1. Introduction

One of the cores of the software testing process is the implementation and testing execution phase. This phase includes, among other activities, specifying the procedure to be adopted in addition to all the information necessary for execution of the tests,for example, the environment preparation and device configuration [1].One of the key points of this phase, is the configuration activity of the devices to be tested and loading data necessary for testing execution, this activity consist in the selection and inclusion of this dataand, carrying out necessary settings to run tests. Usually this activity is performed manually by testers, following the procedures listed during the analysis and modeling of test cases [2]. Mistakes made in the preparation procedures may lead to inaccurate results, in addition to increasing the consumption of time required in this activity.

In recent years, the automation of software testing has been widely adopted by technology companies.Due, the possibility of repetition of the certain testing activities, with low cost combined with an increase in test coverage [3]. Test automation is seen as an alternative for improvement and efficiency of the testing process as well as, a means to ensure the quality of a product under development [2].

The automation of testing procedures, can also be adopted to ensure a correct configuration of the item being tested. The automate the configuration procedures is useful when it is necessary to have different settings on multiple devices or even to unify this setting.

The Project CIn / Motorola, a partnership of the Center for Informatics of the Federal University of Pernambuco, with Motorola mobility,develops testing activities on mobile devices.Tests conducted on the project using various approaches: some tests are performed manually, automated or semi-automated manner.With respect to automated testing, the creation and use of test scripts, is one of the means adopted for the implementation of this approach in the Project CIn / Motorola environment.This scripts are responsible for the configuration and, in some cases they also perform the test.

In most cases, the scripts are created to perform interaction with GUI elements, simulating manual use of the device.However, this technique has some problems, according to [4] automated tests has a Strong dependence with GUI components of the application. Changes in graphical user interface of the device such as: a different kind of screen resolution and addition of a new prompt for example, may hinder the Script reuse for different devices.

Moreover, another problem that often occurs in the environment of the Project CIn/Motorola is to changes in the system versions (builds), as well as the region where this system will be adopted. This entails a change in the position of elements in the GUI and even adding new screens, generating inaccuracy in executing automated tests. Another issue that arises in this approach is portability, as some scripts are written to be compatible with the Linux system, while there are scripts that run only under Windows.

To run tests on different platforms, the script can not be reused, it is necessary for it to be translated to the platform which tests will be performed. According to [4], one of the ways to solve this problem is through the creation of flexible scripts to allow, if necessary a quickly adapt to the changes occurred during the execution of the test. Techniques such as Keyword-driven and data-driven, has been applied with the aim of making the Scripts more maintainable and reusable (Fewster, 1999).

The solution adopted by the Project CIn / Motorola to resolve this situation, is to dedicate a member of the test team and part of time to running the tests for the adaptation of the script. This solution, adopted to try ensure that the script is in accordance with the changes caused by the different builds, and if applicable, migrate the Script to another platform where testing should be performed.

Therefore, this work presents the development of a prototype for testing setup automation. Given the need for greater accuracy in testing configuration procedures. Regarding test scripts, the develop of this prototype, aims reduce the need to translate the scripts, through techniques such as Keyword-driven and use of technologies, as Python, Java and Android. The technologies adopted for the development of the prototype, allow the adoption of the tool by the various platforms used in the Project CIn / Motorola environment, and provides an architecture that enables greater granularity and better match the frequent changes in the builds of the devices under test.

The development of this tool was based on a survey conducted with collaborators of the Project CIn/Motorola, which sought to observe what kind of test, a wrong configuration would cause greater damage, inaccuracy or consumption of time. The data showed the CTS (Confirmation …), as the test that small mistakes in the test setup would cause the loss of an entire execution, since the device is configured the test running lasts about six (6) hours.

Initially, this proposal covers only the configuration of CTS tests. The initial modules of the prototype, were implemented in a flexible architecture that allows the addition of new modules for the future other types of tests are aggregated to this solution. Only the GUI, need to be reformulated for later versions, because it has been designed only for the initial needs of the development of the tool.

The contributions of this work for the environment of the Project CIn / Motorola include;

* Increase the ability to run tests: Reducing time spent in device configuration;
* Increase the precision in the results;
* Avoid rework: Through a multi-platform solution no will be necessary to update or rewrite the test scripts

This paper is organized as follows, Section 2 contains the theoretical review and a survey of the techniques used in this work. Section 3 describes the development of the prototype, showing the phases of the process, how was the survey of the development of the tool, relevant data as well as all the diagramming the use cases and a description of the project manage. Section 4 presents the results obtained in this work, Section 5 provides the conclusion and final remarks.

2. Theoretical Referential

2.1 Software Testing

According to Pressman (2006), software testing can be described as a set of activities for the verification and validation of software, these can be planned in advance and conducted systematically. To Sommervile (2007), the test is a dynamic technique for verification and validation, which involves running a program with a set of input data and verify that conforms to the expected result. However verify and validate have different meanings, the verification attests if the application is in according to their specification, in turn, validation verify if the results is in according to expected result by the clients.

To Pezze and Young, 2008, to evaluate the degree to which a software system really meets your needs, in order to meet the actual needs of the user, is called validation.

Validation against real requirements, necessarily involves human judgment and a potential for ambiguity, misunderstanding or disagreement. The specification must be sufficiently precise and unambiguous to the point that there can be no disagreement about the acceptability of a given system behavior (Pezze and Young, 2008). Meet requirements is not the same as being in accordance with its specification. Entende se como especificação uma declaração sobre uma solução particular apresentada por um problema, sendo a solução proposta pode ou não atingir seu objetivo.

To Delamaro et. al. 2007 the activities of verification and validation must be conducted throughout the process of software development, starting from its conception, these activities are being divided into static or dynamic. The statics activities, not require the execution or even the existence of an application or executable model to be driven however the dynamics are based on the execution of an application or model. Review, inspection, reading techniques among others, are considered static verification and validation activities. The dynamic assessment, focuses on evaluate the behavior of the software at runtime, in order to prove the non-conformities in the results presented.

The literature presents some definitions and terms used in software testing, according Delemaro et. Al 2007 are:

Fault: Process or definitions of incorrect data;

Mistake: Human action who produce an incorrect result;

Error: During execution of the software, is characterized by a inconsistent or unexpected;

Failure: The produced results are different of the expected.

The IEEE pattern (IEEE STD. Glossary of Software Engineering Terminology, padrão 610.12/1990) define that the problems introduced by the developer are called faults (fault). Mistakes can be committed both in the specification and in the system code. When a fault is activated during execution of the software, an error is generated. If the problem manifests at the frontier of system, a failure occurs, which can be perceived by the user.

As Pressman (2006),the presence of defects indicates faults in software, can consider cases of successful tests, which reveal failures that have not been discovered. The tests do not guarantee the absence of failures, if no faults occur during testing, does not mean that the system do not contain faults.

The tests has as main objective to find bugs before the system is delivered to the customer, since the later these faults are found, the more expensive is the cost to repair these bugs (RIOS & Moreira Filho, 2006). According to Pressman (2006) Software testing aims to ensure greater coverage of verification and validation in system functionality and a higher likelihood for detecting errors.

The testing activities are complex, since many factors influence, this way the tests are divided into phases with different goals: unit testing, integration testing and system testing. (Delamaro et. Al. 2007).

Unit Testing: Is the smallest unit of software that can be executed. The test unit checks if a portion of the code performs its function adequately, in isolation from the rest of the system. The unit testing is recommended as the first test to be performed in software, as a software as a whole is composed of parts (units) and these units shows defective, software will not work. So the effort is concentrated on these smaller units and only after the unit test follows incrementally to the next phases.

Integration testing: This is performed after the units are individually tested, the emphasis is on the construction of the system structure. The connections between the elements are built through communication interfaces of the units. The main objective of integration testing is to verify that the interaction between the units operate properly and does not lead to errors. Integration tests are usually conducted in descending order \ top-down and (or) bottom-up \ button-up.

System Test: The goal is to verify that the specific features in the requirements documents are all fully implemented. Aspects of correctness, completeness and consistency should be explored, as well as non-functional requirements such as security, performance and robustness.

In addition to the tests mentioned above, there is also the Regression Testing, which are performed during software maintenance. Every change made to the system, after its release, there is a risk of new defects be introduced. Due to modifications is necessary to conduct tests that validate the new and previous requirements.

The importance of automated generation of test cases is shown in Lindlar et al. (2010), which states that some activities performed manually, as the projection of the test cases, data selection and evaluation of the tests, require and consume a significant amount of time, and these activity could be performed automatically, guaranteeing best quality, as would, avoid errors caused by the testator, they enable the tests to be performed more frequently and with greater advance.

2.1.1 Functional Testing

Functional tests belong to the black-box approach, they are designed using the specification and not the system code (RIOS and Moreira Filho, 2006). These are designed to verify the functionality of the application, without worrying about the logic and methods used in testing system (RIOS and Moreira Filho, 2006).

This type of test is used to ensure that the system behavior conforms to the requirements specification, and the focus of this type of test is in the appropriate inputs and outputs for each function, Where, all features of the system must be tested (Burnstein, 2002).

According yet with Burnstein (2002) is important to ensure that the system is dealing with inadequate inputs and unexpected. For both, the functional tests should cover these cases, and other that are specified in the requirements.

2.1.2 Test Cases

Test cases consist in a sets of tests to run, aimed at ensuring a higher likelihood for detecting errors in the system under test (Pressman, 2006).

As described by Perry (2006), a test case is defined by a set of test inputs, execution conditions, and expected results to reach a goal of a particular test.

The test cases are compound by some information, presented in Burnstein (2002), as the set of test inputs, consisting of data received via an external source, such as a software, hardware or a human being; conditions for the tests, as a particular configuration of a device, or an entry in the database, and the expected results, which are the values ​​to be generated by the code being tested after insertion of the input data previously determined.

2.1.3 MTB - Model Based Testing

Model-based testing consist in a test generated from the model that describes the expected behavior for the software or part thereof, using as a reference the method of black box testing (JACKY, 2008). Still according Jacky (2008), this practice is used for the automatic generation of test cases, in which is utilized a formal and functional model of the system in testing (SUT - System under test).

According to Reza and Lande (2010), model-based testing can be described as a testing technique, which is possible from requirements and system behavioral models, automatically generate test cases.

As the model-based test is generated from the system specification, is possible start the testing process after the requirements have been defined. Moreover, another benefit gained from the model-based testing is the cost reduction for the generation of tests, since using this technique acquires a reduction of time spent on creating of these during the test cycle (REZA and Lande 2010).A técnica MBT faz uso de algumas atividades para a geração do teste.

Besides to building the model used to create tests from the requirements, it is necessary to identify what should be tested, understand the software and its environment, as well as enumerate inputs and outputs that will be used during testing (EL-FAR and WHITTAKER , 2001). For each generated input, are creating new scenarios to be tested. These will generate outputs, that should be checked if correspond to what is expected by the system (EL-FAR and WHITTAKER, 2001). As Benavente (2006), this result will indicate the verdict of the test, indicating whether it passed or failed.2.1.4 Teste de Regressão

The Regression testing can be conducted manually re-running a subset of all test cases or using automated tool of capture / replay. Tools capture / replay enables the software engineer to capture test cases and results for subsequent replay and comparison (Pressman, 2006).

The Regression Testing represent a more specific step. They aim to ensure that all the effort of tests already carried out in relation to software artifact continue providing quality assurance after any amendment thereto (SOMMERVILLE, 2003).

According to Pezze and Young (2008), regression testing is used to support the testing activities and ensure the achievement of appropriate quality through several versions of software.

When new or modified components, embedded in software, causing defects to other components not modified, it can be stated that the system under test regressed (BINDER 2000). Therefore, the tests applied to the new system, containing the amendments are called "regression testing", seeking to avoid the regression of the system.

2.1.5 Automation testing techniques

To Fewster (2001), the technique, record & playback consist in use a test automation tool to record the actions performed by the user, who interacts with the graphical user interface and converts the actions in the test scripts that can be executed as many times needed. This technique is considerably simpler and practical.

The technique of scripts, is considered an extension of the technical record & playback. Through programming, the recording test scripts, are changed so that they perform a different behavior of the original script during its execution (Hendrickson 1998).

In comparison with the technique record & playback, the technique of scripts has higher reuse rate, longer life, better maintenance and greater robustness of test scripts (Tervo 2001).

The Keyword-Driven technique consists in extracting of the test scripts the test procedure that represents the execution logic. Test scripts shall contain, only the specific test actions on the application, which are identified by keywords. These test actions, works like a program function, and may also have parameters, which are activated by the key words, from the execution of different test cases. The test procedure is stored in a separate file, in form of an ordered set of keywords and parameters (Fewster 1999).

2.2 UML – Unified Modeling Language

As described in Alhir (2002), UML is a language for specifying, visualizing, constructing, and documenting the artifacts of a system process. This consists of a standard for creating models, being flexible and independent of programming languages ​​(PENDER, 2002; LIMA, 2011).

Due a use a standard notation, the system can be represented by models generated using any methodology or programming language (LIMA, 2011). The UML are composed by diagrams that describe the system, through models and is used to design object-oriented systems. These models consist of a set of ideas that include information necessary to understand and eliminate any irrelevant information or that might hinder the understanding of the system (ALHIR, 2002).

UML was created as a standard graphical notations for OMG2 in 1997 and has been used until now, currently UML is in the second version (Larman, 2001).

2.3 OOP – Object Oriented Programing

The concept of object orientation, makes up part of our understanding and interaction with the world in which we live (Dall'Oglio, 2007). Formalizes a vision of the real world within which the system is developed, setting objects as organizational structure of the system and transforms a given set of collaboration in the execution of the work.

As Yourdon and Argili (1999), object is an independent entity, asynchronous and concurrent , stores data, encapsulates services, exchange messages with other objects and is modeled to perform the methods of the final system. The object orientation provides a direct mapping between the real world (concepts) and the units of organization used in the project (code).

Among the languages ​​supporting OO will include: Samlltalk, Pearl, Python, PHP, Java, C # Net and others..

2.3.1 Phyton Language

According Summerfield (2012) Python is a multi-platform language, runs on Windows and Unix derivatives such as Linux, BSD, and Mac OS X. Python can be used in procedural programing, object-oriented, functional programming, but its main feature is related to object orientation.

The language has as characteristic, object-oriented, dynamic typing and interpreted and interactive. Presents a clear and concise syntax that facilitates the readability of source code, making it the most productive language (Borges, 2010).

As described in Menezes (2010), the language is interpreted through bytecodes by the Python virtual machine, making the code portable. This makes it possible to build applications on one platform and run on other systems or run direct from the source.

Besides being used as the main language in the development of systems, Python is also widely used as a scripting language in various software, enabling you to automate tasks and add new features (Borges, 2010).

2.3.2 The Java Language

Gonçalves (2007) defines Java as a programming language, object-oriented, that can be used for desktop applications, web and mobile devices. Unlike of the conventional languages ​​that are compiled to native code, the Java language is compiled into a bytecode that is executed by a virtual machine.

For Thompson (2005) the main characteristics of this language are: (i) encapsulation: classes and methods in Java can be created so as not to be visible to the classes with which there is no interest in sharing data. (ii) inheritance: a class already created will create other classes with the same characteristics, but with different attributes. (iii) Polymorphism: the ability of two objects of different classes, implementing the same method. This can happen through an inheritance or interface;

2.5 Uiautomator

The Android SDK provides two tools to support user interface test. The test user interface ensures that your application returns a correct output in response to a sequence of actions such as: (i) keyboard input, (ii) the toolbar (iii) menus, (iv) dialogs; (v) images, and other user interface controls.

The available tools are:

UiautomatorViewer which is a GUI tool to scan and analyze the components of an Android application interface. Provides a convenient visual interface to inspect the hierarchy layout and view the properties of the individual components of the user interface that is displayed in the test device.

Uiautomator, which is a Java library that contains APIs to create funcinionais testing the customized user interface, and an implementation mechanism to automate and run the tests.

2.6 Unifed Procces

Unified process tends to develop a software system through a set of customer requirements using a set of activities. Unified process can also be defined as a generic process framework that can be customized by adding or removing activities based on specific needs and available resources for a project (Scott, 2003).

2.6.1 Driving by Use Cases

Use cases are sequences of actions, where the system itself or the actors perform, generating results value for one or more actors. One of the main attributes of the unified process is the use of use cases in their development.

According to Sccott (2003), use cases are important to find requirements, elaborate the analysis, design and implementation for the following reasons: (i) are stated from the perspective of users. (ii) expressions are simple. (iii) they are intuitive and in the language of the customer. (iv) Increase considerably the understanding of the actual system requirements. (v) achieve a high degree of tracking requirements arising from further development. (vi) simplify the decomposition of requirements which allocate work to subteams which helps in project management.

2.4.2 Phases of the Unified Process

To Booch, Rumbaugh, and Jacobson (2000) the unified process consists of four phases, according to the authors, a phase is the period of time between two major milestones of progress of the process in which a set of well-defined goals is achieved. The main phases of the Unified Process are: (i) Conception, in this phase the business case is established and defined scope for the project. The business case should include success criteria, risk assessment, definition of resources and a plan defining the milestones of progress. At the end of this phase we define the feasibility of proceeding with the project. (ii) development, the project plan and the architecture are defined. The objectives of this phase are analyzing the problem domain, establish the foundation of a solid architecture, the development of the project plan and the elimination of the higher risk elements of the project (BOOCH, RUMBAUGH e JACOBSON: 2000). As is necessary an understanding of the entire system is needed to define the architecture, it is necessary to drill the majority of its requirements. (iii) construction: is the development of the system. This stage is developed a complete system, from the description of the remaining requirements and acceptance criteria for the development itself, implementation, and finally by testing the software. After analyzing the software, the environment and users are ready to become operational. (iv) Transition - when the system is delivered to end users.

After the initial use of software by users, may arise needs for further developments. As Booch, Rumbaugh, and Jacobson (2000), this phase usually starts with a beta version of the system that is later replaced by the production system. At the end of this phase it is decided whether the objectives of the project life cycle were achieved and determines if another development cycle should be started.

2.7 Design Patterns

Design patterns are specific approaches used to solve generic problems of programming systems which can be fully adapt or according to the needs of your solution.

Patterns fit well when you need to change an architecture to improve some aspects related to performance, scalability, reuse and maintenance. The majority of standards have been devised to such a development. Often, a inadequate implementation, indicates the application need of patterns, and refactoring technique, demonstrates how to solve recurrent problems. (ALUR, 2004).

Design patterns can be divided by their function or scope, are presented in three main categories: Patterns of Creation, Structural and Behavioral.

By using patterns, there is the understanding that all patterns have been previously tested and implemented and that perfect communication between developers, unifying the language is spoken among those involved in the project.

4. Results

Every day, many products must be tested to be validated. In this particular case, these products are the hardware and the software applied to a platform, the Android operating system. These versions are called System Builds. After a system update or change the build is necessary to conduct tests in order to find previously compatibility issues between devices.

Front of product launches and especially OS versions, which will be launched on the world market, known as builds candidates, a gradual demand test execution occurs. The test involves an effort to set the device to the initial state of execution and, thereafter, initiate a series of test cases that last for approximately 6 hours to complete.

The error rate generated after implementation has to be zero, and in some cases known errors are accepted, but often there is a higher error rate, having to redo the tests on the errors that have been identified. The errors that appear are caused by many reasons such as: (i) network inaccessible, (ii) limited data plan, (iii) invalid initial state among others. Due part of the initial device configuration be performed manually, several problems can occur due to invalid or nonstandard configurations, which leads to an increased error rate.

Given the above scenario, after implementation of the system and seeking to validate the study, with the aim to simulate the input data through different versions of builds, as well as its application in different product lines.

The load data test initial of CTS has eight automation processes: (i) SetupScreen, (ii) Wifi (iii) timeout, (iv) Security (v) DeveloperOptions (vi) Location, (vii) and Admin (viii) Gmail.

It is possible to observe that a critical point when performing data load testing is Wifi connection, diagnosed as an internal problem of data network currently in the project network. To validate the test, the setup was applied in 2 products, the Jelly Bean operating system version 4.3, applied in 7 versions of builds in 3 continetes.

FIGURA X - Setup de Testes

Fonte: Autores (2013)

According to the results in the graphic below, the tests showed valid results, observing the tests that depend on the network for data connection (Wifi and Gmail) in mostly failed, due the network conditions vary greatly between criticism (with much access) and not critical (without much access or reasonable access). In environments where the network does not present critical state, the tests pass. Another relevant point is the build-16\_62 14.10.0Q3.X Europe, which showed a different initial configuration screen, thus invalidating the model developed. The screen layout is a factor aggravating, as when it is changed or a new layout appears in new builds, the code will have to be adapted.

GRÁFICO 2 - Validação da Ferramenta de Testes CTS

Fonte: Autores (2013)

In its entirety, 63 tests were performed to validate the tool, and of these 16 came to fail, and the rest passed. From the collection of these data, the tool is now available for use in the Project CIn / Motorola, requiring only improve network conditions of the environment.

Taking into account the current execution time of the CTS test case manually, being approximately 13 minutes and 50 seconds and the automation of tests takes 08 minutes and 05 seconds to setup, with a gain of 59.62% in the time of initial setup.

GRÁFICO 3 - Resultados tempo de carga

Fonte: Autores (2013)

During the automation process, we can observe, the facility that the test engineers with different profiles had to use the proposed tool. This was possible because the real time spent in the initial execution of the script and executions with hits of automated tests was lower than the time estimated for these same activities, previous experiences of automation, when other tools were used and needed the intervention of an implementer to advance the automation of tests, freeing the testers to perform other activities during the initial setup.

6. Conclusão

Systems development covers issues such as productivity, portability, quality, documentation and other which makes it a complex task that requires technical knowledge, organization, creativity and intense communication between teams. That occurs during development some of these requirements fails; being unpredictable moment when the failure occurred. Checking this set of variables becomes Necessary there's an easy and quick way to run a test scenario at any time, and this is feasible with the help of automated testing.

The automation of testing presents, more security of test execution, allows you to create more elaborate and complex tests, which are identically repeated numerous times.

Another factor considered relevant, automation increases the amount of time spent on checking the system and decreases the time spent on the identification and correction of errors. By isolating the tests of artesal manufacturing, in other words, manually tests can be performed at any time and, consequently, the errors tend to be found earlier.

Despite the automated tests help create a cohesive and less coupled modeling of the system, its main purpose is to verify the quality of different features on the devices.

The process of test automation is complex, which is error-prone and requires maintenance, factor by which, is most important that the batteries of tests be high quality, ie, organized, readable, fast among others. Which requires knowledge of best practices, patterns, anti-patterns and indications of problems.

Suggestions for future work: (i) improve the tool, by implementing new features, creating reports of test execution thereby having greater utility for documentation, (ii) adapting the testing tool with data persistence, (iii) Create metrics of standards quality, thereby obtaining the number of defects with the amount of performed tests, (iv)

The development of the tool, as well as during the course together with the work performed in laboratory, was possible to define steps used in the evaluation and implementation of the improvement of the test procedures. Also enabling theoretical approaches in software testing activities related to this activity.