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).

Em comparação com a técnica record & playback, a técnica de scripts apresenta maior taxa de reutilização, maior tempo de vida, melhor manutenção e maior robustez dos scripts de teste (Tervo 2001).

A técnica Keyword-Driven consiste em extrair dos scripts de teste o procedimento de teste que representa a lógica de execução. Os scripts de teste passam a conter apenas as ações específicas de teste sobre a aplicação, as quais são identificadas por palavras-chave. Estas ações de teste são como funções de um programa, podendo inclusive receber parâmetros, que são ativadas pelas palavras-chave a partir da execução de diferentes casos de teste. O procedimento de teste é armazenado em um arquivo separado, na forma de um conjunto ordenado de palavras-chave e respectivos parâmetros (Fewster 1999).

2.2 UML – Unified Modeling Language

Conforme descrito em Alhir (2002), a UML é uma linguagem para a especificação, visualização, construção e documentação de artefatos do processo de um sistema. Essa consiste em um padrão para a criação de modelos, sendo flexível e independente de linguagens de programação (PENDER, 2002; LIMA, 2011).

Por utilizar uma notação padrão, o sistema pode ser representado pelos modelos gerados utilizando qualquer metodologia ou linguagem de programação (LIMA, 2011). A UML é composta por diagramas, que descrevem o sistema através de modelos, sendo utilizada para projetar sistemas orientados a objetos. Esses modelos são formados por um conjunto de ideias que incluem informações necessárias para o entendimento e eliminam qualquer tipo de informação irrelevante ou que possa vir a dificultar o entendimento sobre o sistema (ALHIR, 2002).

A UML foi criada como um padrão de notações gráficas pela OMG2 em 1997 e tem sido utilizada até hoje, encontrando-se, atualmente, na segunda versão (LARMAN, 2001).

2.3 POO - Programação Orientada a Objetos

O conceito de orientação a objetos compõe parte de nosso entendimento e interação com o mundo em que vivemos (DALL’OGLIO, 2007). Formaliza uma visão do mundo real dentro do qual o sistema está desenvolvido, estabelecendo os objetos como estrutura organizacional do sistema e transforma a colaboração de um dado conjunto na execução do trabalho.

Conforme Yourdon e Argili (1999), objeto é uma entidade independente, assíncrona e concorrente, armazena dados, encapsula serviços, troca mensagens com outros objetos e é modelado para executar os métodos finais do sistema. A orientação objeto fornece um mapeamento direto entre o mundo real (conceitos) e as unidades de organização utilizadas no projeto (código).

Entre as linguagens de dão suporte á orientação destacam-se: Samlltalk, Pearl, Python, PHP, Java, C#.Net entre outras.

2.3.1 Linguagem de Programação Python

Conforme Summerfield (2012) Python é uma linguagem multiplataforma, executado em Windows e em sistemas derivados de do Unix, como Linux, BSD, e de Mac OS X. Python pode ser usado para programar em procedural, orientado a objetos e para extensões menores, em estilo funcional, mas sua principal característica esta relacionada à orientação a objetos.

A linguagem tem como característica, very high level language, orientada a objeto, de tipagem dinâmica e forte, interpretada e interativa. Apresenta uma linguagem clara e concisa, que favorece a legibilidade do código fonte, tornando a linguagem mais produtiva (BORGES, 2010).

Conforme descrito em Menezes (2010), a linguagem é interpretada através de bytecode pela máquina virtual Python, tornando o código portável. Com isso é possível compilar aplicações em uma plataforma e rodar em outros sistemas ou executar direto do código fonte.

Além de ser utilizado como linguagem principal no desenvolvimento de sistemas, o Python também é muito utilizado como linguagem script em vários softwares, permitindo automatizar tarefas e adicionar novas funcionalidades (BORGES, 2010).

O Python não possui suporte nativo para a programação da Graphic User Interface, apresenta bibliotecas que possuem GUI. A biblioteca padrão Tcl\TK-Tcl, é uma linguagem script de sintaxe, já o Tk é uma biblioteca GUI, escrita em Tcl e C, sendo esta instalada como padrão , junto ao Python (SUMMRFIELD, 2012).

2.3.2 Linguagem de Programação Java

Gonçalves (2007) define Java como uma linguagem de programação orientada a objeto, sendo sua aplicação para desktop, web e dispositivos móveis. Diferentemente das linguagens convencionais, que são compiladas para código nativo, traduz que a linguagem Java é compilada para um bytecode que é executado por uma máquina virtual.

Para Thompson (2005) as principais características desta linguagem são: (i) Encapsulamento: as classes e métodos em Java podem ser criados de maneira a não ficarem visíveis às classes com as quais não haja interesse de compartilhamento de dados. (ii) Herança: uma classe já criada irá criar outras classes com as mesmas características, porém com atributos distintos. (iii) Polimorfismo: é a capacidade de dois objetos, de classes distintas, implementarem o mesmo método. Isso pode acontecer por herança ou por interface;

2.5 Uiautomator

O SDK do Android fornece duas ferramentas para apoiar o teste de interface de usuário. O teste de interface de usuário garante que seu aplicativo retorne uma saída interface correta em reposta a uma sequência de ações como: (i) entrada de teclado; (ii) barra de ferramentas; (iii) menus; (iv) caixas de diálogos; (v) imagens e outros controles de interface do usuário.

As ferramentas disponibilizadas são:

UiautomatorViewer que é uma ferramenta GUI para digitalizar e analisar os componentes de interface de uma aplicação Android. Fornece uma interface visual conveniente para inspecionar a hierarquia de layout e visualizar as propriedades dos componentes de interface do usuário individuais que são exibidas no dispositivo de teste.

Uiautomator, que é uma biblioteca Java que contem APIs para criar testes funcinionais da interface do usuário personalizada, e um mecanismo de execução para automatizar e executar os testes.

2.6 Processo Unificado

Processo unificado tende a desenvolver um sistema de software através de um conjunto de requisitos do cliente utilizando um conjunto de atividades. Processo unificado também pode ser definido como uma estrutura genérica de processo que pode ser customizado adicionando-se ou removendo-se atividades com base nas necessidades específicas e nos recursos disponíveis para um projeto (SCOTT, 2003).

2.6.1 Dirigido por caso de uso

Casos de uso são sequências de ações, onde o próprio sistema ou atores as executam, gerando resultados de valor para um ou mais atores. Um dos principais atributos do processo unificado é a utilização de casos de uso no seu desenvolvimento. A expressão “dirigido por casos de uso” refere-se ao fato de se utilizar os casos de uso para dirigir todo o trabalho de desenvolvimento, desde a captação inicial e negociação de requisitos até a aceitação do código (SCOTT, 2003).

Conforme Sccott (2003), casos de uso são importantes para encontrar requisitos, elaborar a análise, projeto e implementação, pelas seguintes razões: (i) São demonstrados sob a ótica dos usuários. (ii) Apresentam expressos de maneira simples. (iii) São intuitivos e estão na linguagem do cliente. (iv) Melhoram consideravelmente o entendimento dos requisitos reais do sistema. (v) Atingem um alto grau de rastreamento de requisitos que resultam do desenvolvimento posterior. (vi) Simplificam a decomposição dos requisitos que permitem alocar trabalho a subequipes o que ajuda na gerência do projeto.

2.4.2 Fases do Processo Unificado

Para Booch, Rumbaugh e Jacobson (2000) o processo unificado é composto por quatro fases, segundo os autores, uma fase é o período de tempo entre dois importantes marcos de progresso do processo em que um conjunto de objetivos bem definidos é alcançado. As principais fases do Processo Unificado são: (i) Concepção; nesta fase o caso de negócio é estabelecido e o escopo delimitado para o projeto. O caso de negócio deve incluir critérios de sucesso, avaliação de riscos, definição de recursos necessários e um plano definindo os principais marcos de progresso. Ao fim desta fase se define a viabilidade de se prosseguir com o projeto. (ii) Elaboração; são definidos o plano de projeto e a arquitetura. Os objetivos desta fase são análise do domínio de problema, o estabelecimento da fundação de uma arquitetura sólida, o desenvolvimento do plano do projeto e a eliminação dos elementos de mais alto risco do projeto.(BOOCH, RUMBAUGH e JACOBSON: 2000). Como é necessária uma compreensão de todo o sistema para definir a arquitetura, se faz necessário o detalhamento da maioria de seus requisitos. (iii) Construção; consiste no desenvolvimento do sistema. Nesta fase é desenvolvido um sistema completo, passando pela descrição dos requisitos restantes e de critérios de aceitação, pelo desenvolvimento em si, pela implantação e, finalmente, pelos testes do software. Ao fim se analisa se o software, o ambiente e os usuários estão prontos para se tornarem operacionais. (iv) Transição - quando o sistema é entregue aos usuários finais. Após o início da utilização do software por seus usuários, podem surgir necessidades de desenvolvimentos adicionais. Conforme Booch, Rumbaugh e Jacobson (2000), esta fase normalmente se inicia com uma versão beta do sistema que, posteriormente, é substituída pelo sistema de produção. No final desta fase se decide se os objetivos de ciclo de vida do projeto foram alcançados e se determina se outro ciclo de desenvolvimento deve ser iniciado.

2.7 Design Patterns

A palavra “pattern” como uma forma de padronizar soluções para problemas comuns foi utilizada pela primeira vez na área de arquitetura e engenharia civil por Christopher Alexander, para solucionar problemas que costumavam acontecer corriqueiramente (FRANTZ, 2011).

Um pattern nasce em decorrência de um problema e solução recorrente do que acontece no ambiente de trabalho do desenvolvedor. A partir do momento que pessoas envolvidas conseguem visualizar uma solução para o problema, e conseguem promover uma reutilização desta solução, compartilhando com outras pessoas este conhecimento, acaba surgindo um novo pattern (ALUR, 2004).

Os padrões de projeto são abordagens específicas, utilizadas para resolver problemas genéricos dos sistemas de programação dos quais podem se adaptar integralmente ou de acordo com as necessidades de sua solução.

Padrões se encaixam bem quando é necessário alterar uma arquitetura para melhorar alguns aspectos relacionados à performance, escabilidade, reuso e manutenção. A maior parte dos padrões foram construídos visando este tipo de evolução. Muitas vezes a implementação inadequada, indica a necessidade de aplicação de padrões, e a técnica de refatoramento, demostra como solucionar problemas recorrentes. (ALUR, 2004).

Padrões de projeto podem ser divididos por sua função ou escopo, sendo apresentados em três categorias principais: Padrões de Criação, Estruturais e Comportamentais.

Ao utilizar a padronização, há entendimento de que todos os padrões já foram previamente testados e implementados e, que aperfeiçoe a comunicação entre os desenvolvedores, unificando a língua que é falada entre os envolvidos no projeto.