submission tests, particularly for tests of GMS, using the Java programming language, in order to improve the quality, time and standardization of test execution and to make a comparative study on the effort required with and without the use of the tool.

1.2.2 Specific Objectives

- To specify and analyze the requirements of the tool;
- To implement and test the tool;
- To evaluate the results obtained with the use of the tool.

The rest of this document is organized as follows. In section 2 the development environment used in the implementation of the tool will be described. In section 3 we discuss the functional requirements, nonfunctional requirements, and, as a form of better understanding, a flowchart and the entity relationship model of the tool will be presented. The description of the tool is discussed in section 4. Section 5 will be an evaluation of its operation, as well as a comparison between runs made with and without the tool. Finally, in section 6, we present the results and possible improvements that can be incorporated into the software.

2 Development Environment

2.1 Java

The Java [3] language was chosen for the development of the tool. Because it is portable, simple, object-oriented, allowing modularity, reuse and code maintenance is simple and safe, it ensures the integrity of the code. In addition, the Java language was chosen due to the fact that there is extensive documentation, for being on the rise at the moment and also became the development team is already trained and comfortable with it.

2.2 Integrated Development Environment (IDE): Eclipse

To use the Java programming language, as has been previously specified, the use of a tool for rapid development and quality is of paramount importance. For this we chose the Eclipse [2] 3.7 IDE [6], which was developed in Java, following the model of open

source software development, and development-based plug-ins that seek to meet the diverse needs of each programmer.

2.3 System Version Control: Git

The system Git [7] is a distributed version control system with emphasis on speed. The Git system was chosen to control versions during tool development. Each Git directory contains the historic and ability overall of reviews control, moreover, it does not depend on the access network or the central server.

3 Software Project

3.1 Functional Requirements

This section describes the main functional requirements for the development of the tool CITIESSE Execution. For this purpose we used use cases [4, 5], which is a modeling technique used to describe what a new system should do. We defined a set of instances of use cases, where each of these instances describes a sequence of actions performed by a system or part thereof.

3.1.1 Use Case 01 (UC01): Subscribe Test Case Setup

Description: This use case is responsible for registering the information to be used to configure the device.
Actor: Administrator.

Priority: (x) Essential ( ) Important ( ) Desirable

Pre-conditions:

- The administrator must be authenticated and logged into the system with his/her privileges.
- Test cases must be previously registered.

Post conditions: The configuration information is registered successfully.

Main events flow:

1. User must inform the test case ID that will use the setting.
2. User must inform the description of the steps for configuration.
3. The use case is closed.

Alternative Flows:

To step 1:

If the test case ID has not been registered, a message is given to the administrator stating that the test case was not registered and the operation is canceled.

3.1.2 Use Case 02 (UC02): Subscribe Test Case

Description: This use case is responsible for registering each one of the test cases that will be performed during the test run of GMS.

Actor: Administrator.

Priority: (x) Essential ( ) Important ( ) Desirable

Pre-conditions: The administrator must be authenticated and logged into the system with his/her privileges.

Post conditions: The test case is registered successfully.

Main events flow:

1. The user must enter the ID of the corresponding test case;

2. The user must provide the name of the test case;
3. The user must enter a step-by-step description of the test case that will be executed;
4. If necessary, the user can add a note to the corresponding test case;
5. The use case is closed.

3.1.3 Use Case 03 (UC03): Execute Test

Description: This use case is responsible for registering the information of the test execution.
Actor: Tester.

Priority: (x) Essential ( ) Important ( ) Desirable

Pre-conditions: The tester must be authenticated and logged into the system with his/her privileges.
Post conditions: The information is registered successfully.

Main events flow:

1. The user name, start date and end date of the execution are automatically obtained by the system;
2. The use case is closed.

3.1.4 Use Case 04 (UC04): Report Test Executed

Description: This use case is responsible for generating a report of the tests that were run.
Actor: Administrator.

Priority: (x) Essential ( ) Important ( ) Desirable

Pre-conditions: The administrator must be authenticated and logged into the system with his/her privileges.
Post conditions: The report is delivered successfully.

Main events flow:

1. Include (UC03 – Execute Test);
2. The user enters the name of the tester or the date of execution of the test to obtain information from the test performed;
3. The user selects the application that he wants to get the test information;
4. The user chooses the option to send the report containing the information of the test run, as well as its execution time and who performed the test;
5. The use case is closed.

Alternative Flows:
To step 1:
This use case makes a query to the use case *Execute Test* to get the information from the beginning and end of test execution, as well as the implementation of the ID and name of the user who performed the test.
If the ID of the test run has not been registered, a message is given to the administrator stating that the test was not registered and the operation is canceled.

3.1.5 Use Case 05 (UC05): Results

Description: This use case is responsible for storing the execution results of each of the test cases.
Actor: Tester.

Priority: (x) Essential ( ) Important ( ) Desirable

Pre-conditions: The tester must be authenticated and logged into the system with his/her privileges.
Post conditions: Results are stored successfully.

Main events flow:

1. Include (UC03 – Execute Test);
2. The user reports the result of the test that is being executed, which can be 'yes', 'no', 'N/A' or 'not supported';
3. If necessary, the user adds a comment for this test case execution;
4. The use case is closed.

Alternative Flows:
To step 1:
This use case makes a query to the use case *Execute Test* to get the ID of the test run.

3.1.6 Use Case 06 (UC06): Generate Spreadsheet

Description: This use case is responsible for generating the GMS sheet with the results of all test cases executed.
Actor: Tester.

Priority: (x) Essential ( ) Important ( ) Desirable

Pre-conditions: The tester must be authenticated and logged into the system with his/her privileges.
Post conditions: The spreadsheet implementation of the GMS is successfully generated.

Main events flow:

1. Include (UC05 - Results);

2. Information 'Results' and 'Comments' are obtained and included in the spreadsheet implementation of the GMS;
3. The user chooses the option to generate a spreadsheet with the results of each test case;
4. The use case is closed.

Alternative Flows:

To step 1:

This use case makes a query to the use case *Results* to obtain the information and results and the reviews of each of the test cases that were executed.

If the test results have not been registered, a message is given to the administrator stating that the test was not registered and the operation is canceled.

Figure 3.1.1 represents the use case diagram, which shows the sequence of events for each actor participating in the system, allowing for a better understanding of its functionality.

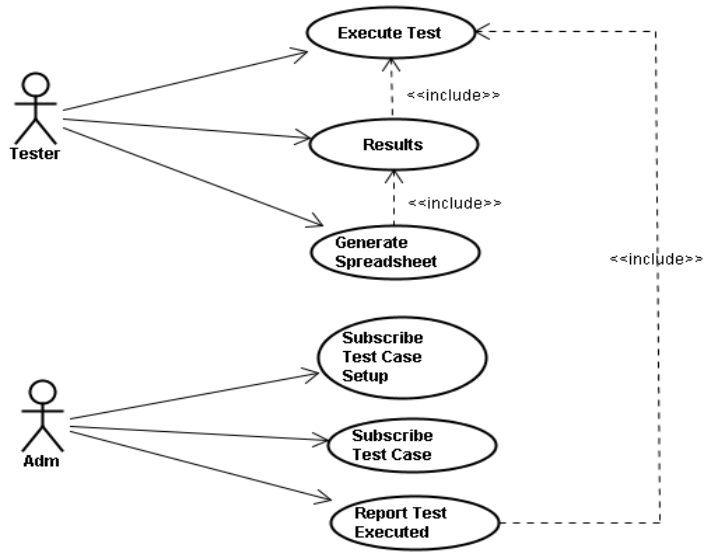

Figure 3.1.1 – Use Case Diagram

3.2 Non-Functional Requirements

This section describes the main non-functional requirements for the development of the CITIESSE Execution tool.

3.2.1 Usability

NFR01: Usabillity
The user interface is of great importance to the success of the system. Even if the system is used frequently, often the user does not have enough time to learn it. Thus, the system must have a friendly interface for both new users and for more experienced users, it should not be difficult to use and should not be tiring.

3.2.2 Performance

NFR02: Performance
Although not considered an essential requirement, because it is a factor of software quality, performance is an important requirement for the system.

3.2.3 Security

NFR03: Security
Since the system will be web-based, it needs to be as safe as possible. This is why only authorized persons with their privileges should have access to the system.

3.2.4 Hardware and Software

NFR04: Eclipse 3.7

The tool will be developed using Eclipse [2].

NFR05: J2EE 7.0

As the tool will be used via the Web, the development platform is Java J2EE [8], which consists of a set of services, APIs and protocols, with functionalities for developing multilayered, with model-based components.

NFR06: JSF 2.0

It is a MVC web application framework that simplifies the development of user interfaces. It allows the creation of user interfaces through a set of predefined components, provides a set of JSP tags to access components, reuse of components on the page, and inserting style sheets (CSS), Java Script commands and Ajax methodology [9].

NFR07: MySQL 5.1

The management database system used was MySQL [10], which uses SQL as the language interface and is currently one of the most popular databases. Moreover, it presents portability, compatibility, ease of use, excellent performance and stability.

NFR08: Hibernate 3.2.5

It is a framework for object-relational mapping, which facilitates the mapping of attributes between a traditional relational database and the object model of an application through XML or Java annotations. Its goal is to provide a reduction of the complexity of Java programs that are based on the object-oriented model and need to work with a relational database. A very important feature is that it transforms the classes that are in Java for data table [11].

NFR09: Prime Faces 3.1.1

Framework that provides a rich set of components for JavaServer Faces [12]. The construction of the components was made to work with Ajax, and provides support to create features that utilize Ajax Push and allows the application of topics with the objective of changing the appearance of components in a simplified manner.

NFR10: Apache (Tomcat - 7.0.26)

The Apache server [13] was used, which is a free web server that allows webcast, implementation of distributed applications and data processing. It is compatible with HTTP 1.1, with functionality that remains in a brick structure, which allows the user to write his/her own modules using the API software.

NFR11: JUnit 4.11

JUnit is an open-source framework that facilitates the creation of automated tests in the Java programming language. The results are presented and show if the class methods are working correctly and displaying possible errors or failures.

NFR12: Android SDK

To perform the tests on any computer, it is necessary to install the Android SDK, which provides tools and libraries needed to run Android devices on the computer where the tests will be executed.

NFR13: Browser

To perform the test, one of the browsers on the market is required so the user can access the page of the tool developed.

3.3 Entity Relationship Model

A data model is the representation of a set of information requirements of the business. It is an implementation model in which the persistent data of the system are represented logically and physically. In addition, it comprises the behavior defined by the database, such as constraints, triggers, stored procedures, among others.

The representation of the data model used for implementation of the database system is presented in Figure 3.3.1.

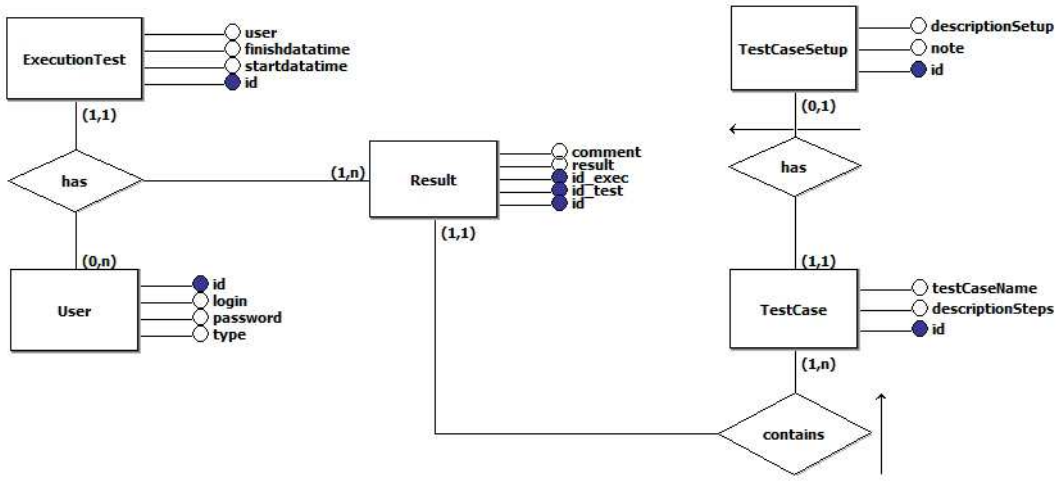

Figure 3.3.1 – Data Model

3.4 Flowchart

To perform the GMS test some artifacts are used, as previously stated. One is the flowchart GMS Execution Workflow flowchart, which describes the order of execution of all test cases that need to be performed by the tester, as well as the actions setup for pre-configuration of the device and some good practices that can be followed. The flowchart is presented in Figure 3.4.1.

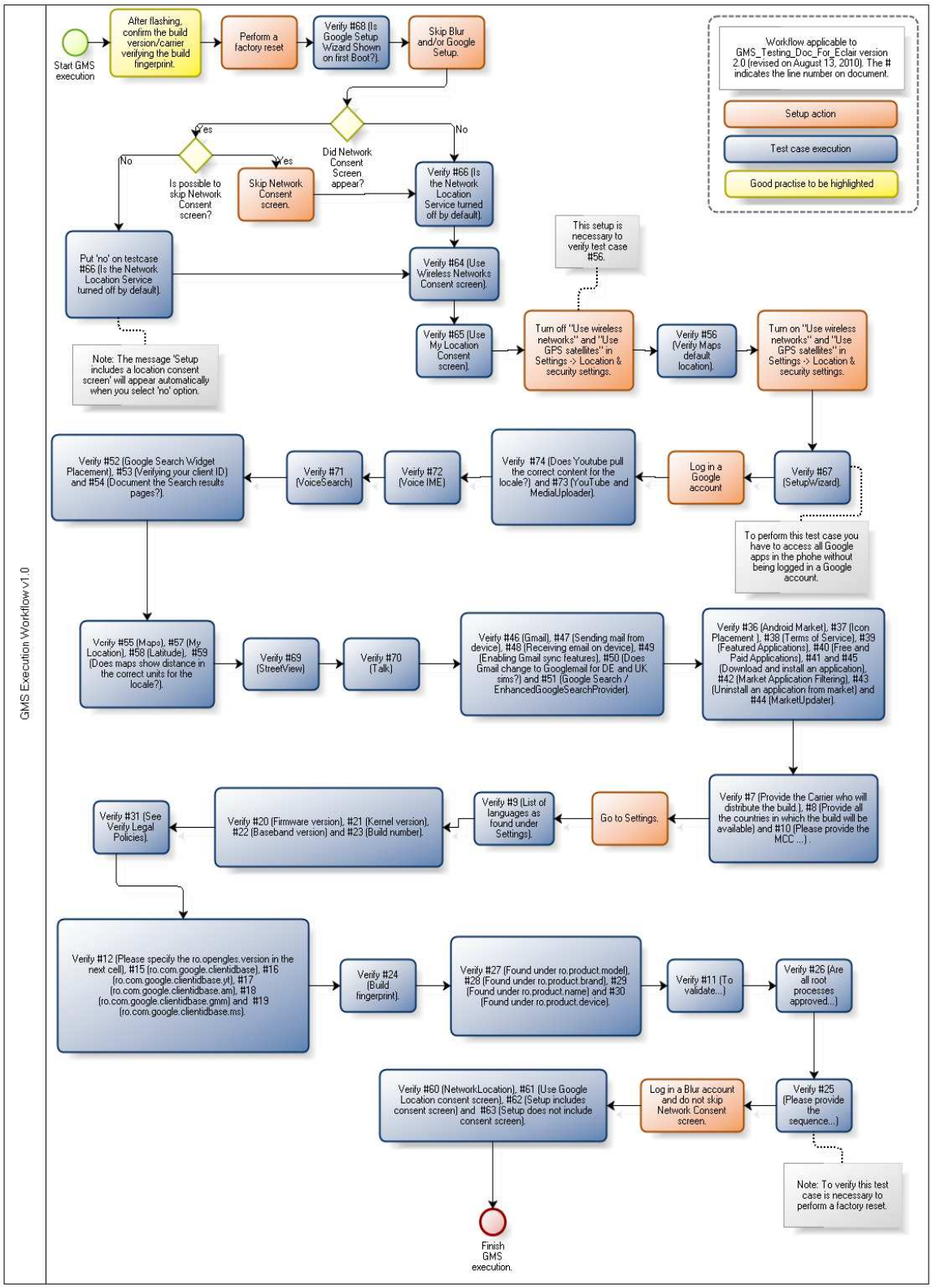

Figure 3.4.1 – GMS Execution Workflow

4 Implementation

The CITIESSE Execution Tool is a software solution for the execution of one of the tests performed on Motorola devices, the GMS test. Currently, testing is done manually and with the help of some artifacts, which are used in parallel. With the creation of the tool it is no longer necessary to use many artifacts to perform the tests because all information obtained from these documents will be unified in the tool, aiming at better performance during the test execution.

4.1 Structure of the Tool

The Citiesse Execution is a tool development using JavaServer™ Faces (JSF) technology, which is the standard component-oriented user interface (UI) framework for the Java EE platform. More familiar terms, it is a Java-based web framework [9].
The tool architecture is based in the Model, View and Controller (MVC) design pattern. It is based on three objects: the *Model,* which is an application model, the *View* that represents the application screen, and the *Controller* that defines the way the UI reacts to its inputs [16]. Figure 4.1.1 illustrates the design pattern objects interaction.

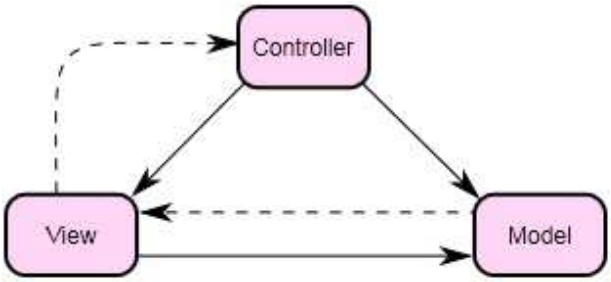

Figure 4.1.1 – MVC

In the tool, the object *View* is composed of the XHTML Pages and Cascade Style Sheet (CSS), which in the JSF works with the two special tag libraries for JavaServer

Pages (JSP) to express the interface for JSF inside a JSP page. The organization of the tool view is presented in the Figure 4.1.2.

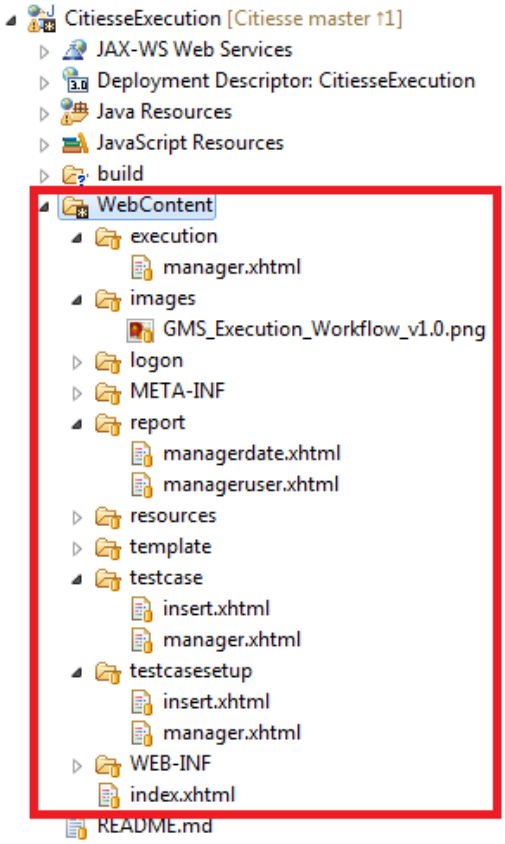

Figure 4.1.2 – View Structure

The *Model* is composed of the Entity and the Data persistence. In this project we used the Data Access Object (DAO), another design pattern whose function is to implement a mechanism to access information stored in a data source, or when you want to decouple the data persistence layer. Together with DAO we used the Hibernate [4] Technology whose function is to facilitate the storage and retrieval of Java domain objects via Object/Relational Mapping. Today, Hibernate is a collection of related projects enabling developers to utilize POJO-style domain models in their applications in ways well beyond Object/Relational Mapping. The organization of the tool model is presented in the Figure 4.1.3.

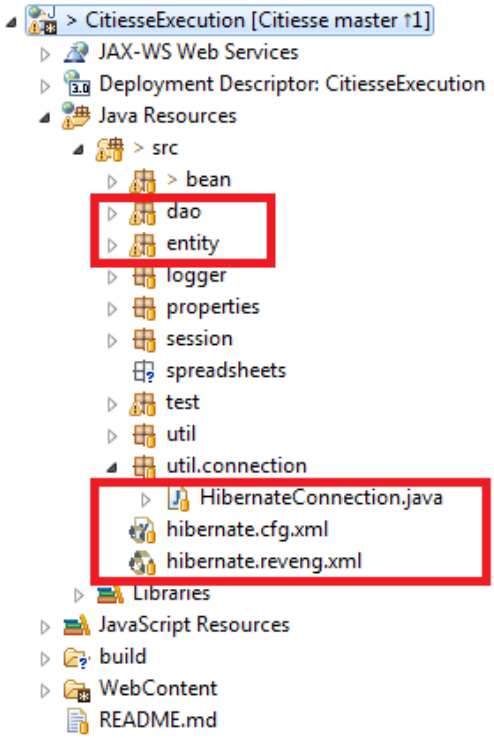

Figure 4.1.3 – DAO and Hibernate

The Controller is composed for control classes. In the JSP the managed bean works as an intermediate layer between our XHTML pages (View) and the Entity/Data Persistence (Model). The organization the tool controller is presented in the Figure 4.1.4.

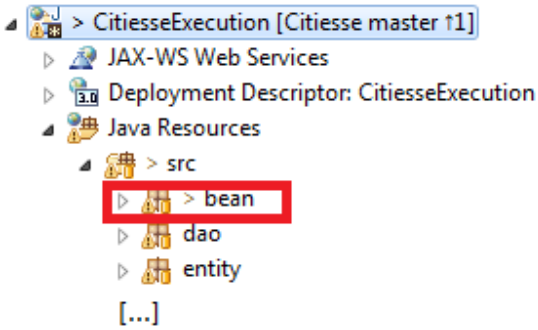

Figure 4.1.4 – Controller

4.2 User Interface

As a way of better understanding of the CITIESSE Execution tool, below we presented the interface of its most important features as well as a short description explaining its operation.

Figure 4.2.1 is the initial screen interface of the CITIESSE Execution tool. From there the user can navigate throughout the application. Clicking on 1, the user is directed to the Home screen; clicking on 2, the user is directed to the test cases screen; clicking on 3, the user is directed to the test case setup screen; clicking on 4, the user is directed to the test execution GMS screen; clicking on 5, the user is directed to the test reports screen; and clicking on 6, the user exits the tool. These options are available according to the privileges of the user who is logged into the system.

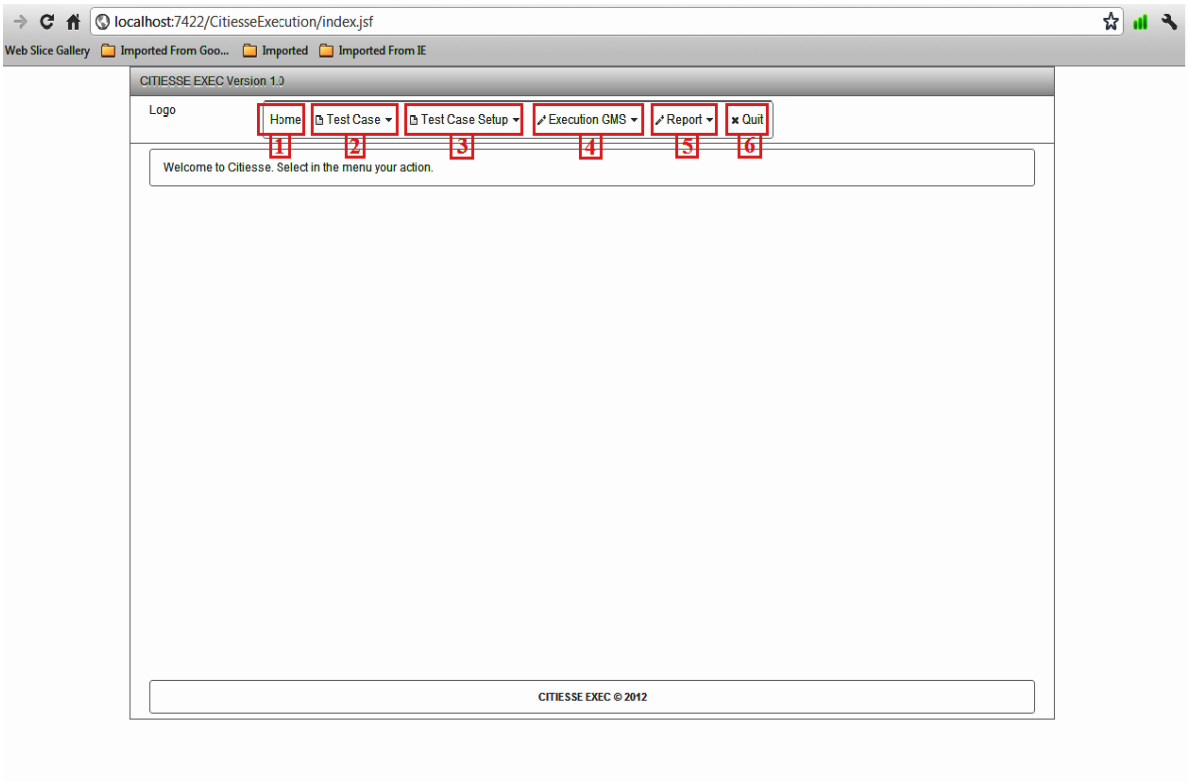

Figure 4.2.1 – Main Screen Interface

Figure 4.2.2 presents the query interface for each test case that is registered in the system. The user can search for a test case by any of the fields 1, 2, 3 or 4. In 5 all the test cases that have been registered in the system, as well as all information inserted in each of the fields mentioned above, are listed.

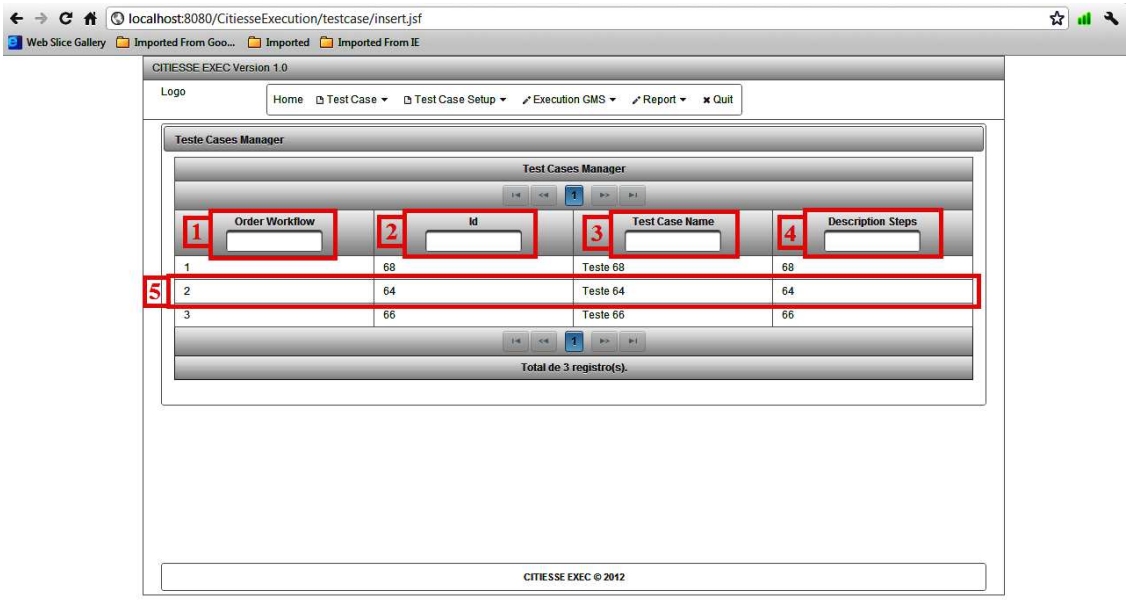

Figure 4.2.2 – Test Case Manager Screen

Figure 4.2.3 presents the interface used to register the test cases setup. To include the test case setup, the user must fill in each field that appears on the screen. In field 1 the id of the test case that requires setup to be performed should be inserted; in field 2 the description of the setup, with details of what should be done before the test case is executed shall be inserted; in field 3 a note with information regarding the setup can be inserted or not and in 4 all the test case setups that have been registered in the system, as well as all information inserted in each of the fields mentioned above are listed.

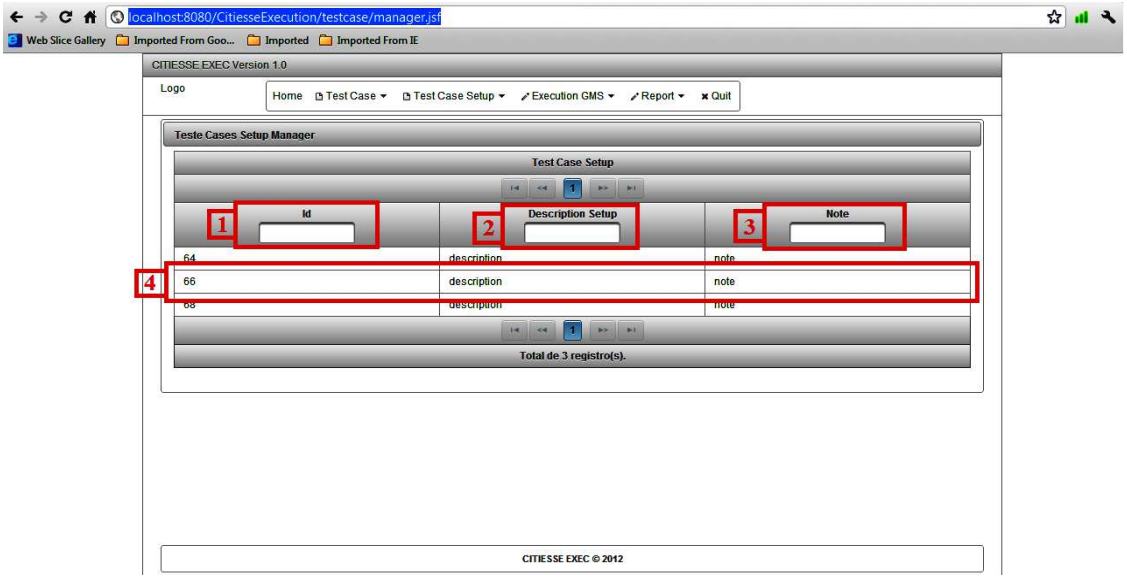

Figure 4.2.3 – Test Case Setup Manager Screen

Figure 4.2.4 presents the interface used to register the results of each test case of the GMS. This interface displays in the flowchart in which position the test case is, which is represented by the field 1 in the figure. The description of the test case is displayed by the field 2. The user can include any comments regarding the test case being performed, which is inserted in field 3 of the Figure. To include the result the user simply chooses one of the options that appear when he/she clicks in field 4. To save the results the user must click the Save button, represented by the field 5.

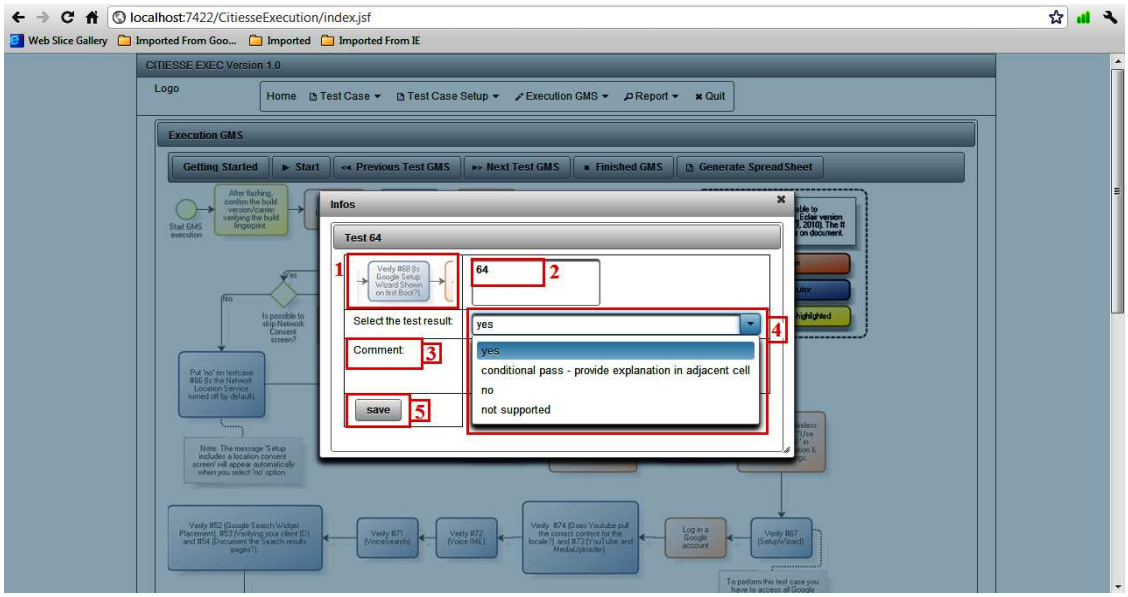

Figure 4.2.4 – Results Interface

Figure 4.2.5 presents the interface that shows how the test run of GMS will be with the tool. By clicking on the Getting Started button, represented by the number 1, the tool displays an introduction about the GMS test and starts the session of the test. To start the test, the user clicks on the Start button, represented by number 2. To return to the previous test case, the user just clicks on the Previous Test GMS button and to advance to the next test case, the user clicks on the Next Test GMS button, represented the numbers 3 and 4 respectively. Hovering the mouse over the desired test case its image is displayed more clearly, as the number 7 in Figure shows. To register the results the user clicks in the chosen and the tool test case desired that will show the results interface for this test case (see Figure 4.2.6). To generate the results spreadsheet of the GMS test, the user clicks on the Generate Spreadsheet button, that can be viewed in the number 6. To finish the execution and save the results obtained during the execution the user clicks on the Finished GMS button, represented by the number 5 in figure. In 8, there are more information about the GMS test, more specifically, the Test Case Setup, the Test Case and Good Practice directions.

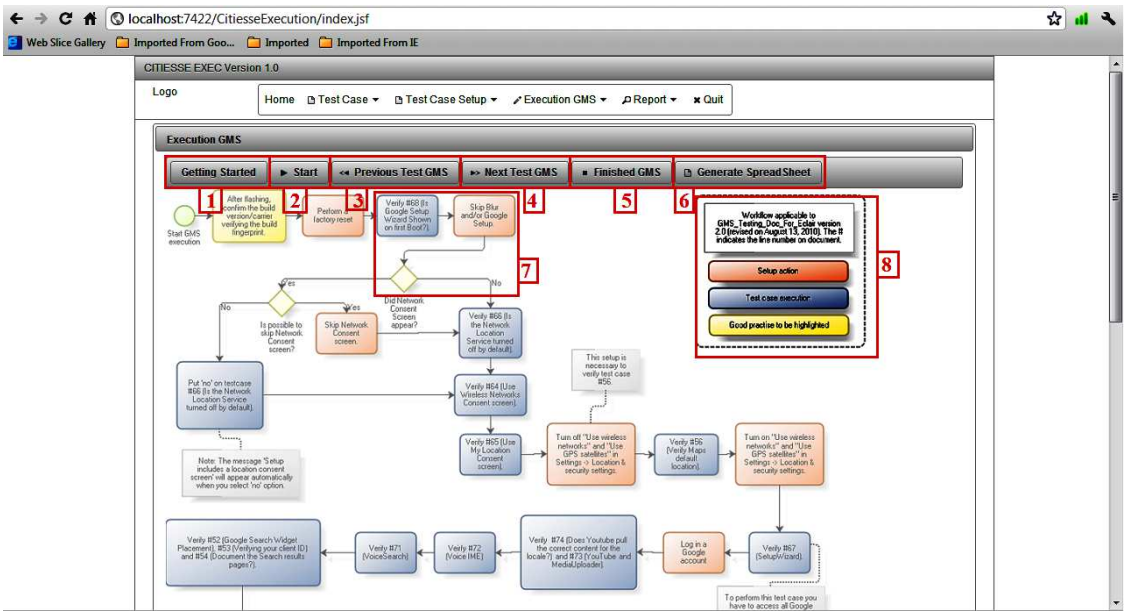

Figure 4.2.5 – Execution GMS Screen

Figure 4.2.6 presents the registration of each interface test case that will be executed by the tester. To make the inclusion of test cases the user must fill in each field that appears on the screen. In field 1 the id of a test case is inserted according to the execution spreadsheet of the GMS test; In 2, the name of the test case should be inserted, in field 3 the description of how the test case should be performed should be inserted, i.e., the step-by-step test case; in field 4 the order in which the test case should be performed according to the flowchart (see Figure 3.4.1) should be inserted. Note that all fields must be filled. After the insertion of all information, the user registers the result of test case by clicking on the Insert button as indicated by number 5.

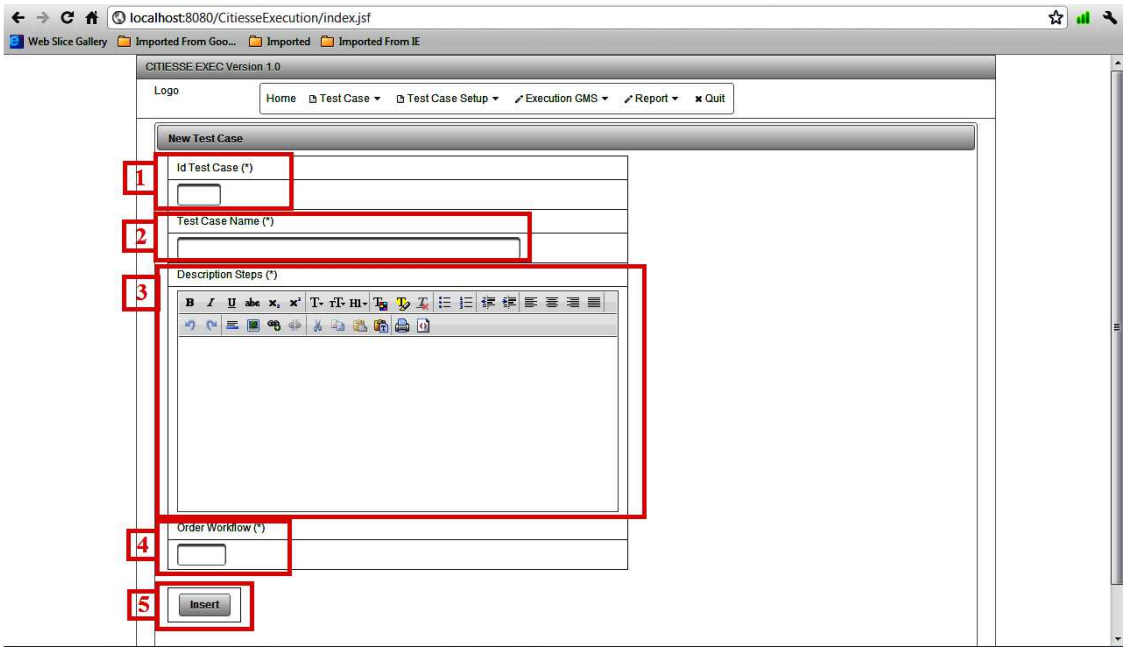

Figure 4.2.6 – New Test Case Screen

4.3 Tests

To ensure the quality of the software we run some tests. This section describes the entire testing process used in developing the project.

The software testing [14] performed consisted in seeking information in the system that would help ensure the quality of the product that is being developed. Initially the tests should be projected to pass, to determine that the implementation does what was estabilished in the specification [14]. In the following stages of development, we also tested alternative flows of the system: at this stage, the projected test cases aimed to insert failures in the system to evaluate its behavior in anomalous situations. In this project, we used two techniques of testing: white box and black box, as well as various stages of testing: unit testing, integration testing and system testing. These techniques are described below.

4.3.1 White Box

The white box test technique, also called structural test or oriented logic, aims to evaluate the internal behavior of the software. Test cases are created based on the source and aim to scroll through the different logical paths of the system. In this project, this technique was used together with unit testing and integration.

Unit Test: Individual components were tested to ensure that they operate correctly, boundary conditions were also tested to ensure that the units or modules operate properly within the limits established. With this technique it was possible to check the flow of conditional and logical operation of the units tested.

Integration Test: After the unit tests were performed separately and behaved as expected we moved to integration testing. The integration of the modules was done incrementally, each small segment was grouped and tested soon after, so when and if any errors occur the tester would have no trouble finding out the two conflicting parts of the action. Integration testing related to communication with the database were also made, which assessed the logical flow of all methods responsible for manipulating the database, such as insert, update, selection and removal, that is illustrated in Figure 4.3.1.

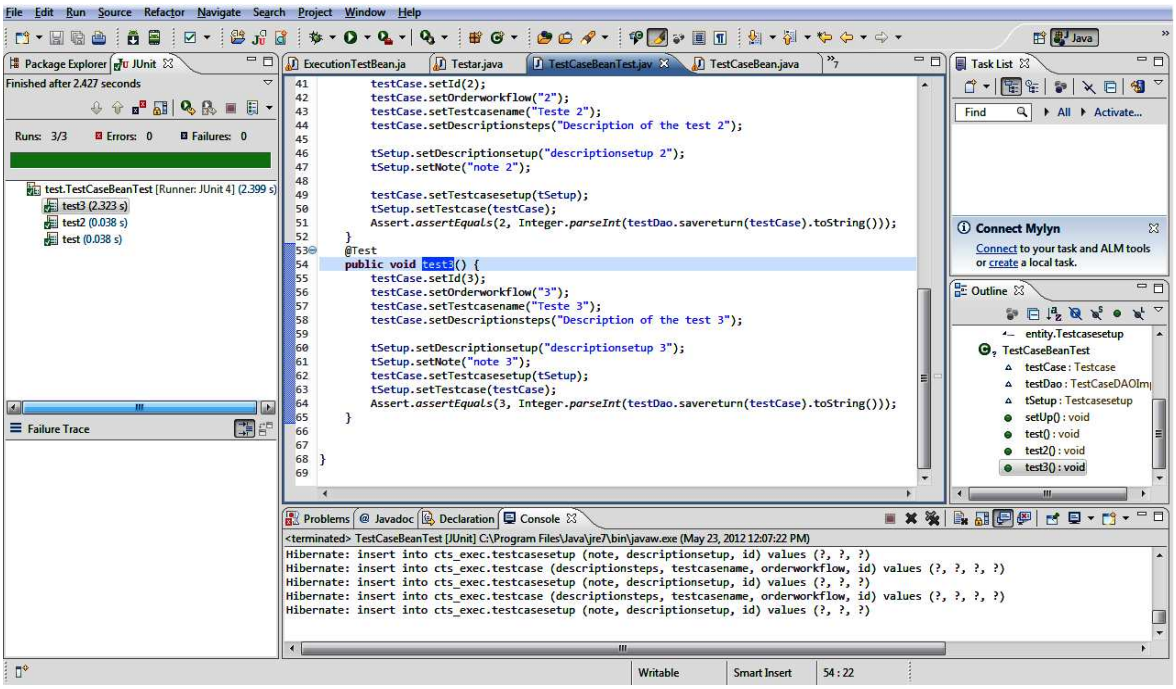

Figure 4.3.1 – JUnit Test

4.3.2 Black Box

The black box test technique, or functional test, takes into account the external behavior of the software, no matter how it works internally. The test designer must develop and project the tests based on the requirements raised and accepted by the client. These tests are done providing input data and comparing output data with expected data (previously known). In this project, this technique was used together with unit testing and integration and system testing.

Unit Test: In this step different test cases were designed aimed at testing the correct operation of software modules in terms of results. The test cases were implemented such that the input data were provided, the test was executed and the result obtained was compared with the expected result previously known. In this test, evaluation was made solely on the value of the returned result, regardless of the conditional flow, variable values internal or the logic of the system.

Integration Test: Integration testing using the technique of black box aims to verify the correct functioning of the parties acting together as a whole. The test cases are implemented such that the data stream walk down two or more parts.

5 Results Evaluation

The purpose of this section is to present the reader with the expected results of the GMS test using the CITIESSE Execution tool.

Initially, the test executions were performed manually. With the help of some artifacts needed to understand how to perform each test case, the tester should search these artifacts trying to find the appropriate information to achieve the same. This method of execution as well as being dull, because the tester should look for different information in several documents, it also leaves a margin for interpretation, which could cause errors. With the tool, the user does not need to access many documents simultaneously, because the information contained in all of them, relevant to each execution of the test case, will be presented on the screen of the tester, the respective test case being executed.

With the use of the tool in the GMS test, it is possible to make tests faster, and more comprehensive and with more clarity of the information passed to the tester, thus reducing the margin of subjectivity of different information and consequently increasing the rate of correctness of the artifacts produced.

The main features of the tool were developed and are ready to be used, allowing the system administrator to register all test cases of the GMS as well as their descriptions and explanations on how to accomplish them. In addition, it is not necessary to use many documents. Furthermore, the results can be obtained and collected with the use of the tool and sheet results can already be successfully generated, with only the addition of images to be obtained after performing the test cases and internal information to the devices, are obtained through scripts.

It was not possible to run complete tests to adequately evaluate the performance of the tool. In order to make an assessment of effort and time spent running, we performed some GMS tests *with* and *without* the use of the tool which enabled us to make estimates.

With the use of the tool we selected and run 10 tests in the order they appear in the flowchart. In this it run has been spent about 20 minutes. From this result, and knowing that the test GMS consists of 75 test cases, we obtained a rough estimate of time of 1 hour and 50 minutes. For other tests, which were not implemented in this version of the tool, we estimated 29 minutes, giving a total time of 2 hours and 19 minutes of running.
For a tester who has experience with the GMS test, the actual runtime gets to be around 1 hour and a half, compared to a tester who has never conducted a GMS test and has knowledge on the Android platform, who would perform the test, without the tool in about 5 hours. The hours was obtained using a instrument of measure (CHRONOS tool) .
Table 5.1 shows more clearly the results obtained:

Table 5.1 - Results obtained after the execution

	Experienced Tester	**Inexperienced Tester**
Without Tool	3h 40 min	5 horas
With Tool	2h 19 min	2h 19min
Difference (Without Tool – With Tool)	1h 21min	2h 41 min

6 Conclusion

In this section we present the results of this work as well as some possibilities of improvements for CITIESSE Execution.

6.1 Results

The main motivation for the development of this work was the perception that we had to improve the time efficiency and to standardize the manual executions performed in Motorola devices. Initially, it was possible to implement the tool for implementing the GMS test.

We found out that, when using the CITIESSE Execution tool, the test engineer can perform the test in a more efficient and simple, way using only the tool instead of all the artifacts that were previously used to understand the test cases and inclusion of results. This indicates that the use of the tool the user can achieve more consistent and standardized results. So, we consider that our objectives have been achieved in order to optimize the GMS test and decrease the effort spent on their implementation.

6.2 Difficulties

The difficulties are present in the development of any and all software projects. Throughout the development of this project, several difficulties were encountered, some were instrumental in the final result product.

Unstable Requirements: constant changes that occurred during the development of the software made the elicited requirements quite unstable. Even in this short period, some of the requirements became inappropriate and need to be reviewed and changed.

Tight Schedule: One of the major difficulties in any software development project is related to follow and maintain the activities described in the schedule. In this project, this difficulty became even more evident given the different roles and responsibilities played by stakeholders.

6.3 Future Work

This work aimed to show the feasibility of using the CITIESSE Execution tool for the execution of GMS tests, as a way to optimize the effort spent on execution and reduce the error rate. Although the tool works as expected, it still needs some improvements, so that the tests can be performed at their best.

Future work will include improving the tool for the test GMS with the inclusion of some features for implementing the test, as the inclusion of some images of the device and some internal information that is obtained through scripts. Furthermore, we expect

that the tool is ready to add all the other tests which are executed in manual from Motorola devices, to obtain the results that were obtained for the GMS test. Among the tests that can be optimized by using the tool we can mention the CTS Verifier, Data Migration Test and Widevine.

References

[1] LUCKOW, D. H., Melo A. A.. *Programação Java para Web*. Novatec, 2010.

[2] JAVA DEVELOPMENT WITH ECLIPSE, available at: http://www.developer.com/lang/article.php/3528616/Java-Development-with-Eclipse.htm. Accessed on 30/Mar/12.

[3] LINGUAGEM JAVA, CIn-UFPE, available at: http://www.cin.ufpe.br/~arfs/introjava.pdf. Accessed on 30/Mar/12.

[4] CASO DE USO, available at: http://www.wthreex.com/rup/process/artifact/ar_uc.htm. Accessed on 10/Apr/12.

[5] MODELANDO SISTEMAS EM UML - *Casos de Uso*, available at: http://www.macoratti.net/net_uml2.htm. Accessed on 10/Apr/12.

[6] OS IDE's *(Ambientes de Desenvolvimento Integrado) como ferramentas de trabalho em informática*, available at: http://www-usr.inf.ufsm.br/~alexks/elc1020/artigo-elc1020-alexks.pdf. Accessed on 22/Apr/12.

[7] GIT, available at: http://git-scm.com/. Accessed on 30/Apr/12.

[8] PLATAFORMA JAVA J2EE*: a atual pedida no cenário corporativo*, available at: http://www.luizmatos.eti.br/artigos/Plataforma_Java.pdf. Accessed on 6/May/12.

[9] JAVA Server Faces. *ORACLE, Sun Developer Network*, available at: http://javaserverfaces.java.net/. Accessed on 6/May/12.

[10] MYSQL - o que é?, available at: http://www.oficinadanet.com.br/artigo/2227/mysql_-_o_que_e. Accessed on 7/May/12.

[11] INTRODUÇÃO AO HIBERNATE 3, available at: http://www.guj.com.br/content/articles/hibernate/intruducao_hibernate3_guj.pdf. Accessed on 7/May/12.

[12] WHY PRIMEFACES, available at: http://primefaces.org/whyprimefaces.html. Accessed on 7/May/12.

[13] APACHE TOMCAT, available at: http://tomcat.apache.org/. Accessed on 7/May/12.

[14] MOTA, A. C.; VASCONCELOS, A. M. L.; Testes de Software. Recife, 2012. 179 p. Handout of Sequential Course of Formation Complementary in Test Analysis - Center Informatic, UFPE.

[15] JUNIT.ORG. *Resources for Test Driven Development*, available at: http://www.junit.org/. Accessed on 20/May/12.

[16] GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. Design *Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[17] MODEL View and Controller. *Wikipédia, a enciclopédia livre*. Disponível em:http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller. Accessed em: 16/May/12.

[18] HIBERNATE, *JBOSS Community*. Disponível em: http://www.hibernate.org/ . Accessed em: 22/May/12.