UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ
BACHARELADO EM ENGENHARIA DE SOFTWARE

ANTONIO GERBSON DA SILVA LIMA

AUTOMAÇÃO DE TESTES FUNCIONAIS EM AMBIENTES WEB: UM ESTUDO DE CASO NO LABORATÓRIO DE SISTEMA E BANCOS DE DADOS DA UNIVERSIDADE FEDERAL DO CEARÁ

QUIXADÁ
2014

ANTONIO GERBSON DA SILVA LIMA

AUTOMAÇÃO DE TESTES FUNCIONAIS EM AMBIENTES WEB: UM ESTUDO DE CASO NO LABORATÓRIO DE SISTEMA E BANCOS DE DADOS DA UNIVERSIDADE FEDERAL DO CEARÁ

Trabalho de Conclusão de Curso submetido à Coordenação do Curso de Bacharelado em Engenharia de Software da Universidade Federal do Ceará, como requisito parcial para obtenção do grau de Bacharel.

Área de concentração: computação

Orientadora: Profa Dra Paulyne Matthews Jucá

QUIXADÁ
2014

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca do Campus de Quixadá

L699a Lima, Antonio Gerbson da Silva
Automação de testes funcionais em ambientes Web: um estudo de caso no Laboratório de Sistemas e Bancos de Dados da Universidade Federal do Ceará / Antonio Gerbson da Silva Lima. – 2014.
59 f. : il. color., enc. ; 30 cm.

Monografia (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Bacharelado em Engenharia de Software, Quixadá, 2014.
Orientação: Profa. Dra. Paulyne Matthews Jucá
Área de concentração: Computação

1. Banco de dados da Web 2. Software - Testes 3. Sistemas de computação - Tecnologia da Informação I. Título.

CDD 005.30287

ANTONIO GERBSON DA SILVA LIMA

AUTOMAÇÃO DE TESTES FUNCIONAIS EM AMBIENTES WEB: UM ESTUDO DE CASO NO LABORATÓRIO DE SISTEMA E BANCOS DE DADOS DA UNIVERSIDADE FEDERAL DO CEARÁ

Trabalho de Conclusão de Curso submetido à Coordenação do Curso Bacharelado em Engenharia de Software da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Bacharel.

Aprovado em: 12 de Junho de 2014.

BANCA EXAMINADORA

_____
Profa Dra Paulyne Matthews Jucá (Orientadora)
Universidade Federal do Ceará-UFC

_____
Prof. Msc. Carlos Diego Andrade de Almeida
Universidade Federal do Ceará-UFC

_____
Prof. Msc. Camilo Camilo Almendra
Universidade Federal do Ceará-UFC

AGRADECIMENTOS

A Deus, por ter me concedido saúde e sabedoria para me manter firme na busca de meus sonhos.

A meus pais, que sempre me apoiaram na conquista dos meus objetivos.

A minha família, em especial a minha tia Gerlânia, um modelo de determinação e conquista no qual me inspiro até hoje.

Aos amigos da Escola João Araújo Torres, por contribuírem para minha Educação e acesso a Universidade

Aos meus amigos de graduação que me ajudaram nos momentos de dificuldade e partilharam dos momentos de superação

A todos do Laboratório de Sistemas e Bancos de Dados da UFC, em especial a Profa. Roselia, Prof. Javam, Rodrigo e Rebeca pela confiança e incentivo na realização dessa pesquisa.

A Profa. Paulyne, pela amizade e apoio durante toda a graduação, especialmente durante a realização deste trabalho.

Um sincero,
Muito Obrigado.

“Nada no mundo se compara à persistência. Nem o talento; não há nada mais comum do que homens malsucedidos e com talento. Nem a genialidade; a existência de gênios não recompensados é quase um provérbio. Nem a educação; o mundo está cheio de negligenciados educados. A persistência e determinação são, por si sós, onipotentes.”
(Calvin Coolidge)

RESUMO

O competitivo mercado de software demanda cada vez mais que as organizações desenvolvam produtos de qualidade, em meio a limitações de prazo e recursos. Diante desse cenário, é cada vez mais evidente a busca das empresas por métodos e ferramentas que agilizem o processo de desenvolvimento e garantam uma maior qualidade dos sistemas. Nesse contexto, a área de automação de testes tem ganhado destaque, devido à agilidade e a qualidade que essa atividade pode trazer aos processos de testes das empresas. Tendo como base esse potencial, o presente trabalho apresenta um estudo de caso que aborda a automação de testes funcionais em um sistema web, desenvolvido no Laboratório de Sistemas e Bancos de Dados da Universidade Federal do Ceará. O estudo envolveu a definição de um processo de desenvolvimento e execução de testes automatizados baseado no Modelo de Melhoria do Processo de Teste Brasileiro – MPT-Br, utilizando a ferramenta Selenium e a aplicação desse processo em um projeto web desenvolvido no laboratório. A avaliação do estudo de caso foi feita com base no impacto das atividades de automação no processo de teste e na qualidade do sistema. Como principais contribuições estão a definição do processo de testes e o desenvolvimento do projeto de testes automatizados e como principais resultados percebidos estão a maior agilidade na execução dos testes e um aumento na identificação de falhas que permitiu uma melhoria da qualidade do software desenvolvido.

Palavras chave: Teste de Software, Automação de Testes, MPT-Br, Selenium.

LISTA DE QUADROS

Quadro 1 – Classificação dos testes segundo o modelo FURPS 17
Quadro 2 – Classificação dos casos de teste segundo o critério importância 38
Quadro 3 – Classificação dos casos de teste segundo o critério tempo de execução manual ..38
Quadro 4 - Classificação dos casos de teste segundo o critério repetitividade 39
Quadro 5 - Classificação dos casos de teste segundo o critério necessidade de intervenção manual 39
Quadro 6 - Classificação dos casos de teste segundo o critério estabilidade 40
Quadro 7 - Classificação dos casos de teste segundo o critério necessidade de múltiplos valores de dados 40
Quadro 8 - Classificação dos casos de teste segundo o critério necessidade de múltiplos ambientes 40

LISTA DE ILUSTRAÇÕES

Figura 1: Erro x Defeito x Falha 14
Figura 2: Modelo V relacionando as atividades de desenvolvimento e teste de software 16
Figura 3 - Representação do Selenium Webdriver 28
Figura 4: Processo de testes manuais adotado no Projeto PTDH 34
Figura 5: Processo de testes manuais em conjunto com as atividades do processo de testes automatizados 36
Figura 6: Script gerado pelo Selenium IDE para o caso de teste Registrar Usuário 43
Figura 7: Script gerado com o Selenium IDE e exportado para Java 44
Figura 8: Estrutura do projeto de testes automatizados em Java 44
Figura 9: Caso de teste em Java 45
Figura 10: Resultados da execução dos testes com Junit 46
Figura 11: Resumo da execução dos casos de teste 48
Figura 12: Histórico de execução de um caso de teste 48
Figura 13: Matriz de execução dos testes ao longo de várias releases 49
Figura 14: Métricas da execução de testes automatizados 50
Figura 15: Gráfico com o número de casos de teste executados antes e após a automação 53
Figura 16: Gráfico com o nº de falhas encontradas na execução manual x nº de falhas encontradas na execução automatizada 54

SUMÁRIO

1 INTRODUÇÃO 11

2 REVISÃO BIBLIOGRÁFICA 13

2.1 Teste de software 13
2.1.1 Níveis de testes de software 15
2.1.2 Tipos de teste de software 16
2.2 Automação de testes 20
2.2.1 Benefícios da automação de testes 20
2.2.2 Desafios da automação de testes 21
2.2.3 Processo de automação de testes 22
2.2.4 Técnicas de automação de testes 25
2.3 Ferramentas de apoio à automação de testes 26
2.3.1 Selenium 27
2.3.2 Junit 28
2.3.3 Testlink 29

3 PROCEDIMENTOS METODOLÓGICOS 30

4 ESTUDO DE CASO 32

4.1 Contexto do Estudo de Caso 32
4.2 Processo de testes manuais adotado no projeto 33
4.3 Processo de automação de testes 35
4.3.1 Definição dos objetivos da automação 37
4.3.2 Seleção de casos de teste para automação 37
4.3.3 Desenvolvimento dos casos de teste automatizados 43
4.3.4 Execução dos testes automatizados 46
4.3.5 Verificação e correção de defeitos nos testes automatizados 47
4.3.6 Visualização de resultados dos testes 47
4.3.7 Geração de relatórios e métricas 48
4.3.8 Verificar aderência aos objetivos da automação 50

5 DIFICULDADES E LIÇÕES APRENDIDAS 51

6 RESULTADOS DA PESQUISA 52

7 CONCLUSÕES E TRABALHOS FUTUROS 56

REFERÊNCIAS 58

1 INTRODUÇÃO

A qualidade de Software é um tema que tem ganhado destaque nas empresas e universidades, sendo alvo de diversos estudos que visam melhorar a qualidade dos sistemas e dos processos de desenvolvimento. Segundo Bernardo (2011), dentre esses estudos, as pesquisas na área de teste de software têm crescido de maneira significativa nos últimos tempos, especialmente as pesquisas em automação de testes. A prática de testes automatizados também tem se difundido no mercado de software, tornando-se uma atividade importante no processo de testes das organizações (CAETANO, 2013)(COLLINS; LUCENA, 2012)(LIMA *et al*., 2012).

Em um cenário real de desenvolvimento, muitas vezes testadores não tem o tempo necessário para testar completamente o produto dentro dos prazos definidos para entrega, demandando um aumento de esforço por parte do testador ou a alocação de novos recursos (CERVANTES, 2009). Além disso, o analista de testes, usualmente, é o único responsável pela atividade de testes do projeto, tendo que dedicar parte do seu tempo na identificação de falhas em fluxos básicos, que poderiam ser identificadas durante o desenvolvimento, ao invés de focar em cenários mais elaborados (CHIAVEGATTO *et al*., 2013). Tendo em vista que esse esforço de testes em projetos de software pode ser responsável por até 50% do esforço total de desenvolvimento, automatizar o processo de testes é importante para reduzir os custos e melhorar a eficácia dos testes de software (BUDNIK; CHAN; KAPFHAMMER, 2010).

Na literatura, é possível encontrar diversos trabalhos que propõem a utilização da automação de testes em projetos de software. Em Collins e Lobão (2010), é mostrada uma experiência de automação de testes em um ambiente ágil de desenvolvimento, utilizando ferramentas *open source*. Lima *et al.* (2012) apresenta a implantação de um processo de automação em uma empresa pública de TI, com o apoio da ferramenta SilkTest. Em Oliveira, Gois e Farias (2007), é relatada uma experiência de automação de testes no Serviço Federal de Processamento de Dad7os – SERPRO, através do desenvolvimento do framework FunkTest para automação de testes funcionais.

O objetivo geral deste trabalho é apresentar um estudo de caso na implantação de testes funcionais automatizados em um sistema web desenvolvido no Laboratório de Sistemas e Bancos de Dados – LSBD, da Universidade Federal do Ceará - UFC. Os resultados específicos relacionados ao trabalho são: definir um processo de automação de testes para o projeto, executar as atividades definidas no processo avaliando sua aplicação em um projeto

real de desenvolvimento de software e verificar o impacto da automação de testes no projeto, a partir de um comparativo com o processo de testes manual.

O trabalho está organizado da seguinte forma: Inicialmente, no capíulo 2, é feita uma revisão bibliográfica sobre os principais conceitos que guiam este trabalho, mostrando os conceitos de teste de software, automação de testes de software e ferramentas de apoio à automação. No capítulo 3, são apresentados os procedimentos metodológicos utilizados para atingir os objetivos desse trabalho de pesquisa. O capítulo 4 apresenta o estudo de caso desenvolvido, mostrando o projeto-alvo da automação e os processos adotados, apresentando também como o processo de automação de testes foi inserido no projeto, e como as atividades foram executadas. O capítulo 5 mostra as principais dificuldades encontradas na realização desse trabalho e as principais lições aprendidas. O capítulo 6 apresenta uma avaliação do impacto da automação de testes no desempenho e qualidade dos testes executados. Por último, na seção 7, são feitas as considerações finais do trabalho.

2 REVISÃO BIBLIOGRÁFICA

Neste capítulo, serão apresentados os conceitos, processos e técnicas que servem de base para esse trabalho de pesquisa. Na primeira seção, são introduzidos os principais conceitos relacionados ao teste de software, definindo as principais abordagens. A seção seguinte define a automação de teste de software, enfatizando os seus principais benefícios e desafios, e mostrando as principais atividades envolvidas no processo de automação. A terceira seção apresenta as principais técnicas de automação de teste existentes na literatura, mostrando como elas podem ser aplicadas e suas vantagens e desvantagens em potencial. Por último, são apresentadas algumas ferramentas que auxiliaram na atividade de automação de testes de software realizada nesse trabalho de pesquisa.

2.1 Teste de software

A Engenharia de Software combina atividades de projeto e construção com atividades que verificam produtos intermediários e finais de forma que os defeitos possam ser identificados e removidos, garantindo a construção de um software de alta qualidade (PEZZÉ; YOUNG, 2008, p. 25). Sommerville (2011) define quatro atividades básicas para um processo de software: especificação, desenvolvimento, validação e evolução. As atividades de validação, também chamadas de verificação e validação (V&V) têm como objetivo mostrar que um software se adequa as suas especificações e satisfaz os requisitos do cliente.

"Verificação se refere ao conjunto de atividades que garantem que o software implementa corretamente uma função específica, enquanto validação se refere a um conjunto de atividades que garante que o software construído corresponde aos requisitos do cliente." (PRESSMAN, 2006, p. 289). Dentre as atividades de V&V, o teste de software é um importante recurso para a garantia da qualidade em conjunto com outras atividades como revisões técnicas formais, auditoria da qualidade, revisões de documentação, entre outras.

Existem na literatura várias definições para teste de software. Dentre elas podemos destacar:

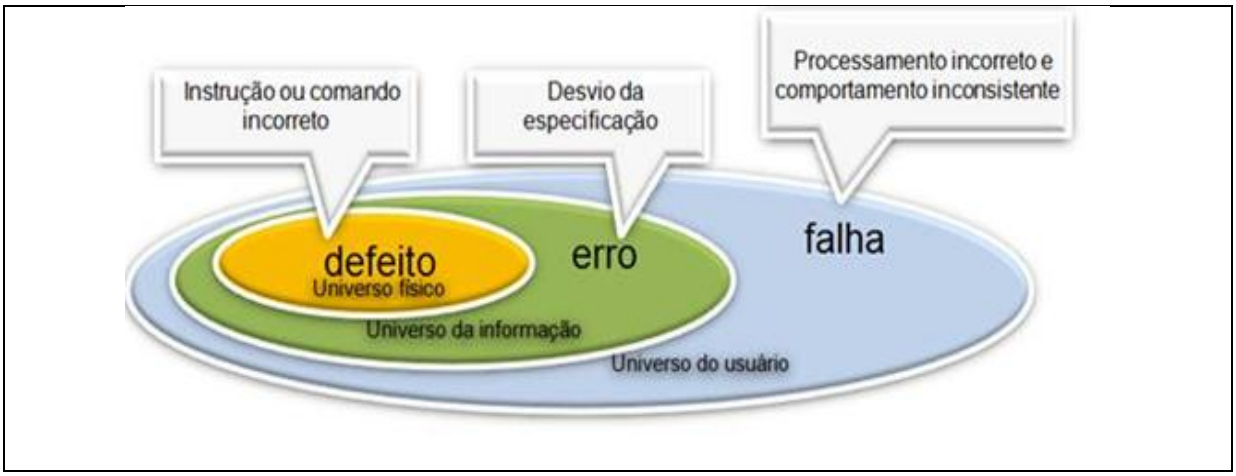
> [...] o processo de avaliar um sistema ou um componente de um sistema por meios manuais ou automáticos para verificar se ele satisfaz os requisitos especificados ou identificar diferenças entre resultados esperados e obtidos. (ANSI/IEEE Standart 729, 1983, p. 729).
>
> [...] é um processo, ou um grupo de processos, definidos para garantir que um código faz o que ele foi desenhado para fazer, e não faz nada que não foi especificado para fazer .(MYERS; COREY, 2004).

> [...] um conjunto de atividades que podem ser antecipadamente planejadas e conduzidas sistematicamente, com o objetivo de encontrar erros, defeitos e/ou falhas no sistema. (PRESSMAN, 2006, p. 289).

Para uma ampla compreensão da atividade de teste de software, é necessário conhecer alguns conceitos associados a essa atividade, dentre eles os conceitos de erro, defeito e falha. Esses conceitos são mostrados abaixo e ilustrados na figura 1, de acordo com a definição do *Institute of Electrical and Electronics Engineers* – (IEEE 610, 1990).

- Erro é uma ação inconsistente cometida por algum indivíduo ao tentar entender uma determinada informação, resolver um problema ou utilizar um método ou uma ferramenta.
- Defeito é uma manifestação concreta de um erro num artefato de software. Um erro pode causar vários defeitos.
- Falha é a diferença entre os resultados obtidos pelo sistema e os resultados esperados pelo usuário. Uma falha pode ser causada por vários erros e um erro pode nunca causar uma falha.

Figura 1: Erro x Defeito x Falha



Fonte: Dias (2008)

Segundo Koscianski e Soares (2007, p. 337) quando a atividade de testes é planejada de maneira sistemática e rigorosa, pode ser usada como uma forma de estimar a confiabilidade e qualidade do software desenvolvido.

Nas próximas subseções serão abordados os principais tipos de teste de software, as principais técnicas existentes na literatura, bem como aspectos referentes ao processo de testes de software.

2.1.1 *Níveis de testes de software*

Pressman (2006, p. 294) aponta que realizar testes apenas quando todo o sistema estiver construído é uma abordagem ineficaz. Segundo o autor, uma estratégia de testes de software deve seguir uma abordagem incremental, começando com o teste das unidades, seguindo com os testes de integração, culminando com os testes do sistema final. Seguindo a mesma abordagem, Softex (2011b) classifica os testes em quatro níveis: unidade, integração, sistema e aceitação.

2.1.1.1 Testes de unidade

Os testes de unidade verificam a menor unidade do projeto de software, com foco na lógica interna de processamento e estruturas de dados de um componente. Normalmente, ele é associado a funções para linguagens procedurais e métodos em linguagens orientadas a objetos (BERNARDO, 2011). Os testes desse nível são realizados durante a fase de implementação, pelos próprios desenvolvedores, e devem garantir que o código foi desenvolvido em conformidade com o que foi especificado ante de sua integração com as demais unidades (SOFTEX, 2011b).

2.1.1.2 Testes de integração

Os testes de integração verificam a interface entre os componentes e ocorre em paralelamente à atividade de integração do sistema. Segundo Softex (2011b), testes desse tipo "são feitos em sistemas completos ou subsistemas, compostos de componentes integrados, onde unidades de software e hardware são testadas juntas para avaliar a integração entre eles, permitindo exposição dos problemas nas interfaces das unidades."

2.1.1.3 Testes de sistema

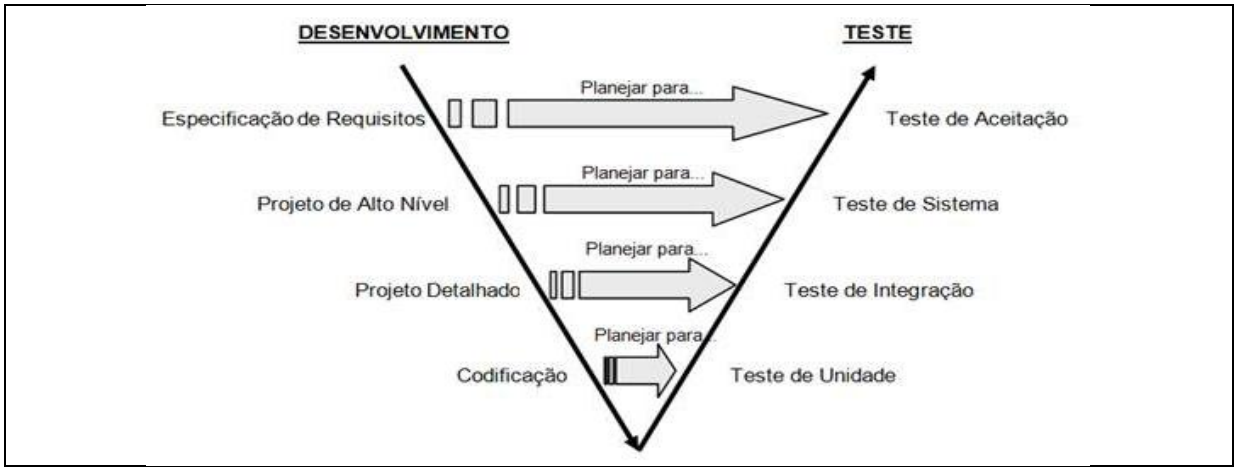
Segundo Softex (2011b), os testes de sistema têm como objetivo verificar o comportamento do sistema final, e devem ser executados em um ambiente que corresponda ao ambiente no qual o sistema será implantado. "O teste de sistema refere-se ao comportamento do sistema como um todo, com foco na análise da conformidade com os requisitos. Quando o sistema é desenvolvido iterativamente, o teste de sistema pode ser executado para cada subconjunto desenvolvido." (SOFTEX, 2011b).

2.1.1.4 Testes de aceitação

Segundo Softex (2011b), "o teste de aceitação é aquele realizado para assegurar ao usuário de que o sistema irá funcionar de acordo com as suas expectativas." Para a execução de desse tipo de testes é necessário definir critérios de aceitação a partir dos requisitos do software, estabelecendo como o teste será conduzido e a partir desses critérios, avaliar se o produto satisfaz aos requisitos (SOFTEX, 2011b).

A figura 2 ilustra a execução das atividades de teste paralelamente as atividades de desenvolvimento.

Figura 2: Modelo V relacionando as atividades de desenvolvimento e teste de software



Fonte: Craig e Jaskiel (2002).

Dias (2008) explica que o planejamento e projeto dos testes devem ocorrer de cima para baixo, sendo inicialmente planejado o teste de aceitação a partir do documento de requisitos. Após essa fase é planejado o teste de sistema a partir do projeto de alto nível do software. Em seguida ocorre o planejamento dos testes de integração a partir o projeto detalhado e por fim, o planejamento dos testes de unidade a partir da codificação. Já a execução das atividades de teste deve acontecer de baixo para cima, de acordo com a ordem: teste de unidade, teste de integração, teste de sistema, teste de aceitação.

2.1.2 *Tipos de teste de software*

Além de avaliar as funções, a interface e as características de tempo de resposta de um sistema, os testes de software devem também se concentrar em características e atributos como a integridade (resistência a falhas), capacidade de o sistema ser instalado e executado em diferentes plataformas e capacidade de lidar com várias solicitações ao mesmo tempo

(RUP, 2007). Para atingir esses objetivos, o RUP (2007) define vários tipos de testes, com objetivos específicos, classificando-os de acordo com as dimensões da qualidade definidas no modelo FURPS (*Functionality*, *Usability*, *Reliability*, *Performance*, *Supportability*), conforme mostra a tabela abaixo. Na próxima subseção, serão definidos cada um desses tipos de teste.

Quadro 1 – Classificação dos testes segundo o modelo FURPS

Dimensão da Qualidade	Tipo de teste
Funcionalidade	Teste de função Teste de segurança Teste de volume
Utilidade	Teste de usabilidade
Confiabilidade	Teste de integridade Teste de estrutura Teste de estresse Teste em tamanho real Teste de contenção
Desempenho	Teste de carga Teste de perfil de desempenho
Suportabilidade	Teste de configuração Teste de instalação

Fonte: RUP (2007)

2.1.2.1 Funcionalidade

Essa dimensão agrupa testes cujo objetivo é verificar se o sistema construído está de acordo com o que foi especificado RUP (2007), define os seguintes testes para essa dimensão:

a) **Teste de função:** verifica se os objetivos do teste estão sendo atendidos conforme o esperado, fornecendo os serviços, métodos ou casos de uso necessários. Pode ser executado nos diferentes níveis de teste, como unidade, integração e sistema.

b) **Teste de segurança:** concentra-se em garantir que os objetivos do teste e os dados ou sistemas associados estejam acessíveis somente para os atores aos quais se destinam.

c) **Teste de volume:** verifica a capacidade do objetivo do teste em lidar com diferentes quantidades de dados de entrada, saída e acesso a banco de dados.

2.1.2.2 Usabilidade

Segundo Eeeles (2005), essa dimensão tem interesse em demonstrar a facilidade de uso do sistema, através de características como estética e consistência na interface do usuário. É representada pelo teste de usabilidade:

- **Teste de usabilidade:** de acordo com RUP (2007), esse tipo de teste tem como objetivo verificar a facilidade de compreensão e manipulação do sistema pelo usuário. Concentram-se em aspectos como fatores humanos, estética, consistência na interface com o usuário, ajuda on-line e sensível ao contexto, assistentes e agentes, documentação do usuário e materiais de treinamento.

2.1.2.3 Confiabilidade

De acordo Eeeles (2005), essa dimensão da qualidade tem interesse nas características como disponibilidade, acurácia nos cálculos realizados pelo sistema, e habilidade do sistema se recuperar no caso de falhas RUP (2007), define os seguintes testes de confiabilidade:

- **Teste de integridade:** tem como objetivo avaliar a robustez do software e a compatibilidade técnica em relação à linguagem, sintaxe e uso dos recursos. Pode ser executado para unidades individuais do sistema e para unidades integradas.
- **Teste de estrutura:** avalia o design e formação do sistema. “Geralmente, esse teste é feito para sistemas web, para assegurar que todos os links estão conectados, o conteúdo apropriado é exibido e nenhum conteúdo é órfão.” (RUP, 2007).
- **Teste de estresse:** enfatiza como o sistema se comporta em condições extremas, como grandes cargas de trabalho, memória insuficiente, serviços e hardware indisponíveis ou recursos compartilhados limitados. Conforme RUP(2007) “normalmente, esses testes são executados para compreender melhor como e em quais áreas o sistema será dividido, para que os planos de contingência e a manutenção de atualização possam ser planejados e orçados com bastante antecedência.”

2.1.2.4 Desempenho

Sedgundo Eeeles (2005), essa dimensão da qualidade tem foco em questões como perfis de andamento, fluxo de execução, tempos de resposta, tempo de inicialização e limites operacionais. RUP (2007) apresenta os seguintes testes de desempenho

- **Teste em tamanho real:** compara o desempenho de um sistema com uma carga de trabalho e um sistema de referência conhecidos.
- **Teste de contenção:** Segundo RUP (2007), esse tipo de testes faz a validação da habilidade do objetivo do teste para conseguir manipular aceitavelmente as demandas dos agentes em um mesmo recurso (registros de dados, memória e assim por diante).”
- **Teste de carga:** segundo RUP (2007) o teste de carga é “utilizado para validar e avaliar a aceitação dos limites operacionais de um sistema em cargas de trabalho variadas enquanto o sistema em teste permanece constante.” As cargas emuladas normalmente são feitas dentro dos limites médios e máximos suportados pelo sistema (RUP, 2007).
- **Perfil de desempenho:** “um teste em que o perfil de cronometragem do objetivo do teste é monitorado, incluindo o fluxo de execução, o acesso a dados, as chamadas de função e do sistema para identificar e abordar gargalos de desempenho e processos ineficientes.” (RUP, 2007).

2.1.2.5 Suportabilidade

De acordo com Eeeles (2005), essa dimensão concentra-se nas características como testabilidade, adaptabilidade, gerenciabilidade, compatibilidade, configurabilidade, instalabilidade, escalabilidade e locabilidade. RUP (2007) define os seguintes testes de suportabilidade:

- **Teste de configuração:** Visa garantir que os objetivos do teste serão atendidos em diferentes configurações de hardware e/ou software, podendo também, ser implementados como um teste de desempenho do sistema.
- **Teste de instalação:** Tem como meta garantir que o objetivo do teste será instalado conforme o previsto em diferentes configurações de hardware e/ou

> software e sob diferentes condições, como por exemplo, no caso de espaço insuficiente em disco ou interrupção de energia (RUP, 2007).

2.2 Automação de testes

Apesar de ser uma atividade complexa, o teste de software nem sempre é realizado de forma sistemática devido a restrições impostas pelos projetos, como tempo e recursos limitados, qualificação dos envolvidos e como consequência da complexidade e da rápida evolução dos sistemas, sendo a automação de testes uma importante medida para melhorar a eficiência dessa atividade (FANTINATO; CUNHA; DIAS, 2004).

"Automação de testes é uso de software para controlar a execução do teste de software, a comparação dos resultados esperados com os resultados reais, a configuração das pré-condições de teste e outras funções de controle e relatório de teste."(KOLAWA, A.; HUIZINGA, D, 2007 *apud* HOODA, 2012). Essa prática vem sendo difundida tanto nos processos ágeis, como o Extreme Programming (XP), quanto nas organizações que usam metodologias mais tradicionais como o Rational Unified Process (LIMA *et al*., 2012).

Segundo Fewster e Graham (1999), a automação de testes de software pode reduzir de maneira significativa o esforço necessário para a execução dos testes. Molinari (2010, p. 37) afirma que o teste automatizado aumenta a produtividade e atinge em tempo menor os objetivos do teste. Apesar desse potencial que os testes automatizados apresentam, ainda existe resistência de algumas empresas para a automação de testes, sobretudo, devido ao esforço inicial do processo e por demandar uma mudança de cultura organizacional (LIMA *et al*., 2012).

Nas próximas seções serão discutidos os principais benefícios e dificuldades da implantação de testes automatizados, as principais técnicas existentes na literatura e aspectos sobre a implantação de testes automatizados em uma organização.

2.2.1 *Benefícios da automação de testes*

De acordo com Fewster e Graham (1999), os principais benefícios da automação de testes de software são:

a) Facilitam a execução de testes de regressão, tendo em vista que os casos de testes automatizados para uma versão do sistema podem ser reproduzidos nas versões seguintes com um esforço manual mínimo;

b) Permite executar um maior número de testes em menor tempo, aumentando a frequência dos testes e trazendo mais confiança ao sistema;

c) Realizar testes difíceis ou impossíveis de serem executados manualmente, como verificação de cálculos complexos ou testes de carga com um grande número de acessos;

d) Melhorar a utilização de recursos, permitindo que os testadores reduzam o esforço em tarefas repetitivas, dando maior atenção ao planejamento dos testes e a execução de testes cuja automação é inviável;

e) Permite aumentar a consistência dos resultados dos testes entre diferentes plataformas, já que as entradas e condições esperadas são as mesmas;

f) Reduzir o *time to market*, tendo em vista a redução do tempo do projeto dedicado aos testes;

g) Aumentar a confiança na qualidade do sistema. Um grande conjunto de testes automatizados bem planejados executados com êxito pode dar a equipe um sentimento de confiança na qualidade do produto a ser liberado.

A partir desses benefícios é possível perceber o potencial da automação de testes na melhoria das atividades de teste de software. Alguns desses benefícios foram evidenciados através do estudo de caso proposto nesse trabalho de pesquisa, principalmente, no que se refere à redução do esforço de testes. Esses resultados são mostrados na seção 6 deste trabalho.

2.2.2 *Desafios da automação de testes*

Segundo Fewster e Graham (1999, p. 9), os principais desafios na implantação da automação de testes de software em uma organização são:

a) **Expectativas irreais**: a automação de testes não deve ser encarada como a solução de todos os problemas, devendo ser encarada de maneira realista;

b) **Prática de testes pobre**: se o processo de testes adotado na organização é ineficaz, com testes mal planejados e documentação inconsistente, é necessário melhorar esse processo antes de implantar a automação;

c) **Manutenção dos testes**: quando o software é modificado muitas vezes, o esforço de manutenção dos testes automatizados pode ser maior do que o esforço de reexecutá-los manualmente;

d) **Problemas técnicos**: as ferramentas de automação de testes também estão sujeitas a problemas comprometendo a estratégia de automação. Além disso, é necessário que o responsável pela automação tenha conhecimentos técnicos detalhados da ferramenta, para que seu uso possa ser efetivo;

e) **Questões organizacionais**: a automação de testes precisa ser implementada na cultura da organização, com alocação de tempo para a escolha de ferramentas e formação, visando entender e selecionar as práticas que melhor se adequam a organização;

Diante dessas dificuldades, automatizar testes de software torna-se uma atividade complexa. Dessa forma, a automação de testes precisa ser bem analisada sobre a sua necessidade e como ela será mais bem adaptada às necessidades de cada processo de desenvolvimento de software. Na seção 5, são mostrados quais desses desafios foram enfrentados na automação de testes realizada nesse trabalho, e como eles foram contornados pela estratégia de automação.

2.2.3 *Processo de automação de testes*

Segundo Caetano (2013), apesar de ser uma área em expansão, a automação de testes ainda é uma área muito imatura, sendo o sucesso de boa parte dos projetos de testes automatizados decorrentes de processos empíricos de tentativa e erro. Entretanto, é possível encontrar na literatura modelos de processos de automação de teste, que propõem atividades necessárias à implantação de testes automatizados pelas organizações. Para a realização desse trabalho, foi escolhido o Modelo de Melhoria do Processo de Teste Brasileiro (MPT.Br), por ser um modelo baseado em normas reconhecidas internacionalmente, como a ISO/IEC 29119 e IEEE 829:2008 e devido aos resultados positivos de sua implantação em várias empresas brasileiras, conforme aponta Rios (2013).

“O Modelo de Melhoria do Processo de Teste Brasileiro (MPT.Br) trata a melhoria do processo de teste através de melhores práticas relativas às atividades desenvolvidas ao longo do ciclo de vida de teste do produto.” (SOFTEX, 2011a). Uma das áreas do modelo é a AET - Automação da Execução do Teste, que tem como objetivo estabelecer e manter uma estratégia para a automação da execução de teste a partir da execução de um conjunto de práticas que serão detalhadas nas subseções seguintes.

2.2.3.1 Definir objetivos do regime de automação

Segundo Softex (2011a), essa prática tem como meta “estabelecer e manter objetivos para a automação da execução de teste”. Para determinar se esses objetivos estão sendo alcançados é necessário avaliar periodicamente o regime de automação verificando os procedimentos, ferramentas e recursos envolvidos na automação de teste. A definição dos objetivos é necessária a uma estratégia de automação, pois permite avaliar a efetividade da automação de testes em uma organização (SOFTEX, 2011a).

2.2.3.2 Definir critérios para seleção de casos de teste para automação

Essa atividade tem como objetivo "estabelecer e manter critérios para automação de casos de teste, conforme ressalta Softex (2011a).

> O esforço empreendido na criação e manutenção de um caso de teste automatizado, deve ser avaliado com o intuito de evitar que casos de teste sejam automatizados de forma ad hoc. Desta forma, devido a esta relação de custo X benefício, nem todo caso de teste deve ter sua execução automatizada (SOFTEX, 2011a)..

Dessa forma, a fim de obter melhores resultados com uso da automação, essa prática recomenda que a automação seja realizada apenas para casos de teste que atendem aos critérios de seleção definidos.

2.2.3.3 Definir um framework para automação de teste

Tem como objetivo estabelecer e manter um framework para apoio às atividades de automação da execução de teste, provendo uma infraestrutura responsável por “dar suporte às tarefas da automação da execução do teste, compreendendo entre outros, ferramentas, rotinas, dados e utilitários.” (SOFTEX, 2011a)

2.2.3.4 Gerenciar incidentes de teste automatizado

Segundo Softex (2011a), essa prática tem como objetivo "identificar, analisar e gerenciar incidentes da execução de teste automatizada". Essa atividade é importante, pois durante a execução de testes automatizados vários incidentes podem ser encontrados, principalmente no início de sua implantação, devido à instabilidade da infraestrutura de automação, que faz com que muitos defeitos sejam originados de código de automação problemático, erro na definição dos dados de teste, preparação imprópria do ambiente de teste e até mesmo defeitos no próprio framework de automação.

2.2.3.5 Verificar aderência aos objetivos de automação

O objetivo desta prática é analisar periodicamente o regime de automação para verificar se os objetivos da automação estão sendo atendidos, e em caso isso não aconteça dimensionar ações corretivas (SOFTEX, 2011a).

2.2.3.6 Analisar retorno sobre investimento na automação

Tem como objetivo realizar uma análise periódica do retorno sobre o investimento com a automação de testes, levantando informações relativas aos ganhos da automação como esforço, tempo e custo da execução dos testes e também das despesas associadas como, por exemplo, a aquisição de hardware e software, consultorias e treinamentos e desenvolvimento de casos de teste automatizados (SOFTEX, 2011a).

As atividades propostas no MPT-Br serviram de arcabouço para a definição da estratégia de automação proposta nesse trabalho, tendo em vista que a organização alvo desse estudo de caso não possuía nenhuma atividade de automação de testes definida que servisse de base para esta pesquisa. As atividades do modelo foram adaptadas para adequar-se ao contexto da organização alvo conforme é mostrado na subseção 4.4.

A atividade "Definir um framework para automação de teste" foi dividida em atividades de desenvolvimento, execução e visualização de resultados dos testes. A atividade Verificar aderência aos objetivos da automação é apresentada como parte dos resultados desse trabalho, na seção 6. Por limitações de escopo a atividade "Analisar retorno sobre investimento na automação" foi descartada desse trabalho, por demandar uma análise mais detalhada dos recursos envolvidos no processo de automação.

2.2.4 *Técnicas de automação de testes*

De acordo com Fantinato, Cunha e Dias (2004) e Socolowski, Alarcon e Antonio (2012) as principais técnicas de automação de automação de teste, presentes na literatura são: *record* & *playback*, *scripts*, *data-driven* e *keyword-drivers*. Essas técnicas são explicadas a seguir.

2.2.4.1 Record & playback

Segundo Fantinato, Cunha e Dias (2004), essa técnica baseia-se na gravação das ações do usuário durante a interação com a interface gráfica do sistema, convertendo essas ações em scripts de teste que podem ser reproduzidos várias vezes, exatamente como na execução original. Para cada caso de teste é gravado um script de teste completo que inclui os dados de teste (dados de entrada e resultados esperados), o procedimento de teste (passo a passo que representa a lógica de execução) e as ações de teste sobre a aplicação. É considerada uma técnica simples e prática, mas apresenta algumas desvantagens para uma grande quantidade de testes automatizados, como alto custo e dificuldade de manutenção, baixa taxa de reutilização, curto tempo de vida e alta sensibilidade a mudanças no software a ser testado e no ambiente de teste.

2.2.4.2 Programação de scripts

É uma extensão da técnica record & playback que utiliza recursos de programação para alterar os scripts gravados, permitindo alcançar uma maior quantidade de verificações de resultados esperados. Essa técnica permite uma maior taxa de reutilização, maior tempo de vida, melhor manutenção e maior robustez dos scripts de teste. Entretanto, segundo Fantinato (2004) “a aplicação pura dessa técnica produz uma grande quantidade de scripts de teste, visto que para cada caso de teste deve ser programado um script de teste, o qual também inclui os dados de teste e o procedimento de teste”.

2.2.4.3 Data driven

Segundo Socolowski, Alarcon e Antonio (2012), essa técnica consiste em extrair, dos scripts de teste, os dados de entrada e armazená-los em arquivos separado, fazendo com que os scripts contenham somente a lógica de execução dos testes. Os dados de entrada são obtidos de um arquivo separado de acordo com cada caso de teste desenvolvido. Essa

abordagem tem como principal vantagem o baixo esforço de manutenção ao adicionar, remover ou modificar um caso de teste.

2.2.4.4 Keyword-driven

Segundo Socolowski, Alarcon e Antonio (2012) a técnica keyword-driven (técnica orientada a palavras-chave) "consiste em extrair a lógica de execução dos scripts de teste, que passam a conter apenas as ações específicas de teste sobre a aplicação, as quais são identificadas por palavras-chave." As ações podem ser comparadas a funções de um programa que são ativadas pelas palavras chave, durante a execução do caso de teste. Os passos executados pelo script são armazenados em um arquivo separado, na forma de um conjunto de palavras-chave e parâmetros de teste no próprio código, obtendo-os diretamente dos arquivos de procedimento de teste. A principal vantagem da técnica keyword-driven é que se pode facilmente adicionar, modificar, ou remover passos de execução com um pequeno esforço de manutenção (SOCOLOWSKI; ALARCON; ANTONIO, 2012).

A implementação inicial dos testes automatizados realizada nesse estudo de caso utilizou a técnica de *record & playback*, através da ferramenta Selenium IDE, permitindo um mapeamento automático dos elementos da página e das ações do usuário. Apesar da facilidade, os scripts gerados com essa técnica não foram usados em sua forma inicial na execução dos testes, por não permitirem a simulação de novas ações ou inserção de novos dados, tendo em vista que representam uma interação específica do usuário. Dessa forma, a partir dos scripts gerados foi aplicada a técnica de programação de scripts, permitindo agrupar os passos dos scripts em métodos que permitem sua reutilização, mudando apenas os dados de entrada. Além disso, foi possível incorporar outros recursos como a verificação automática de resultados com o Junit e integração com Testlink, ferramentas que serão explicadas na próxima seção. A aplicação das técnicas de *data driven* e *keyword driven* foram descartadas devido ao casos de teste desenvolvidos não possuírem uma grande quantidade de dados de entrada que justificasse a adoção dessas técnicas. Entretanto, elas podem ser utilizadas futuramente para testes mais complexos e com uma maior carga de dados, tendo em vista seu alto potencial de reutilização dos scripts desenvolvidos.

2.3 Ferramentas de apoio à automação de testes

Desde o surgimento do teste de software, vem sendo discutida a utilização de ferramentas que facilitem o trabalho dos profissionais de teste, tarefas como o planejamento

de casos de testes, execução de testes, abertura de defeitos entre outros (PATUCI, 2013). Com o objetivo de apoiar a estratégia de automação de testes apresentada nesse trabalho, foram utilizadas algumas ferramentas que serão apresentadas a seguir.

2.3.1 *Selenium*

Selenium é um conjunto de ferramentas que permite automatizar testes de diferentes formas, através de um conjunto de funções que permitem a localização de elementos de interface do usuário e a comparação entre os resultados previstos para o teste e o comportamento real da aplicação (SELENIUMHQ, 2013).

Selenium IDE (*Integrated Development Environment*) é uma ferramenta para a construção de scripts de teste, por meio de um plugin para o navegador Firefox, que fornece uma interface amigável para o desenvolvimento dos testes automatizados. O Selenium IDE usa uma abordagem *record-and-playback*, capturando as ações executadas pelo testador em um script que permite reproduzir o caso de teste posteriormente.

A criação dos scripts com o Selenium IDE pode ser feita seguindo os seguintes passos:

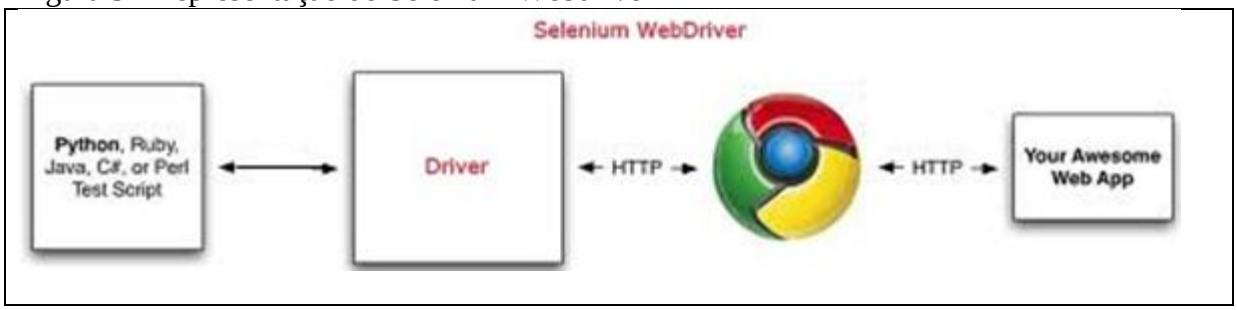
- **Capturar os passos do caso de teste**: o testador interage diretamente com o sistema seguindo um caso de teste específico. Para cada ação feita pelo testador, a ferramenta cria um passo no script de testes.
- **Definir critérios de validação**: para validar as condições de saída do caso de teste o Selenium IDE fornece vários mecanismos que permitem ao testador verificar o resultado da execução. Baseando-se nos critérios do caso de teste, o testador seleciona determinado elemento da página e faz asserções para verificar se os valores presentes naqueles componentes atendem aos resultados esperados para o caso de teste;
- **Executar script**: Executar o script de teste para verificar se o script gerado capturou todos os passos do caso de teste e as verificações correspondem aos resultados esperados. Essa execução reproduz todas as ações e entradas informadas pelo testador durante a gravação do script. Ao final da execução, o critério de validação é avaliado informando ao usuário o sucesso ou a falha do caso de teste.

Após a criação do script é possível exportá-lo para uma das linguagens de programação suportada, permitindo ao testador aprimorar o caso de teste de uma forma programática. Essa é uma importante característica do Selenium IDE, já que segundo (SELENIUMHQ, 2013) ela é uma ferramenta de prototipagem rápida que não fornece recursos mais avançados como iterações e instruções condicionais, recomendando, para uma estratégia de automação mais robusta o Selenium Web Driver.

Segundo (SELENIUMHQ, 2013), o Selenium Webdriver é uma api (*application programming interface)* orientada a objetos, disponível para as linguagens de programação Java, C#, Ruby, Phyton e Javascript. Além da flexibilidade proporcionada pela programação, a ferramenta suporta vários navegadores, dentre eles, o Google Chrome, Internet Explorer, Firefox, Opera, permitindo também testes em plataformas móveis como Android e IOS.

Conforme mostra a figura3, os scripts são desenvolvidos nas linguagens de programação suportadas e as chamadas são feitas pelo Webdriver diretamente ao navegador usando seus recursos nativos para acessar o sistema a ser testado.

Figura 3 - Representação do Selenium Webdriver



Fonte: Adaptado de http://rritw.com/a/shujuku/Oracle/20110714/96681.html

O Selenium Grid é uma ferramenta que permite executar testes em diferentes máquinas e em diferentes navegadores, paralelamente, permitindo acelerar a execução dos testes, produzindo, dessa forma, resultados mais rápidos. Essa paralelização é importante, principalmente, quando existe uma grande quantidade de casos de testes, que podem levar horas para serem executados.

2.3.2 *Junit*

O JUnit é um framework *open source* que facilita o desenvolvimento e a execução de testes unitários em Java (MEDEIROS, 2013). Segundo Chiavegatto *et al*. (2013), o Junit permite que os testes possam ser executados sequencialmente ou de forma modularizada,

dessa forma, os sistemas podem ser testados em partes ou de uma única vez. De maneira geral funcionamento do Junit baseia na verificação das saídas dos métodos de uma classe, analisando se os mesmos apresentaram os resultados esperados, exibindo de forma visual o resultado da execução dos testes.

2.3.3 *Testlink*

O TestLink é uma aplicação Open Source cujo principal objetivo é gerenciar as atividades de testes de um projeto. O TestLink permite criar Test Cases e organizá-los em Test Suites, associar casos de teste a testadores, acompanhar os resultados da execução dos testes, permitindo também, gerar diversas métricas para o acompanhamento da execução dos testes. (CAETANO, 2007). Caetano (2007), destaca os seguintes recursos presentes no TestLink:

- Pode ser executado em qualquer plataforma que suportar PHP/Apache/Mysql (Windows, Linux, Mac, Solaris, AS400/i5, etc);
- Traduzido em várias línguas diferentes (incluindo "Brazil Portuguese");
- Controle de acesso e níveis de permissões por papéis (Líder, Testador, etc);
- Os casos de testes são organizados hierarquicamente em suítes; ? Os casos de testes podem ser classificados por palavras-chave "keyword" para facilitar a pesquisa e organização;
- Criação ilimitada de projetos e casos de testes;
- Os ciclos de testes podem ser priorizados e atribuídos aos testadores;
- Gerador interno de relatórios e gráficos (possibilidade para exportar os dados nos formatos CSV, Excel e Word);
- Integração com ferramentas de gestão de defeitos (Bugzilla, Mantis, Jira);

Cada uma dessas ferramentas teve uma contribuição específica para a estratégia de automação. A escolha do Selenium justificou-se por ser uma ferramenta popular para automação de testes em ambientes web e por permitir o desenvolvimento dos testes em várias linguagens de programação. O Junit foi utilizado em conjunto com o Selenium para facilitar a verificação dos resultados dos testes. O uso do Testlink completa a estratégia de automação, facilitando a atividade de documentação de resultados dos testes e geração de relatórios de feedback sobre a qualidade do sistema.

3 PROCEDIMENTOS METODOLÓGICOS

Tendo em vista que o teste de software é uma das principais atividades da Engenharia de Software e que demanda um grande esforço em projetos de desenvolvimento, esse trabalho de pesquisa parte do seguinte problema: como implantar uma solução de testes automatizados em um projeto de software, de forma a reduzir o esforço de testes e manter a qualidade dos produtos desenvolvidos? Para responder a essa questão, o presente trabalho teve como meta principal realizar um estudo de caso na implantação de testes funcionais no Laboratório de Sistemas e Bancos de Dados da Universidade Federal do Ceará - LSBD. Para a realização do estudo de caso o autor desta pesquisa definiu um processo de automação de testes baseado no MPT-Br e aplicou esse processo em um projeto desenvolvido no LSBD. A avaliação da pesquisa foi feita através da avaliação do esforço empreendido na abordagem de testes manuais e o esforço empreendido no processo de testes automatizados, envolvendo também uma avaliação qualitativa da estratégia de automação pelo responsável pelas atividades de teste no projeto.

Estão fora do escopo deste trabalho o desenvolvimento de novas ferramentas, a avaliação da estratégia de automação em termos de retorno sobre o investimento (ROI), bem como a seleção de casos de testes através de métodos matemáticos devido a complexidade que essas atividades trariam que não seria aplicável a um trabalho de graduação.

Todas as atividades do processo de automação foram executadas pelo autor desta pesquisa, no papel de analista de testes responsável pelos testes automatizado A seguir são apresentados os passos executados para a realização do trabalho:

a) Analisar o processo de testes manuais adotados na organização.

   Essa atividade envolveu um estudo das atividades de teste adotadas no projeto-alvo da estratégia de automação, visando verificar a necessidade para implantação de testes automatizados.

b) Definir um processo de automação de testes para o projeto

   A partir do processo de testes manuais existente e com base no MPT-Br, foi definido um processo de automação para o projeto, estabelecendo as principais atividades para o desenvolvimento e execução dos testes automatizados.

c) Selecionar casos de teste candidatos automação

Como parte do processo de automação definido, foi selecionado um conjunto de casos de teste para automação, com base em critérios definidos na literatura.

d) Implementar testes automatizados

A partir da documentação de casos de teste existente no projeto, foram desenvolvidos os casos de teste automatizados. Primeiro foi feita a gravação dos scripts de teste a partir do Selenium IDE e, em seguida, desenvolvido um projeto em linguagem Java, com o apoio dos frameworks Junit e Selenium Web Driver e da ferramenta de gestão de testes Testlink.

e) Executar testes automatizados

Nessa etapa, foi realizada a execução dos testes automatizados em paralelo à execução manual, registrando os resultados de maneira automática na ferramenta Testlink.

f) Analisar resultados da estratégia de automação

Após a execução dos testes foi feito um comparativo entre as abordagens de testes manuais e de testes automatizados, onde foi possível identificar os principais benefícios da automação em relação à execução manual e as principais deficiências dessa abordagem.

Na próxima seção, será mostrado, de maneira mais detalhada, como essas atividades foram conduzidas no contexto do estudo de caso realizado.

4 ESTUDO DE CASO

A seguir, será apresentado o estudo de caso mostrando como a estratégia de automação foi definida e como suas atividades foram desenvolvidas no projeto PTDH.

4.1 Contexto do Estudo de Caso

O Laboratório de Sistemas e Bancos de Dados (LSBD) é um laboratório de pesquisa e desenvolvimento vinculado ao Departamento de Computação da Universidade Federal do Ceará (UFC), que atua na formação e capacitação de profissionais. Suas principais competências são o desenvolvimento de aplicativos para dispositivos móveis, aplicações de internet rica (RIA), soluções web, tecnologia RFID, aplicações para ciclo de vida de produtos e diagnósticos de hardware.

O LSBD não adota nenhum modelo de processos de software formal, mas baseia suas atividades nos princípios das metodologias ágeis como *Scrum*, *Extreme Programming* (XP) e *Lean*, incorporando também atividades do Modelo de Melhoria do Processo de Software Brasileiro (MPS.Br) e boas práticas do *Project Management Body of Knowledge* (PMBOK).

O projeto que serviu de base para esse estudo de caso teve como objetivo o desenvolvimento de um sistema de controle do processo de recebimento, avaliação e conserto ou devolução de componentes de hardware. Por motivos de confidencialidade o projeto será chamado nesse trabalho de Projeto de Troca e Devolução de Hardware - PTDH. O projeto PTDH foi desenvolvido utilizando a plataforma .NET em conjunto com framework *front-end Bootstrap*. As principais funcionalidades do sistema são:

- **Requisitar peças junto ao fabricante:** os clientes podem requisitar a troca ou devolução de uma peça junto ao fabricante. Essa requisição baseia-se no preenchimento de um formulário ou no anexo de uma planilha com os dados do pedido.
- **Visualizar status das requisições:** os clientes podem visualizar o andamento da requisição junto ao fornecedor, verificando se ela foi aprovada, reprovada ou se ainda se encontra em fase de avaliação.
- **Avaliar a requisição do cliente:** o fabricante pode aprovar ou rejeitar uma requisição do cliente.

- **Cadastrar clientes:** o sistema mantém o cadastro dos clientes com os respectivos produtos comprados.

O projeto era composto por um time de 9 pessoas, sendo 6 desenvolvedores, 1 analista de requisitos 1 analista de testes e 1 analista responsável pela execução de todo o processo de testes automatizados, desde a seleção dos casos de teste candidatos ao desenvolvimento e execução dos testes automatizados, papel desempenhado pelo autor deste trabalho. A execução do projeto era dividida em *sprints* de desenvolvimento com duração média de 3 semanas com geração de *releases* para teste duas vezes por semana.

A documentação adotada no projeto incluía especificação de requisitos, casos de uso e casos de teste. A adoção desse nível de documentação justifica-se pela alta complexidade de negócio do sistema desenvolvido que demandava uma definição clara dos objetivos do cliente e do comportamento do sistema. O registro da execução dos casos de teste era feito através do preenchimento de uma planilha com os resultados da execução dos testes para cada release.

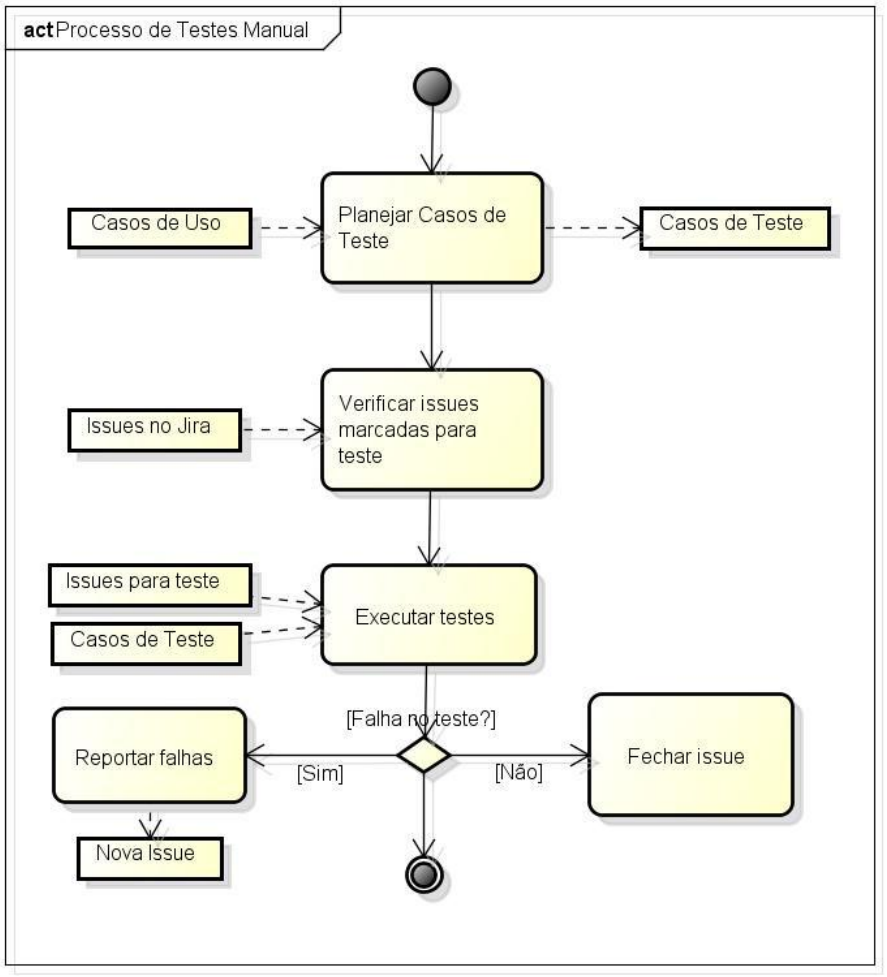
O gerenciamento das atividades do projeto foi feito com apoio da ferramenta de *issue tracking* Jira. No Jira, uma *issue* representa uma atividade que deve ser desenvolvida no projeto por determinado membro do time, e cuja execução é verificada através da atribuição de status (a fazer, em progresso, resolvida, fechada) que representa o progresso dessa atividade ao longo da *sprint*.

Na próxima seção, será apresentado o processo de testes manuais adotado no projeto antes da implantação dos testes automatizados, mostrando as atividades executadas e a dependência entre elas, bem como os produtos de trabalho gerados.

4.2 Processo de testes manuais adotado no projeto

O Laboratório de Sistemas e Bancos de Dados não adota nenhum processo formal de testes em seus projetos, mas possui atividades definidas com base em modelos como o MPS-Br, com foco nas necessidades de cada projeto. A figura 4 apresenta o processo de testes adotados no projeto PTDH, definindo suas principais atividades.

Figura 4: Processo de testes manuais adotado no Projeto PTDH



Fonte: Elaborado pelo autor

a) **Planejar casos de teste:** os casos de uso são analisados pelo analista de testes a fim de levantar todos os cenários de execução possíveis em um caso de uso e verificar o comportamento esperado do sistema. Todos esses cenários são definidos, resultando nos casos de teste do sistema. Cada caso de teste deve conter todas as pré-condições necessárias a sua execução, um descrição detalhada dos passos a serem executados, e critérios de verificação.

b) **Verificar *issues* marcadas para teste:** o analista de testes verifica no Jira as *issues* que já foram desenvolvidas e foram passadas para teste. Essas issues são adicionadas a uma planilha de execução de testes que permite registrar e acompanhar todos os testes realizados em uma versão.

c) **Executar testes:** as *issues* marcadas para teste são testadas seguindo os passos e os resultados esperados definidos nos casos de teste;

d) **Reportar falhas:** caso seja encontrada alguma falha para a *issue* testada, o analista de testes deve passar para o desenvolvedor responsável uma descrição detalhada da falha. Esse reporte é feito através da criação de uma nova *issue* que contém todas as informações necessárias para que o desenvolvedor reproduza a falha encontrada, como o ambiente de execução, pré-condições, passo a passo, além da severidade da falha encontrada, que é usada para priorização das falhas a serem corrigidas.

e) **Fechar *issue*:** uma vez executados todos os casos de teste relativos a uma *issue* e caso nenhuma falha seja encontrada a *issue* é fechada no Jira. Isso significa que a funcionalidade alvo da issue foi aprovada pelo analista de testes, possuindo qualidade suficiente para compor o produto final.

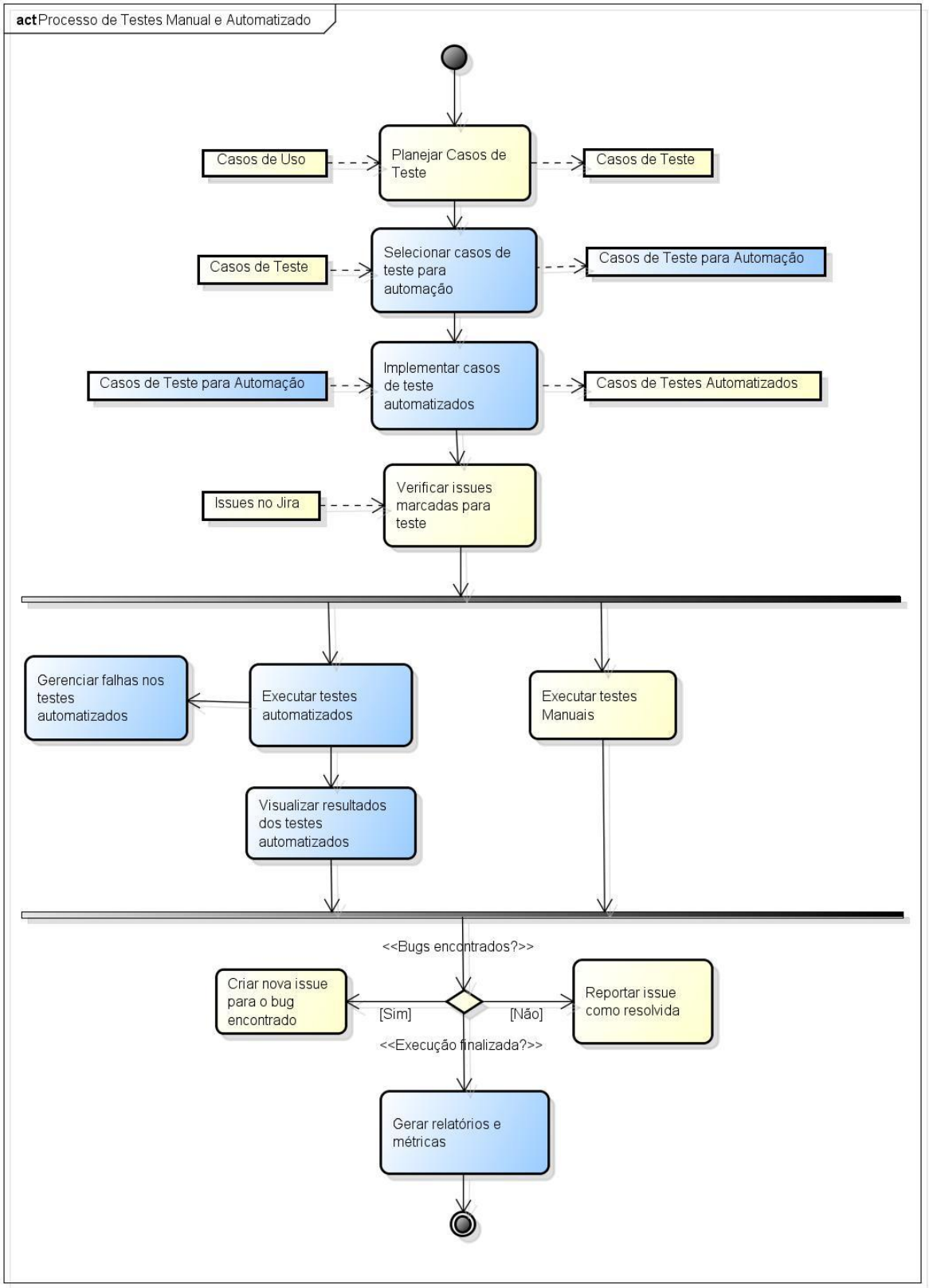
A estratégia de testes adotada no projeto baseava-se na execução de testes funcionais, e não possuía nenhuma atividade de testes automatizados. Todas as atividades de testes do projeto eram concentradas somente no analista de testes, que se dedicava as atividades de planejamento e execução dos testes, o que demandava um grande esforço e acabava comprometendo a cobertura dos testes. Além disso, não eram gerado nenhum relatório que desse *feedback* da qualidade geral do sistema, como por exemplo, relatórios de falhas encontradas, que permitem mapear a qualidade do sistema ao longo das versões. Na próxima seção será apresentado o processo de testes proposto nesse trabalho, mostrando como o uso de testes automatizados pode contribuir para a resolução desses problemas.

4.3 Processo de automação de testes

O processo de automação de testes para o Sistema de Troca e Devolução de Hardware foi organizado de forma a não impactar no processo já adotado no projeto, tendo em vista que se tratava de um projeto real e mudanças no seu processo poderiam representar riscos ao projeto. Para que isso fosse possível, as atividades de automação foram conduzidas paralelamente à execução manual dos testes procurando adotar as atividades base do projeto, já existentes na organização. Todas as atividades de automação de testes foram executadas pelo testador responsável pela automação de testes do projeto, papel desempenhado pelo autor

deste trabalho. A figura 5 apresenta a introdução dessas atividades (em azul) em conjunto com as atividades do processo manual (em amarelo) já existentes.

Figura 5: Processo de testes manuais em conjunto com as atividades do processo de testes automatizados



Fonte: Elaborado pelo autor

Essas atividades foram definidas com base na área de processo AET - Automação da Execução do Teste, definida pelo Modelo de Melhoria do Processo de Teste Brasileiro (MPT.Br), apresentada na seção 2.2.3 deste trabalho. Nas próximas subseções serão descritas as atividades de automação executadas e como elas foram conduzidas no projeto

4.3.1 *Definição dos objetivos da automação*

Inicialmente, foi conduzida uma análise do projeto a fim de identificar as principais necessidades para a implantação de testes automatizados. Essa análise foi feita a partir de discussões feitas com a analista de testes responsável pelo projeto e do levantamento de pontos negativos durante as reuniões de retrospectiva das *sprints*. Foi possível perceber que uma das principais dificuldades relacionadas à execução dos testes no projeto é a grande quantidade de casos de teste a serem verificados em vários ambientes, em curtos prazos para uma *release* do sistema. Esse cenário tem impactado diretamente na cobertura de testes por release, fazendo com que o analista de testes tivesse que priorizar os casos de testes que seriam executados e remover alguns ambientes do escopo de testes, comprometendo dessa forma a cobertura dos testes e a qualidade do sistema como um todo. De maneira geral, para releases intermediárias (que não eram entregues ao cliente) eram priorizados os casos de teste relativos a novas funcionalidades e o teste de falhas encontradas anteriormente, deixando-se de executar os testes de regressão. Apesar de serem definidos os navegadores Chrome, Firefox, Internet Explorer e Safari para a realização dos testes, eles eram geralmente executados no Internet Explorer em detrimento dos demais navegadores, por esse ambiente apresentar mais problemas nas execuções anteriores.

Tendo como base esses problemas e tendo em vista as potencialidades da automação foram definidos os seguintes objetivos para a estratégia de automação de testes no projeto PTDH: reduzir o esforço na execução dos testes funcionais agilizando a execução dos testes por release.

4.3.2 *Seleção de casos de teste para automação*

De acordo com CAETANO (2013), a automação de testes é pouco eficaz quando os testes são complexos e exigem interações intersistemas ou validações subjetivas, sendo necessário definir os casos de teste candidatos de acordo com o contexto da organização.

A partir dos objetivos da automação, foram definidos alguns critérios para selecionar o conjunto de casos de testes candidatos à automação no projeto PTDH, baseando-se nos critérios sugeridos em Softex (2011) e Nogueira (2009). Esses dois trabalhos apenas listam alguns critérios considerados importantes para a seleção dos casos de teste, mas não apresentam um método de avaliação. Dessa forma, como contribuição dessa pesquisa, foram definidas escalas de avaliação para cada critério selecionado. O resultado dessa seleção é mostrado no final dessa seção.

A seguir, estão listados os critérios utilizados para a seleção dos testes a serem automatizados por esse trabalho.

4.3.2.1 Importância das funcionalidades testadas para o cliente

Os testes são selecionados com o objetivo de validar funcionalidades que estão no centro dos interesses do cliente.

Quadro 2 – Classificação dos casos de teste segundo o critério importância

Escala Numérica	Escala Verbal	Definição
1	Desejável	Casos de teste que verificam funcionalidades que apresentam baixo impacto no valor do produto.
2	Importante	Casos de teste que verificam funcionalidades importantes para o cliente, mas que não impedem a entrega do produto.
3	Essencial	Casos de teste que verificam funcionalidades fundamentais ao atendimento das necessidades do cliente.

Fonte: Elaborado pelo autor

4.3.2.2 Tempo necessário para execução do teste manualmente

Os testes são selecionados com o objetivo de reduzir o tempo de execução dos testes;

Quadro 3 – Classificação dos casos de teste segundo o critério tempo de execução manual

Escala Numérica	Escala Verbal	Definição
1	Baixo	Casos de teste que levam até 1 minuto para serem executados manualmente
2	Médio	Casos de teste que levam de 1 a 5 minutos para serem executados
3	Alto	Casos de teste que levam mais que 5 minutos para serem executados

Fonte: Elaborado pelo autor

4.3.2.3 Repetitividade do caso de teste:

Os testes são selecionados com base no número de vezes que eles serão executados ao longo do projeto. Testes que são repetidos ao longo dos ciclos de desenvolvimento são bons candidatos à automação, pois permitem uma alta reutilização dos casos de teste automatizados.

Quadro 4 - Classificação dos casos de teste segundo o critério repetitividade

Escala Numérica	Escala Verbal	Definição
1	Baixo	Casos de teste que verificam uma condição única que não precisa ser verificada em versões posteriores do sistema
2	Médio	Casos de teste que verificam funcionalidades que precisam ser verificadas nos próximos ciclos de desenvolvimento até apresentarem um comportamento estável
3	Alto	Casos de teste que verificam funcionalidades que precisam ser verificadas a cada nova release

Fonte: Elaborado pelo autor

4.3.2.4 Necessidade de intervenção manual ou verificação visual de resultados

Os testes são selecionados com base na necessidade de intervenção manual durante a execução do caso de teste. Esse critério pode eliminar um caso de teste da estratégia de automação tendo em vista que algumas ações do usuário não podem ser automatizadas.

Quadro 5 - Classificação dos casos de teste segundo o critério necessidade de intervenção manual

Escala Numérica	Escala Verbal	Definição
0	Exige intervenção manual	Casos de teste que exigem intervenção manual
1	Não exigem intervenção manual	Casos de teste que não exigem intervenção manual

Fonte: Elaborado pelo autor

4.3.2.5 Estabilidade do caso de teste

Os testes são selecionados com base na probabilidade de mudança durante o projeto. Testes instáveis são maus candidatos à automação, pois requerem constante atualização.

Quadro 6 - Classificação dos casos de teste segundo o critério estabilidade

Escala Numérica	Escala Verbal	Definição
1	Baixa	Casos de teste ainda não passaram por nenhum ciclo de testes
2	Média	Casos de teste que foram testados e não apresentaram mudanças ao longo das últimas duas versões
3	Alta	Casos de teste que foram testados e não apresentaram mudança ao longo de mais de três versões

Fonte: Elaborado pelo autor

4.3.2.6 Necessidade de múltiplos valores de dados para execução do caso de teste

Os testes são selecionados baseando-se no número de entradas necessárias à sua execução. Testes que demandam um grande número de entradas são bons candidatos à automação, pois requerem do testador um grande esforço no preenchimento de formulários e validação de dados;

Quadro 7 - Classificação dos casos de teste segundo o critério necessidade de múltiplos valores de dados

Escala Numérica	Escala Verbal	Definição
1	Baixo	Casos de testes com até 2 entradas
2	Médio	Casos de testes com 3 a 5 entradas
3	Alto	Casos de teste com mais de 5 entradas

Fonte: Elaborado pelo autor

4.3.2.7 Necessidade de múltiplos ambientes:

Os testes são selecionados com base na quantidade de ambientes no qual o teste deve ser executado.

Quadro 8 - Classificação dos casos de teste segundo o critério necessidade de múltiplos ambientes

Escala Numérica	Escala Verbal	Definição
1	Baixa	Casos de testes a serem executados em um único ambiente
2	Média	Casos de testes a serem executados em até 3 ambientes
3	Alta	Casos de teste com mais de 3 ambientes

Fonte: Elaborado pelo autor

A partir desses critérios, o analista de testes pôde avaliar os casos de testes, atribuindo uma pontuação de acordo com a escala definida. Essa atribuição foi feita no início

da *sprint* considerando todas as funcionalidades que seriam implementadas até o final da *sprint*. A escolha de priorizar os casos de teste de maneira incremental, a cada *sprint*, justifica-se pelo fato que as funcionalidades especificadas podem mudar ao longo do projeto, tendo sua prioridade mudada com base nos critérios definidos.

Os pontos para cada critério do caso de teste foram somados resultando em uma pontuação que permite analisar de maneira comparativa os casos de testes prioritários à automação. Essa pontuação é mostrada na tabela abaixo:

Tabela 1 - Classificação dos casos de teste candidatos à automação

UC	TC	**Descrição**	**Importância**	**Repetitividade**	**Tempo**	**Dificuldade de execução manual**	**Estabilidade**	**Necessidade de múltiplas entradas**	**Necessidade múltiplos ambientes**	**Verificação manual de resultados**	**Total**	**Resultado**
UC01 - Fazer Login de cliente	TC01	Login feito com sucesso	2	3	1	1	3	1	3	1	14	CANDIDATO
	TC02	Email em branco	2	2	1	1	3	1	3	1	13	CANDIDATO
	TC03	Email inválido	2	2	2	3	3	3	3	1	18	CANDIDATO
	TC04	Senha em branco	2	2	1	1	3	1	3	1	13	CANDIDATO
	TC05	Senha inválida	2	2	1	3	3	3	3	1	17	CANDIDATO
UC02 - Solicitar redefinição de senha	TC06	Solicitação feita com sucesso	2	3	1	1	3	1	3	1	14	CANDIDATO
	TC07	Email em branco	2	2	1	1	3	1	3	1	13	CANDIDATO
	TC08	Email não cadastrado no sistema	2	2	1	1	3	1	3	1	13	CANDIDATO
	TC09	Email não cadastrado	2	2	1	3	3	3	3	1	17	CANDIDATO
UC03 - Redefinir senha	TC10	Senha redefinida	2	3	2	1	3	1	3	0	0	NÃO CANDIDATO
	TC11	Nova senha em branco	2	2	2	1	3	1	3	0	0	NÃO CANDIDATO
	TC12	Nova senha inválida	2	2	2	2	3	1	3	0	0	NÃO CANDIDATO
	TC13	Solicitação de recuperação expirada	2	2	2	3	3	1	3	0	0	NÃO CANDIDATO
UC04 - Requisitar peça tipo 1 via formulário	TC14	Requisição feita com sucesso	3	3	3	1	2	3	3	1	18	CANDIDATO
	TC15	Número serial em branco	3	3	2	1	2	3	3	1	17	CANDIDATO
	TC16	Número serial > 50 caracteres	3	3	3	3	2	3	3	1	20	CANDIDATO
	TC17	Número serial com caracteres não alfanuméricos	3	3	3	3	2	3	3	1	20	CANDIDATO
	TC18	Número serial já presente no formulário	3	3	3	2	2	3	3	1	19	CANDIDATO

	TC19	Id do cliente possui caracteres não alfanuméricos	3	3	3	3	2	3	3	1	20	CANDIDATO
	TC20	Id do cliente > 50 caracteres	3	3	3	3	2	3	3	1	20	CANDIDATO
	TC21	Motivo da requisição > 255 caracteres	3	3	3	3	2	3	3	1	20	CANDIDATO
UC04 - Requisitar peça tipo 1 via planilha	TC22	Requisição feita com sucesso	3	3	2	2	1	1	3	0	0	NÃO CANDIDATO
	TC23	Planilha anexada via arrastar e soltar	3	3	2	3	1	1	3	0	0	NÃO CANDIDATO
	TC24	Planilha com extensão inválida	3	3	3	2	1	3	3	0	0	NÃO CANDIDATO
	TC25	Planilha em branco	3	3	2	2	1	1	3	0	0	NÃO CANDIDATO
UC06 - Requisitar peça tipo 2 via formulário	TC26	Requisição feita com sucesso	3	3	2	2	2	3	3	1	18	CANDIDATO
	TC27	Id do Cliente em branco	3	3	2	1	2	2	3	1	16	CANDIDATO
	TC28	Id do cliente > 50 caracteres	3	3	2	3	2	2	3	1	18	CANDIDATO
	TC29	Id do cliente com caracteres não alfanuméricos	3	3	2	3	2	3	3	1	19	CANDIDATO
	TC30	Ordem de trabalho em branco	3	3	2	1	2	3	3	1	17	CANDIDATO
	TC31	Ordem de trabalho maior que 50 caracteres	3	3	2	3	2	3	3	1	19	CANDIDATO
	TC32	Ordem de trabalho com caracteres não alfanuméricos	3	3	2	3	2	3	3	1	19	CANDIDATO
	TC33	Quantidade em branco	3	3	2	1	2	3	3	1	17	CANDIDATO
	TC34	Quantidade menor ou igual a 0	3	3	2	2	2	3	3	1	18	CANDIDATO
	TC35	Quantidade com mais de 4 dígitos	3	3	2	2	2	3	3	1	18	CANDIDATO
	TC36	Quantidade com caracteres não numéricos	3	3	2	2	2	3	3	1	18	CANDIDATO
	TC37	Motivo da requisição > 255 caracteres	3	3	2	2	2	3	3	1	18	CANDIDATO
UC07 - Requisitar peça tipo 1 anexando planilha	TC38	Requisição feita com sucesso	3	3	1	1	1	1	3	0	0	NÃO CANDIDATO
	TC39	Planilha anexada via arrastar e soltar	3	3	1	1	1	1	3	0	0	NÃO CANDIDATO
	TC40	Planilha em formato inval.	3	3	1	3	1	3	3	0	0	NÃO CANDIDATO
	TC41	Planilha em branco	3	3	1	2	1	1	3	0	0	NÃO CANDIDATO
UC08 - Registrar Usuário Cliente	TC42	Cadastro realizado com sucesso	2	3	2	1	3	2	3	1	16	CANDIDATO
	TC43	Nome em branco	2	2	2	1	3	2	3	1	15	CANDIDATO
	TC44	Nome > 50 caracteres	2	2	2	3	3	3	3	1	18	CANDIDATO
	TC45	Email em branco	2	2	2	1	3	2	3	1	15	CANDIDATO
	TC46	Email em formato inválido	2	2	2	3	3	3	3	1	18	CANDIDATO
	TC47	Email > 50 caracteres	2	2	2	3	3	3	3	1	18	CANDIDATO

Fonte: Elaborado pelo autor

A partir dessa avaliação foi possível selecionar os casos de testes candidatos à automação, permitindo também estabelecer uma prioridade para o seu desenvolvimento, a fim de implementar primeiro os casos de testes com maior valor para a estratégia de automação. Todos os casos de teste definidos como candidatos foram automatizados, mas o desenvolvimento de alguns testes foi dificultado por mudanças na interface do sistema, principalmente os testes relacionados aos casos de uso UC04 –“ Requisitar peça tipo 1 via formulário” e UC-06 - “Requisitar peça tipo 2 via formulário” devido as constantes mudanças na sua interface no início da *sprint*, devido a não terem uma alta estabilidade.

Nas próximas seções serão descritas as atividades de desenvolvimento dos casos de teste selecionados.

4.3.3 *Desenvolvimento dos casos de teste automatizados*

A versão inicial dos casos de teste automatizados foi feita no Selenium IDE, através da captura da interação do testador ao executar um determinado caso de teste. A figura 6 mostra um script com as ações do usuário e dados de entrada para o caso de teste “Registrar Usuário”.

Figura 6: Script gerado pelo Selenium IDE para o caso de teste Registrar Usuário

	RegisterCustomerUser		
Abrir página inicial	open	/projetos/project/Admin	
Selecionar a página de Usuário	clickAndWait	link=Customer User	
Selecionar a opção criar	clickAndWait	link=Create	
Informar o nome do usuário	type	id=Name	Customer 1
Informar o email do usuário	type	id=Email	customer1@lsbd.ufc.br
Marcar se o usuário está ativo	click	css=ins.iCheck-helper	
Confirmar a criação do usuário	clickAndWait	css=button.btn.btn-blue2	
Verificar a confirmação	assertText	css=div.alert.alert-success	× Customer User successfully saved

Fonte: Elaborado pelo autor

Em seguida esses scripts foram exportados para a linguagem Java, permitindo sua edição de maneira programática. A figura 7 mostra o script mostrado na figura 6 em linguagem Java.

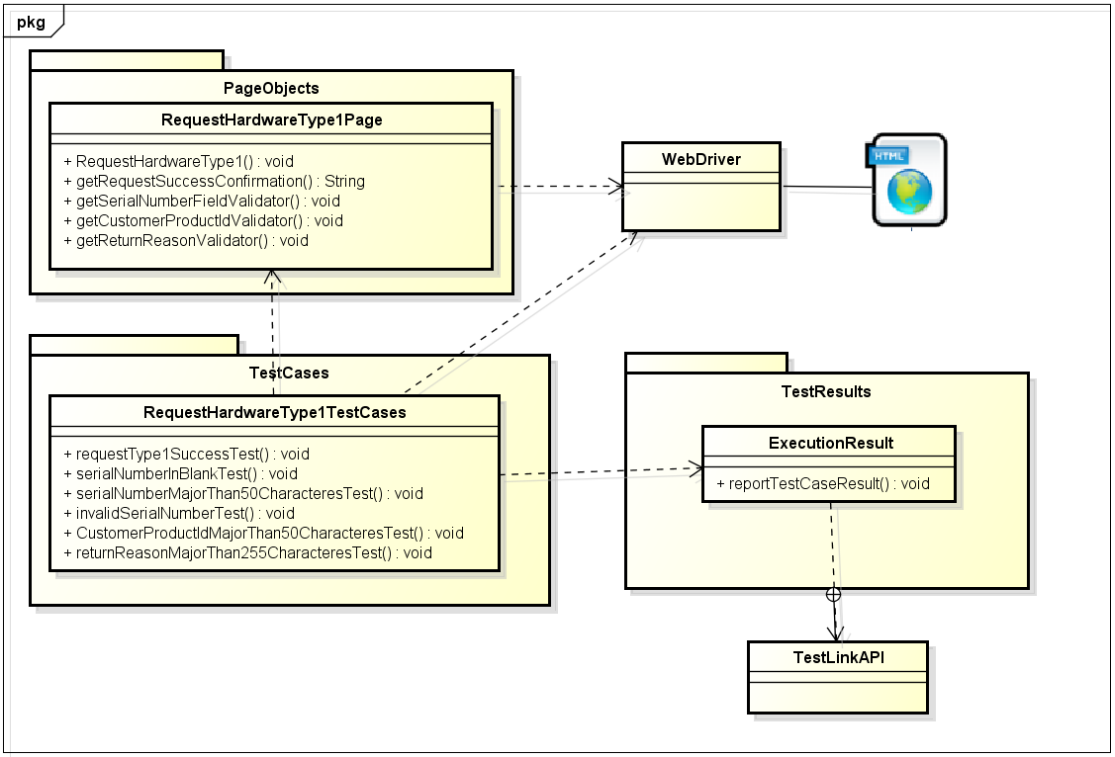
Figura 7: Script gerado com o Selenium IDE e exportado para Java

```
@Test
public void testRegisterCustomerUser() throws Exception {
  driver.get(baseUrl + "/projetos/project/Admin");
  driver.findElement(By.linkText("Customer User")).click();
  driver.findElement(By.linkText("Create")).click();
  driver.findElement(By.id("Name")).clear();
  driver.findElement(By.id("Name")).sendKeys("Customer 1");
  driver.findElement(By.id("Email")).clear();
  driver.findElement(By.id("Email")).sendKeys("customer1@lsbd.ufc.br");
  driver.findElement(By.cssSelector("ins.iCheck-helper")).click();
  driver.findElement(By.cssSelector("button.btn.btn-blue2")).click();
  assertEquals("Ã— Customer User successfully saved",
          driver.findElement(By.cssSelector("div.alert.alert-success")).getText());
}
```

Fonte: Elaborado pelo autor

O projeto de criação e execução de testes automatizados foi desenvolvido com o apoio do ambiente de desenvolvimento integrado (IDE) Eclipse[1]. A estrutura do projeto é mostrada na imagem abaixo e detalhada a seguir:

Figura 8: Estrutura do projeto de testes automatizados em Java



Fonte: Elaborado pelo autor

- Módulo *Page Objects*: Composto pelas classes *page objects* que representam as páginas web do sistema e são responsáveis pela execução

[1] http://www.eclipse.org/

dos passos do caso de teste, desde a navegação entre páginas e inserção de dados até a verificação dos resultados;

- Módulo Test Cases: Agrupa todas as classes responsáveis pela execução dos testes do sistema. Nessas classes, os testes tem seus valores de entrada configurados e os resultados dos testes são avaliados utilizando o framework Junit;
- Módulo *Test Results*: Responsável pelo registro dos resultados dos testes, Os resultados obtidos através do Junit são enviados para o TestLink.

A figura 9 mostra uma caso de teste implementado a partir da estrutura do projeto mostrada. Inicialmente no método *setUp()* é feita a configuração do caso de teste de acordo com o navegador. Em seguida é criado uma classe CustomerUserPage, responsável pela interação com a página web e captura das respostas do sistema. Por último os valores de resposta são verificados com os métodos *assert* do Junit e enviados para o TestLink.

Figura 9: Caso de teste em Java

```
public class RegisterUserTest {
    private static WebDriver driver;
    private StringBuffer verificationErrors = new StringBuffer();

    @BeforeClass
    public static void setUp() throws Exception {

        //Configuração so Selenium Web Driver para executar no navegador Mozilla Firefox
        driver = new FirefoxDriver();
        driver.manage().timeouts().implicitlyWait(3, TimeUnit.SECONDS);
    }

    @Test
    public void TC023RegisterUserSuccess() throws TestLinkAPIException{
        String resultado = null;
        String nota = null;

        //Criação do Page Object responsável pelo registro de um novo usuário no sistema
        CustomerUserPage page = new CustomerUserPage(driver);
        page.registerCustomerUser("Gerbson Lima", "gerbson.lima@lsbd.ufc.br");

        try{
            //Verificação do resultado do teste usando Junit
            assertEquals("Customer User successfully saved.",page.getConfirmation(driver));
            resultado = TestLinkAPIResults.TEST_PASSED;
        }catch(Error e){
            verificationErrors.append(e.toString());
            resultado = TestLinkAPIResults.TEST_FAILED;
            nota = e.getMessage();
        }catch(Exception e){
            verificationErrors.append(e.toString());
            resultado = TestLinkAPIResults.TEST_BLOCKED;
            nota = e.getMessage();
        }finally{
            //Envio do resultado do test para o Testlink
            ExecutionResult.reportTestCaseResult(IConstants.PROJETO, IConstants.PLANO,
                    "TC-23", IConstants.BUILD, nota, resultado);
        }
    }
```

Fonte: Elaborado pelo autor

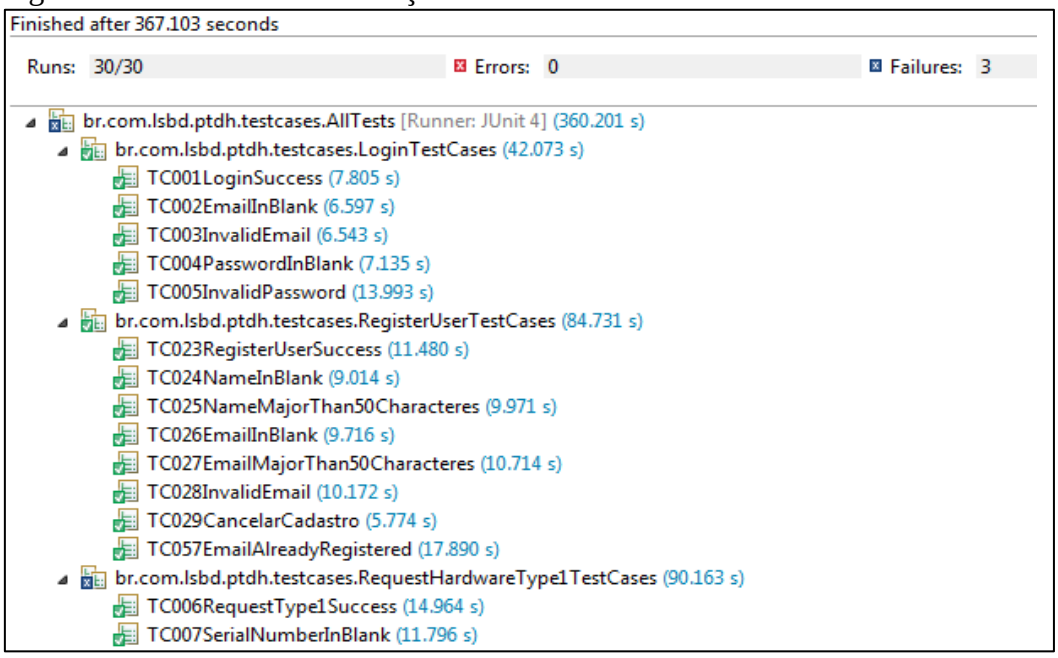
4.3.4 *Execução dos testes automatizados*

Após a implementação dos casos de teste e a definição dos seus respectivos valores de entrada, foi iniciado os ciclos de execução dos testes automatizados. Os testes foram executados em três versões do sistema PTDH, nos navegadores Google Chrome, Mozilla Firefox e Internet Explorer. Para cada versão foram executados 30 casos de teste, divididos em 4 suítes de acordo com o caso de uso alvo dos testes.

A execução dos testes em múltiplos ambientes foi facilitada pela interoperabilidade oferecida pelo Selenium Web driver, tendo em vista que para executar os testes em diferentes navegadores, foi necessário apenas substituir os drivers do navegador nas configurações do projeto, permitindo que os testes sejam reproduzidos em vários ambientes sem custos de implementação adicionais.

A execução dos testes foi apoiada pelo framework de automação de testes Junit integrado a IDE Eclipse. Através do Junit, foi possível acompanhar a execução dos casos de teste, visualizando os casos de teste executados, com os seus respectivos resultados (sucesso ou falha) e tempos de execução, e os casos de teste que não foram executados devido a falhas decorrentes de erros de implementação, conforme mostra a figura 10. Para cada caso de teste que apresenta uma falha, o Junit mostra o rastro de falha indicando a área do código onde aconteceu a falha, facilitando a identificação e correção dos testes automatizados.

Figura 10: Resultados da execução dos testes com Junit



Fonte: Elaborado pelo autor

Ao final da execução de um caso de teste, os resultados são enviados para a ferramenta de gestão de testes Testlink, utilizando uma API de acesso à ferramenta. Para cada caso de teste executado a API registra o resultado do caso de teste na ferramenta. Para casos de teste que falha a API também registra o rastro de falha exibido pelo Junit, sendo possível acompanhar os resultados diretamente pelo Testlink;

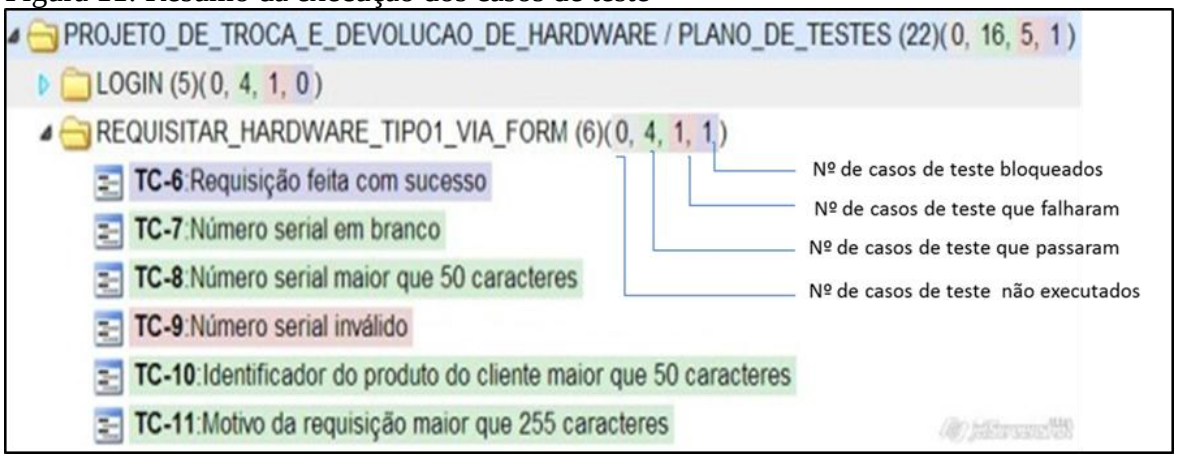
4.3.5 *Verificação e correção de defeitos nos testes automatizados*

Após a execução dos testes, foi necessário verificar os casos de teste que não foram executados devidos a falhas de implementação, visando identificar e corrigir os erros. Tendo como base os rastros de falha exibidos pelo Junit, os erros eram identificados e corrigidos, e os testes reexecutados isoladamente e depois em conjunto com os demais testes da suíte, a fim de garantir que as mudanças efetuadas não impactaram nos outros casos de teste. A verificação e correção de defeitos nos testes mostrou-se uma atividade importante, pois conforme eram feitas mudanças na interface do sistema alguns testes automatizados quebravam, e realizar essa manutenção permitiu que todos os testes criados pudessem ser executados, assegurando a cobertura total dos testes automatizados.

4.3.6 *Visualização de resultados dos testes*

A análise dos resultados do teste foi feita a partir dos resultados registrados no Testlink. A partir da ferramenta foi possível visualizar um resumo dos ciclos de execução de testes realizados, mostrando o resultado para cada caso de teste e sumarizando os resultados para cada suíte e para todo o plano de testes. Conforme mostra a figura 11, essa visualização dá ao testador um *feedback* geral sobre a qualidade do sistema e uma visão de cada funcionalidade testada, permitindo definir por exemplo, uma ação corretiva no caso de um grande número de casos de teste falharem para determinada funcionalidade, ou informar sobre a necessidade de uma manutenção nos scripts de teste, quando existirem casos de teste bloqueados devido a erros de implementação.

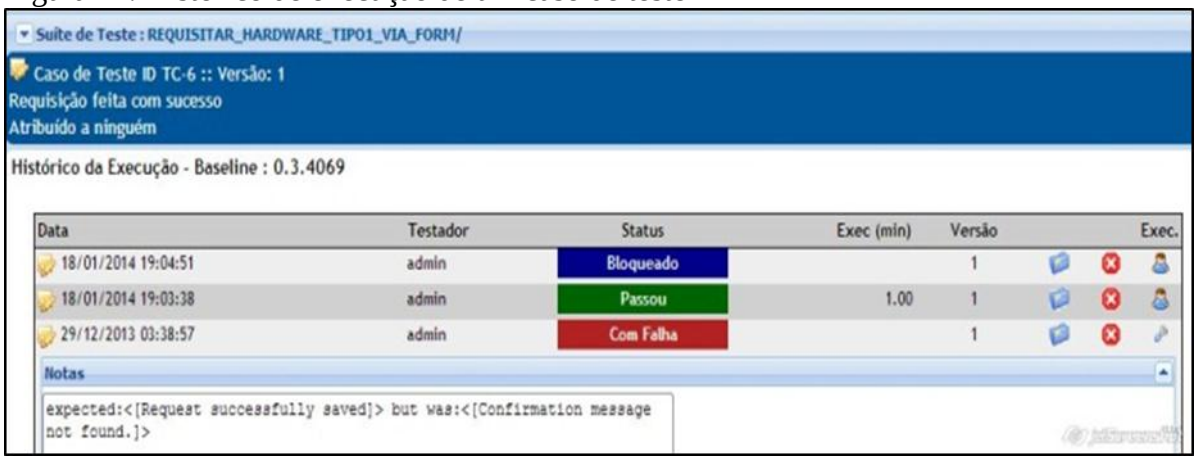
Figura 11: Resumo da execução dos casos de teste



Fonte: Elaborado pelo autor

Conforme mostra a figura 12, também é possível visualizar o histórico de execução de um caso de teste. Para as execuções que resultam em falha o Testlink mostra uma nota com a falha registrada pelo Junit. Essa funcionalidade é particularmente útil, pois dá um *feedback* mais detalhado da falha encontrada, facilitando a descrição dos problemas encontrados ao time de desenvolvimento.

Figura 12: Histórico de execução de um caso de teste



Fonte: Elaborado pelo autor

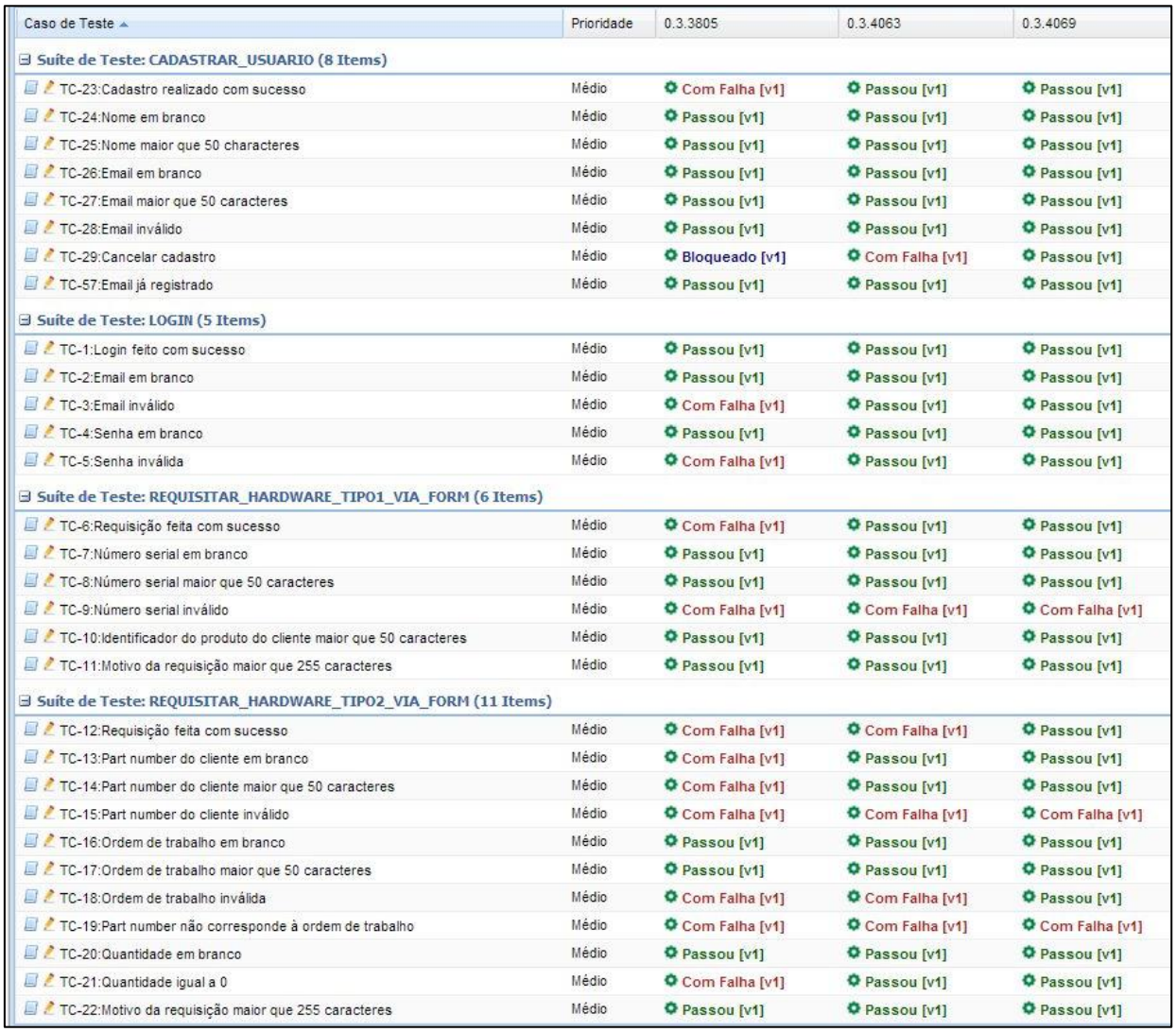
A visualização dos resultados é importante para acompanhar a execução dos testes, mas não fornece uma visão ampla da qualidade do sistema. Dessa forma, foi incluída uma atividade para a geração de relatórios e métricas, conforme é mostrado na próxima seção.

4.3.7 *Geração de relatórios e métricas*

Após a execução de cada ciclo de testes, foi possível gerar relatórios e métricas sobre a execução dos testes através do Testlink. Esses relatórios são úteis para o

acompanhamento da execução dos testes, pois permitem visualizar os resultados de maneira estruturada. Um desses relatórios é uma matriz que mostra o resultado de cada caso de teste, ao longo de várias *releases*, conforme mostra a imagem abaixo:

Figura 13: Matriz de execução dos testes ao longo de várias releases
s



Caso de Teste	Prioridade	0.3.3805	0.3.4063	0.3.4069
Suíte de Teste: CADASTRAR_USUARIO (8 Items)				
TC-23:Cadastro realizado com sucesso	Médio	Com Falha [v1]	Passou [v1]	Passou [v1]
TC-24:Nome em branco	Médio	Passou [v1]	Passou [v1]	Passou [v1]
TC-25:Nome maior que 50 characteres	Médio	Passou [v1]	Passou [v1]	Passou [v1]
TC-26:Email em branco	Médio	Passou [v1]	Passou [v1]	Passou [v1]
TC-27:Email maior que 50 caracteres	Médio	Passou [v1]	Passou [v1]	Passou [v1]
TC-28:Email inválido	Médio	Passou [v1]	Passou [v1]	Passou [v1]
TC-29:Cancelar cadastro	Médio	Bloqueado [v1]	Com Falha [v1]	Passou [v1]
TC-57:Email já registrado	Médio	Passou [v1]	Passou [v1]	Passou [v1]
Suíte de Teste: LOGIN (5 Items)				
TC-1:Login feito com sucesso	Médio	Passou [v1]	Passou [v1]	Passou [v1]
TC-2:Email em branco	Médio	Passou [v1]	Passou [v1]	Passou [v1]
TC-3:Email inválido	Médio	Com Falha [v1]	Passou [v1]	Passou [v1]
TC-4:Senha em branco	Médio	Passou [v1]	Passou [v1]	Passou [v1]
TC-5:Senha inválida	Médio	Com Falha [v1]	Passou [v1]	Passou [v1]
Suíte de Teste: REQUISITAR_HARDWARE_TIPO1_VIA_FORM (6 Items)				
TC-6:Requisição feita com sucesso	Médio	Com Falha [v1]	Passou [v1]	Passou [v1]
TC-7:Número serial em branco	Médio	Passou [v1]	Passou [v1]	Passou [v1]
TC-8:Número serial maior que 50 caracteres	Médio	Passou [v1]	Passou [v1]	Passou [v1]
TC-9:Número serial inválido	Médio	Com Falha [v1]	Com Falha [v1]	Com Falha [v1]
TC-10:Identificador do produto do cliente maior que 50 caracteres	Médio	Passou [v1]	Passou [v1]	Passou [v1]
TC-11:Motivo da requisição maior que 255 caracteres	Médio	Passou [v1]	Passou [v1]	Passou [v1]
Suíte de Teste: REQUISITAR_HARDWARE_TIPO2_VIA_FORM (11 Items)				
TC-12:Requisição feita com sucesso	Médio	Com Falha [v1]	Com Falha [v1]	Passou [v1]
TC-13:Part number do cliente em branco	Médio	Com Falha [v1]	Passou [v1]	Passou [v1]
TC-14:Part number do cliente maior que 50 caracteres	Médio	Com Falha [v1]	Passou [v1]	Passou [v1]
TC-15:Part number do cliente inválido	Médio	Com Falha [v1]	Com Falha [v1]	Com Falha [v1]
TC-16:Ordem de trabalho em branco	Médio	Passou [v1]	Passou [v1]	Passou [v1]
TC-17:Ordem de trabalho maior que 50 caracteres	Médio	Passou [v1]	Passou [v1]	Passou [v1]
TC-18:Ordem de trabalho inválida	Médio	Com Falha [v1]	Com Falha [v1]	Passou [v1]
TC-19:Part number não corresponde à ordem de trabalho	Médio	Com Falha [v1]	Com Falha [v1]	Com Falha [v1]
TC-20:Quantidade em branco	Médio	Passou [v1]	Passou [v1]	Passou [v1]
TC-21:Quantidade igual a 0	Médio	Com Falha [v1]	Passou [v1]	Passou [v1]
TC-22:Motivo da requisição maior que 255 caracteres	Médio	Passou [v1]	Passou [v1]	Passou [v1]

Fonte: Elaborado pelo autor

Através dessa matriz é possível ter uma visão global dos resultados, permitindo acompanhar a evolução da qualidade do sistema e caso necessário dimensionar ações de melhoria dentro do projeto. Por exemplo, casos de teste que falham ao longo de três releases consecutivas podem indicar que os problemas não estão sendo corrigidos realmente pela equipe de desenvolvimento ou que a funcionalidade testada apresenta uma complexidade que demanda um maior tempo de desenvolvimento antes de sua liberação para testes.

Outro relatório importante que pôde ser gerado foi um conjunto de métricas para cada *baseline* do sistema, mostrando o número de casos de testes definidos, o número de casos de testes executados e os resultados de casos de testes por status. Na figura 14 é possível visualizar a execução dos testes ao longo de três versões consecutivas do sistema. Para as três versões foram executados 30 casos de teste automatizados. Na versão 0.3.3805, 12 casos de testes (40%) falharam e 1 caso de teste (3.3%) foi bloqueado devido a falhas de implementação. As falhas encontradas foram então reportadas pelo analista de testes e corrigidas pelo time de desenvolvimento. Como resultado, nas versões posteriores houve uma redução dos casos de teste com falha, sendo que na última versão, apenas 3 casos de teste (3.3%) falharam, o que demonstra uma evolução da qualidade do sistema ao longo das versões.

Figura 14: Métricas da execução de testes automatizados

Métricas do Plano de Teste

Projeto de Teste : PTDH
Plano de Teste : PLANO_DE_TESTES

- Resultado das Métricas de Teste por Baseline

Baseline	Atribuído	Não Executado	[%]	Passou	[%]	Com Falha	[%]	Bloqueado	[%]	Completado [%]
0.3.3805	30	0	0.0	17	56.7	12	40.0	1	3.3	100.0
0.3.4063	30	0	0.0	24	80.0	6	20.0	0	0.0	100.0
0.3.4069	30	0	0.0	27	90.0	3	10.0	0	0.0	100.0

Fonte: Elaborado pelo autor

4.3.8 *Verificar aderência aos objetivos da automação*

Após a implantação dos testes automatizados e de sua execução ao longo de algumas releases do sistema foi feita uma análise do processo de automação aplicado visando identificar se os objetivos definidos na seção 4.3.1 foram atendidos. Essa análise é importante, pois permite avaliar se existem benefícios reais que justifiquem a implantação de testes automatizados no projeto. Além de ser parte importante do processo, essa atividade faz parte dos resultados desse trabalho de pesquisa, sendo mostrada portanto na seção 6 – Resultados da Pesquisa.

5 DIFICULDADES E LIÇÕES APRENDIDAS

Implantar a automação de testes em uma organização que não possui nenhuma atividade de testes automatizados é uma tarefa complexa. Durante esse trabalho foram encontradas algumas dificuldades técnicas e relacionadas ao processo de implantação dos testes automatizados.

A primeira dificuldade enfrentada durante o trabalho foi a falta de conhecimento técnico durante a criação dos primeiros casos de teste automatizados, pois demandou um estudo detalhado das tecnologias usadas, principalmente do Selenium WebDriver. Entretanto, a realização dos testes, inicialmente, utilizando o Selenium IDE permitiu conhecer o Selenium Webdriver com uma menor curva de aprendizado, pois facilitou o aprendizado de como a ferramenta interage com as páginas web e como os resultados são verificados.

Outro ponto a ser levantado, foram as mudanças que aconteceram na interface do sistema durante o desenvolvimento da primeira versão dos casos de teste automatizados. Como o Selenium interage com as páginas através da identificação dos seus elementos, algumas mudanças feitas nas páginas causaram a quebra dos scripts de teste, gerando um retrabalho para mapear as mudanças nesses elementos. Esse problema foi minimizado a partir da modularização do projeto de teste em classes responsáveis pela interação com as páginas web e classes específicas para teste, que permitiram que as mudanças na interface impactassem apenas nas classes responsáveis pela captura dos dados na página.

Outra dificuldade encontrada foi em relação à definição dos casos de teste automatizados a partir da documentação de casos de teste, pois alguns casos de teste estavam englobando a verificação de vários resultados, demandando a interpretação do testador a cerca do resultado de um teste. Para a execução automática foi preciso dividir esses casos de teste em casos de teste mais granulares, com entradas e resultados únicos e verificáveis, que fossem usados pelo Junit para reportar se o teste passou ou falhou.

Foi possível perceber também, a dificuldade em realizar um trabalho tendo como base um projeto real, devido a limitações de prazo e a menor capacidade de introduzir mudanças nos processos já adotados no projeto.

6 RESULTADOS DA PESQUISA

A pesquisa teve como objetivo o desenvolvimento e implantação de um processo de automação de testes do projeto PTDH e avaliar o impacto dessa iniciativa para a atividade de testes do projeto. Para atingir esse objetivo foram automatizados 30 casos de testes, divididos em 4 suítes de teste, executadas ao longo de 3 versões sucessivas do sistema em três ambientes distintos.

O primeiro aspecto a ser considerado ao avaliar o esforço da automação é a adaptação dos casos de teste do projeto para casos de teste mais detalhados. Essa adaptação foi feita pelo automatizador de testes, a partir dos casos de teste candidatos a automação. È importante ressaltar que o esforço dessa atividade depende da qualidade dos casos de teste já existentes. Para os casos de teste do projeto esse esforço foi baixo, tendo em vista que a especificação de testes possuía uma boa cobertura de cenários, sendo necessário apenas detalhar esses cenários com valores de entrada e resultados mais específicos.

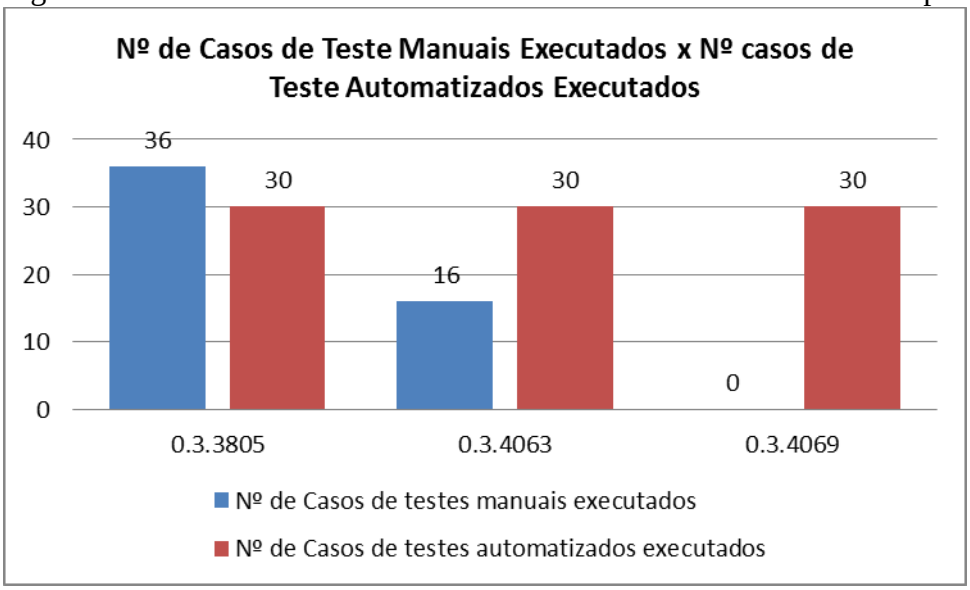
A tarefa mais dispendiosa foi o desenvolvimento dos casos de teste automatizados, pois demandou um esforço iterativo e incremental de desenvolvimento, correção e teste dos scripts desenvolvidos. Nesse ponto, é importante destacar o papel da atividade “Seleção de casos de teste para automação”, descrita na seção 4.3.2, que permitiu empregar os esforços de desenvolvimento apenas para automatizar testes que, potencialmente, pudessem contribuir de forma mais significativa para atingir os objetivos da automação. Foi possível perceber um ganho expressivo em produtividade na criação dos scripts, após a criação dos *page objects*, responsáveis pelo acesso as páginas web, que permitiu que a criação dos *test cases* com Junit demandasse apenas a chamada dos métodos criados com novos valores de entrada. Os mesmos benefícios se aplicam a correção de defeitos nos scripts, pois mudanças na lógica de acesso a página foram feitas apenas nos *page objects*.

Os resultados obtidos na execução dos testes automatizados foram considerados satisfatórios, pois permitiram reduzir o tempo de execução dos testes e aumentaram a cobertura dos testes por versão. A execução automatizada permitiu que os testes fossem executados em todos os ambientes, pois uma vez criados os scripts de teste, foi possível executá-los em vários navegadores, apenas mudando as configurações do projeto. Esses ganhos em tempo justificam a adoção da automação de testes no projeto, pois conforme o sistema for evoluindo, irão aumentar também o número de cenários de teste a serem verificados, e consequentemente o esforço na execução dos mesmos irá aumentar. Essa

característica reforça ainda mais a adoção de testes automatizados, pois a geração de releases para o sistema PDTH acontece duas vezes na semana, o que demanda do testador um grande esforço com testes de regressão.

Em relação à cobertura dos testes foi possível perceber que a automação de um conjunto de testes permite ao testador direcionar seus esforços para a execução de testes de novas funcionalidades ou dos testes difíceis de automatizar, conforme foi apontado na seção 5.4.3 relativa a seleção de casos de teste para automação. Isso permite atingir uma maior cobertura de testes por release, melhorando a qualidade do sistema ao longo de sua evolução. A figura 17 mostra o número de casos de teste executados para as funcionalidades alvo da automação de testes, fazendo um comparativo entre o número de casos de teste que puderam ser executados com e sem automação. É possível observar que na primeira versão na qual a automação foi incluída – 0.3.3805, existe uma pequena diferença entre o número de casos de teste executados com automação em relação a execução puramente manual. Isso acontece, pois para funcionalidades novas geralmente são executados todos os casos de teste manualmente. Entretanto, com a evolução do sistema e diante de limitações de prazos e recursos já discutidas, o analista de testes geralmente prioriza o teste das novas funcionalidades, diminuindo a cobertura das funcionalidades já implementadas, o que fez, por exemplo, com que na última versão abordada nesse trabalho, nenhum teste manual pudesse ser executado para as funcionalidades mais antigas, conforme mostra o gráfico abaixo.

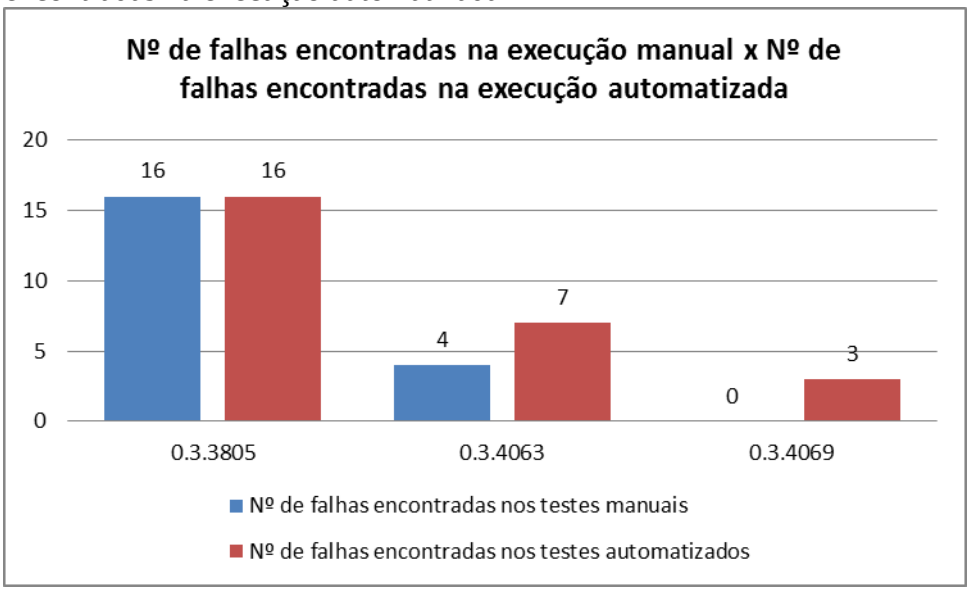
Figura 15: Gráfico com o número de casos de teste executados antes e após a automação



Fonte: Elaborado pelo autor

A eficácia dos testes automatizados em descobrir as falhas no sistema também apresentou resultados positivos. A maior parte das falhas no sistema relacionadas às verificações previstas nos scripts de testes, e identificadas na execução manual também foram capturadas na execução automática. É importante ressaltar que a execução manual, conseguiu identificar problemas nos sistema que não foram cobertos pelos casos de testes automatizados, como por exemplo, problemas de deslocamento de elementos na interface, seleção de vários componentes de maneira exploratória e navegação no sistema através do teclado. A figura 18, apresenta um comparativo entre o número de falhas encontradas antes e após a execução dos testes automatizados. Analisando o gráfico, é possível perceber um aumento do número de falhas encontradas com a inclusão de testes automatizados. Isso decorre, principalmente, da maior cobertura que a combinação entre testes manuais e testes automáticos traz a cada ciclo de testes. Além disso, a execução automática permite executar todos os testes desenvolvidos já que não depende do esforço direto do analista o que aumenta a qualidade do sistema, principalmente para versões onde os testes manuais não podem ser totalmente executados.

Figura 16: Gráfico com o nº de falhas encontradas na execução manual x nº de falhas encontradas na execução automatizada



Fonte: Elaborado pelo autor

Para apoiar a avaliação da estratégia de automação no Projeto PTDH foi realizada uma entrevista com o analista de testes do projeto, responsável por todas as atividades de teste, desde a especificação até a execução e reporte de falhas. Obter a percepção do analista de testes do projeto é um instrumento importante para a avaliação dos testes automatizados, pois permite fazer uma avaliação qualitativa dos resultados da implantação de testes automatizados e dos seus benefícios. Entretanto, é importante enfatizar que apesar de ser um

feedback importante do trabalho, a entrevista apresenta baixa relevância estatística, por se tratar da opinião de um único indivíduo.

Ao comparar o esforço na especificação de testes manuais com a especificação e desenvolvimento de testes automatizados o analista de testes considerou que a especificação dos testes automatizados é mais lenta, pois exige uma definição mais detalhada das condições a serem testadas, incluindo a definição de valores específicos ao invés de apresentar apenas os critérios a serem validados. Entretanto ele ressaltou que apesar da especificação dos testes manuais ser mais rápida, ela demanda do testador maior atenção no momento da execução dos testes, pois demanda uma interpretação mais detalhada dos casos de teste para a definição de valores de entrada a serem testados. Dessa forma, ele considerou que a definição mais detalhada necessária para os testes automatizados, aumenta também a qualidade dos testes manuais, pois permite declarar de maneira explícita os cenários a serem testados, diminuindo as chances de cenários não cobertos ou ambíguos na execução dos testes.

Ao ser perguntado sobre o impacto dos testes automatizados na execução dos testes, o analista de testes considerou os testes automatizados mais rápidos que os testes manuais, principalmente para os casos de teste que envolvem grandes quantidades de dados de entrada, principalmente porque os testes têm que ser executados em quatro navegadores.

O analista de testes considerou que os testes automatizados contribuíram para a qualidade do sistema, principalmente por permitir a identificação de falhas em casos de teste que não puderam ser testados manualmente ao longo de mais de uma versão, devido à limitação de tempo e recursos de teste. Ele ressaltou que muitas funcionalidades do sistema, que são executadas sem falhas em algum dos ciclos de teste, passam por sucessivas versões do sistema sem serem testadas, devido a prioridade de teste dada a novas funcionalidades ou a funcionalidades com falhas. Segundo ele, o problema é que essas funcionalidades, mesmo já tendo sido aprovadas pelos testes, podem apresentar falhas ao longo do tempo, devido ao impacto causado na implementação de nova funcionalidades, enfatizando a importância dos testes automáticos para garantir que ao menos os fluxos básicos estejam funcionando para todo o sistema.

Ao comparar a eficiência dos testes manuais em relação aos testes automatizados, o analista julgou os testes automatizados mais eficientes para os casos de teste que verificaram um maior número de entradas, com regras de validação mais complexas ou repetitivas como, por exemplo, validação de e-mail, contagem do limite de caracteres dos

campos e validação das regras de formação de número seriais. Para cenários desse tipo, a analista julgou que além de mais demorada a execução manual é mais propensa a erros, já que as regras são verificadas de maneira visual. Entretanto ela julgou que os testes manuais são mais eficientes na identificação de falhas na interação com o sistema, pois permitem explorar novas entradas e fluxos que não estão explicitamente cobertos nos casos de teste, como por exemplo, clique múltiplo em botões, preenchimento formulários com validação dinâmica (*javascript)* e testes de interface.

De maneira geral o analista de testes avaliou os testes automatizados de maneira positiva, levantando a possibilidade de utilização em outros projetos. Entretanto ele levantou a necessidade de treinamento da equipe de testes, tendo em vista que o LSBD ainda não utiliza testes automatizados em seus projetos, bem como uma adaptação do processo adotado no laboratório para incluir as atividades de automação.

7 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho de pesquisa apresentou um estudo de caso na implantação de testes funcionais automatizados em um Projeto Web, a partir da definição de um processo de testes automatizados e da sua aplicação em um projeto real.

Tendo como base o Modelo de Melhoria do Processo de Testes Brasileiro (MPT. Br) foram definidas atividades necessárias ao desenvolvimento e execução de testes automatizados tendo como base a ferramenta de automação de testes Selenium. Através da execução dessas atividades, foi possível perceber os benefícios da automação de testes para a agilidade na execução dos testes em um projeto e, consequentemente, para a melhoria da qualidade do sistema desenvolvido, bem como as dificuldades relacionadas a esse processo.

O principal benefício percebido foram os ganhos de tempo na execução dos testes automatizados e a frequência com que eles podem ser executados, já que uma vez desenvolvidos, podem ser reproduzidos em vários ambientes e ao longo de várias versões a um baixo custo. Essa característica ressalta o potencial dos testes automatizados para testes de regressão.

A execução automatizada mostrou-se mais eficiente que a execução manual para casos de teste que apresentam validações mais complexas. Entretanto foi possível perceber que mesmo com a utilização de testes automáticos, o uso de testes manuais é indispensável, já

que muitos problemas no sistema não foram identificados nos testes automáticos, mas foram percebidos na execução manual, principalmente em testes exploratórios.

A partir da opinião da analista de testes do projeto possível perceber que a especificação dos testes automatizados demanda um maior esforço do analista de testes, mas que melhora a qualidade dos casos de teste desenvolvidos, contribuindo também para a execução manual.

Apesar de apresentar alguns resultados que evidenciaram o potencial da automação de testes, este trabalho apresentou algumas limitações como o reduzido número de casos de teste devido ao estágio inicial do sistema que possuía poucas funcionalidades implementadas no período da realização dessa pesquisa e a dificuldade em lidar com as exigências de um projeto real, que limitou o impacto do processo de automação no processo de testes existente na organização.

Por fim, foi possível concluir que a implantação de testes automatizados é um processo complexo, que demanda uma mudança, tanto no processo de testes quanto na capacitação técnica dos analistas de testes de uma organização, mas que seu uso de forma sistemática pode trazer benefícios reais a um projeto de desenvolvimento.

Como trabalho futuro, existe a possibilidade de extensão dos casos de teste automatizados para outras funcionalidades do sistema e a execução dos testes por um período maior de tempo, a fim de verificar se os benefícios percebidos são evidenciados ao longo da evolução do sistema. Além disso, as atividades definidas no processo de automação de testes podem ser utilizadas para outros projetos, tendo em vista que apresentam atividades básicas para a implantação de testes automatizados.

REFERÊNCIAS

BERNARDO, P. C. **Padrões de Testes Automatizados**. 2011. Dissertação (Mestrado em Engenharia Industrial) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2011.

BUDNIK, C. J.; CHAN, W. K.; KAPFHAMMER, G. M. Bridging the Gap between the Theory and Practice of Software Test Automation. *In*: 32ND INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 2010. Cape Town, South Africa.

CAETANO, C. Melhores Práticas E Desafios na Automação de Testes, 2013. Disponível em: http://www.qualister.com.br/blog/melhores-praticas-e-desafios-na-automacao-de-testes. Acesso em: 22 mar. 2014.

CERVANTES, A. Exploring the use of a test automation framework. *In*: AEROSPACE CONFERENCE.2009, Big Sky.

CHIAVEGATTO, R. *et al*. Especificação e Automação Colaborativas de Testes utilizando a técnica BDD. 2013.

COLLINS, E. F.; LOBÃO, L. M. A. Experiência em Automação do Processo de Testes em Ambiente Ágil com SCRUM e ferramentas OpenSource. *In*: IX SIMPÓSIO BRASILEIRO DE QUALIDADE DE SOFTWARE, 2010, Belém.

COLLINS, E. F.; LUCENA, V. F. Software Test Automation practices in agile development environment: An industry experience report. *In:* 7TH INTERNATIONAL WORKSHOP ON AUTOMATION OF SOFTWARE TEST (AST)**,** 2012, Zurich.

CRAIG, R.; JASKIEL, S. Systematic Software Testing. Norwood: Artech House Publishers, 2002.

DIAS, A. C. Introduçao ao Teste de Software. **Engenharia de Software Magazine**, 2008.

EELES, P. Capturing Architectural Requirements. IBM, 2005. Disponível em: <http://www.ibm.com/developerworks/rational/library/4706.html>. Acesso em: 17 dez. 2013

FANTINATO, M., CUNHA, A., DIAS, S., MIZUNO, S., E CUNHA, C. AutoTest–Um Framework Reutilizável para a Automação de Teste Funcional de Software. *In*: III SIMPÓSIO BRASILEIRO DE QUALIDADE DE SOFTWARE, 2004, Brasília.

FEWSTER, M e GRAHAM, D. **Software Test Automation - Effective use of test execution tools**. Addison-Wesley, 1994.

HOODA, R. An Automation of Software Testing: A Foundation for the Future. *In*: INTERNATIONAL JOURNAL OF LATEST RESEARCH IN SCIENCE AND TECHNOLOGY, v. 1, n. 2, p. 152–154, 2012.

IEEE Standard 610-1990: IEEE Standard Glossary of Software Engineering Terminology, IEEE Press.

KOSCIANSKI, A., SOARES, M. S. **Qualidade de Software**. 2.ed. São Paulo: Novatec, 2007.

LIMA, T. *et al*. Usando o SilkTest para automatizar testes: um Relato de Experiência. *In:* 6TH BRAZILIAN WORKSHOP ON SYSTEMATIC AND AUTOMATED SOFTWARE TESTING, 2012, Natal.

MEDEIROS, Manoel Pimentel. JUnit - Implementando testes unitários em Java. 2013. Disponível em: http://www.devmedia.com.br/junit-implementando-testes-unitarios-em-java-parte-i/1432#ixzz30KXKhhi8. Acesso em: 05 jan. de 2014

MYERS, G.; COREY, S. The art of software testing. 2. ed. New Jersey: John Wiley & Sons, 2004.

NOGUEIRA, Elias. 4° Encontro Mensal ALATS: Automação de Teste - Mitos e Verdades, 2009. Disponível em: http://pt.slideshare.net/elias.nogueira/4-encontro-mensal-alats-automao-de-teste-mitos-e-verdades. Acesso em: 18 mar. de 2014

OLIVEIRA, R. DE; GÓIS, F.; FARIAS, P. Automação de Testes Funcionais: Uma Experiência do SERPRO. In: I BRAZILIAN WORKSHOP ON SYSTEMATIC AND AUTOMATED SOFTWARE TESTING, 2007, João Pessoa.

PATUCI, Gabriela De Oliveira. Ferramentas de teste de software. 2013. Disponível em: http://www.devmedia.com.br/ferramentas-de-teste-de-software-revista-engenharia-de-software-magazine-37/21424#. Acesso em: 06 jan. 2014

PEZZÉ Mauro; YOUNG Michal. **Teste e Análise de Software: Processos, Princípios e Técnicas**. São Paulo: Bookman, 2008.

PRESSMAN, Roger S. **Engenharia de Software**, 6ª Edição. Porto Alegre: McGraw-Hill, 2006.

RIOS, Emerson. Por que implantar o MPT.Br. 2013. Disponível em: http://www.emersonrios.eti.br/Artigos/Por%20que%20implantar%20o%20MPT.pdf. Acesso em: 19 fev. 2014.

RUP Rational Software Corporation. Rational Unified Process®, RUP. 2007. Disponível em http://www.wthreex.com/rup/v711_ptbr/index.html/. Acesso em: 07 de dez. 2013.

SELENIUMHQ. Selenium. 2013. Disponível em: http://www.seleniumhq.org/docs/01_introducing_selenium.jsp. Acesso em 25 out. 2013.

SOFTEX RECIFE. Melhoria do Processo de Teste Brasileiro Guia de Referência do Modelo - MPT.Br. Recife, 2011a.

SOFTEX, RECIFE. Fundamentos do Teste de Software. Recife, 2011b. Disponível em: http://ava.nac.softex.br/pluginfile.php/602/mod_resource/content/2/Aula%202.pdf. Acesso em Janeiro de 2014.

SOMMERVILLE, Ian. **Engenharia de Software,** 8ª Edição. São Paulo: Pearson Education do Brasil, 2007.