UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ
BACHARELADO EM ENGENHARIA DE SOFTWARE

WAGNER FRANCALINO SILVA

RELATO DE EXPERIÊNCIA EM AUTOMAÇÃO DE TESTES FUNCIONAIS EM UMA EMPRESA DE TI

QUIXADÁ
2016

WAGNER FRANCALINO SIVA

RELATO DE EXPERIÊNCIA EM AUTOMAÇÃO DE TESTES FUNCIONAIS EM UMA EMPRESA DE TI

Monografia apresentada ao Curso de Engenharia de Software da Universidade Federal do Ceará como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software

Orientadora Prof[a.] Dra. Paulyne Matthews Jucá

QUIXADÁ
2016

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

S578r Silva, Wagner Francalino
Relato de experiência em automação de testes funcionais em uma empresa de TI/ Wagner Francalino Silva. – 2016.
90 f. : il. color., enc. ; 30 cm.

Monografia (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Bacharelado em Engenharia de Software, Quixadá, 2016.
Orientação: Profa. Dra. Paulyne Matthews Jucá
Área de concentração: Computação

1. Software – Testes 2. Software – Desenvolvimento 3. Testes – Automação I. Título.

CDD 005.14

WAGNER FRANCALINO SILVA

RELATO DE EXPERIÊNCIA EM AUTOMAÇÃO DE TESTES FUNCIONAIS EM UMA EMPRESA DE TI

Monografia apresentada ao Curso de Bacharelado em Engenharia de Software da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Bacharel em Engenharia de Software. Área de concentração: computação

Aprovado em: _____ / janeiro / 2016.

BANCA EXAMINADORA

_____

Profª. Dra. Paulyne Matthews Jucá

Universidade Federal do Ceará-UFC

_____

Profª. Dra. Carla Ilane Moreira Bezerra

Universidade Federal do Ceará-UFC

_____

Profª. Dra. Ingrid Teixeira Monteiro

Universidade Federal do Ceará-UFC

A Deus, quando algumas vezes, sentindo-me desacreditado e perdido nos meus objetivos, ideais ou questões pessoais, me assegurou e me fez vivenciar a delícia de me formar.

Aos meus queridos pai e mãe, que me trouxeram com todo o amor e carinho a este mundo, dedicaram, cuidaram e doaram incondicionalmente seu sangue e suor em forma de amor e trabalho por mim, despertando e alimentando em minha personalidade, ainda na infância, a sede pelo conhecimento e a importância deste em minha vida.

Aos amigos (as), familiares, professores (as) e todos aqueles (as) que cruzaram em minha vida, participando de alguma forma na construção e realização deste tão desejado sonho de carregar o canudo de minha formatura.

A minha pequena família Doado Oliveira (futuramente Doaldo Francalino), essa conquista não teria acontecido sem sua força, perseverança e confiança depositada em mim, meu muito obrigado.

AGRADECIMENTOS

Aos meus pais, por dizerem não nos momentos certos.

Ao meu grande e eterno amor Doaldo Oliveira, pois, sem ele, nada disso valeria a pena. A minha colega e irmã de coração Cristina Soriano, meu muito e sincero obrigado. A minha professora e, acima de tudo amiga Paulyne Matthews Jucá.

Aos amigos Expedito Mendonça, Aline Mendes, Talita Vasconcelos, Ygor Rocha, Otávio Augusto, Anderson Uchoa, Priscila Rodrigues, Fábio Maia, Paulo Henrique, Paulo André, André Martins, André Martinz, Antônio Carlos, Geldeson Alves, Sayonara Saraiva, Beatriz Brito, Fernanda Amâncio, Isabelly Damasceno e todos os outros que conviveram comigo todo esse tempo.

E, por último, não menos importante, a Deus por me escolher para trilhar esse momento. Espero não ter decepcionado.

"A persistência é o caminho do êxito." (Charles Chaplin)

RESUMO

O teste de software é muito importante para a garantia da qualidade de um software. No entanto, a atividade de teste não é uma tarefa simples, ela exige um bom planejamento a para execução ser bem-sucedida. Diante deste 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. É nesse contexto que a automação de testes ganha destaque. Baseado neste aspecto, o presente trabalho apresenta um relato de experiência em automação de testes funcionais em uma empresa de TI, tendo como foco um projeto de desenvolvimento de um sistema web, utilizando a ferramenta Selenium para automação dos testes.

Palavras-chave: Testes de Software. Automação de Testes. Selenium.

ABSTRACT

Software testing is very important to guarantee the quality of software. However, the testing activity is not a simple task, it requires good planning to execution to be successful. In this scenario, it is increasingly evident the search of companies by methods and tools that streamline the development process and ensure a higher quality of systems. It is in this context that test automation is highlighted. Based on this aspect, this paper presents an account of experience in automation of functional tests for an IT company, focusing on one of a web system development project, using the Selenium tool for automating tests.

Keywords: Software Testing. Automation Test. Selenium.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Segundo Sommerville (2011), a atividade de teste de software envolve a execução de uma implementação do software com os dados de teste e a análise de suas saídas e de seu comportamento operacional, a fim de verificar se foi executada conforme o esperado.

Teste de software é um assunto que vem ganhando bastante destaque nas empresas e universidades, sendo um alvo de diversos estudos que visam melhorar a qualidade dos sistemas. Bernardo (2011) afirma que as pesquisas na área de teste de software têm crescido de maneira significativa nos últimos tempos, especialmente as pesquisas de automação de teste. Automatizar testes significa fazer uso de software que controle a execução dos casos de teste (TUSCHLING, 2008). O uso desta prática pode reduzir o esforço necessário para realizar os testes em um projeto de software, ou seja, executar maiores quantidades de testes em tempo reduzido. Testes manuais que levariam horas para serem totalmente executados poderiam levar minutos caso fossem automatizados. Porém, nem todos os testes devem ser automatizados e a automação nunca deverá substituir completamente os testes manuais.

Em um cenário real de desenvolvimento, muitas vezes os testadores não têm o tempo necessário para testar completamente o produto dentro dos prazos definidos para a entrega, demandando o aumento de esforço por parte do testador para testar o sistema, a alocação de novos recursos ou a perda da qualidade do software entregue (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).

Durante a construção deste trabalho, foi possível encontrar diversos estudos que propõem a utilização da automação de testes no desenvolvimento de sistema, ressaltando os pontos positivos de se automatizar os testes já executados no processo de desenvolvimento de software. Oliveira, Góis e Farias (2007) descrevem em seu trabalho a experiência prática de se automatizar atividades de teste utilizando um processo de desenvolvimento baseado no método ágil *Scrum*. Lima (2014) apresenta um estudo de caso de automação de testes funcionais em um sistema web, desenvolvido no Laboratório de Sistemas e Bancos de Dados

da Universidade Federal do Ceará. O presente estudo teve como objetivo implantar uma estratégia de automação de testes funcionais em uma empresa de TI, visando solucionar os problemas de grande esforço na execução de testes de regressão e cobertura reduzida dos testes.

O objetivo geral deste trabalho é apresentar um estudo de caso na implantação de estratégia de automação de testes funcionais em uma empresa de TI, impulsionando o aumento da cobertura dos testes e diminuição do tempo de execução dos testes de regressão. Os objetivos específicos relacionados a esse trabalho são: identificar os mecanismos de automação de teste mais utilizados em engenharia de software, definir uma estratégia para automação dos testes para o projeto, introduzir as atividades de automação de teste definidas na estratégia de teste e verificar o impacto da automação dos testes na cobertura dos testes de regressão, a partir de um comparativo com o processo automatizado e o manual.

Os resultados obtidos deste trabalho foram:

- Alteração do processo seguido pela empresa em estudo, na introdução de atividades de desenvolvimento e execução de testes automatizados.
- Redução de tempo na execução de testes a partir da execução automática de *scripts*, estes, por sua vez, gerados automaticamente, que representam testes de regressão.
- Redução direta de custo no desenvolvimento e manutenção de softwares a partir do menor tempo necessário à execução de testes de regressão, dado que estes poderão ser executados de forma automática e não mais totalmente manual.
- Ampliação da cobertura dos testes de regressão, pois além dos testes automatizados, pode-se executar novos testes de forma manual.
- Melhoria no processo de teste executado pela empresa de TI estudada, diminuindo a quantidade de erros e consequentemente aumentando a confiança do software desenvolvido.

O trabalho está organizado da seguinte forma. No capítulo inicial, abordamos uma breve introdução a testes de software. Já no segundo capítulo, são exibidos os trabalhos relacionados. No terceiro capítulo são discutidas as técnicas de testes funcional e estrutural, assim como, estratégia e técnicas de automação dos testes, e um breve relato de benefícios e desafios da automação. A metodologia deste trabalho, baseada na pesquisa-ação, é encontrada no quarto capítulo. No quinto capítulo, temos uma análise do processo adotado pela empresa em estudo, inclusão de atividades de automação no processo de testes manuais, análise e

avaliação de ferramentas de automação, descrição da ferramenta selecionada, montagem do ambiente de testes, descrição e seleção dos critérios de casos de teste, desenvolvimento, execução. No sexto capitulo, é apresentada a análise da automação qualitativa e quantitativa. As conclusões e trabalhos futuros deste trabalho estão no sétimo capitulo.

2 TRABALHOS RELACIONADOS

Lima (2014) apresenta um estudo de caso de 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.

Os resultados obtidos por Lima (2014) foram uma maior agilidade na execução dos testes e um aumento na identificação de falhas que permitiu uma melhoria da qualidade do software desenvolvido.

Um segundo trabalho considerado é um artigo dos autores Oliveira, Góis e Farias (2007), que descreve a experiência prática de se automatizar atividades de teste utilizando um processo de desenvolvimento baseado no método Ágil *Scrum*[1]. A principal diferença deste trabalho é que, nele, não é realizado nenhuma medição indicando que o processo de automação conseguiu atingir os seus objetivos. Sua principal contribuição é a percepção de atividades de automação de testes no processo de desenvolvimento *Scrum*.

São apresentados na tabela 1, os aspectos que diferenciam os trabalhos relacionados a este. Os aspectos marcados com X indicam sua existência nos trabalhos, aqueles que apresentam o traço "-" indica a ausência do aspecto no trabalho. Este trabalho é apresentado na coluna com o título Silva.

Tabela 1 - Aspectos de diferenciação entre os trabalhos.

Aspecto	Lima	Oliveira, Góis e Farias	Silva
Definição de atividades, papéis e artefatos gerados pela implementação dos testes automatizados	X	-	X

[1] SCRUM é uma metodologia ágil para gestão e planejamento de projetos de software. (Disponível em http://www.desenvolvimentoagil.com.br/scrum/ acessado em 15/01/2016)

Utilização do Selenium Web Driver	-	-	X
Definição do ambiente de testes	-	X	X
Estratégia de testes automatizados baseado no *Scrum*.	X	-	X
Definição de critérios de seleção de casos de teste	X	-	X
Análise e avaliação de ferramentas para automação.	-	-	X
Descrição dos resultados quantitativamente da automação dos testes	X	X	X
Descrição dos resultados qualitativamente da automação dos testes	-	-	X
Integração dos resultados dos testes com ferramentas de gerenciamento do software	X	-	-
Estudo detalhado dos impactos das atividades de automação de testes no modelo de desenvolvimento do sistema.	-	X	-
Uso do Selenium como *plugin* do navegador Firefox para criação dos *scripts* de testes.	X	-	-

Fonte -Elaborada pelo autor

No capítulo seguinte, é apresentada a fundamentação teórica deste trabalho.

3 FUNDAMENTAÇÃO TEÓRICA

Na fundamentação teórica desse trabalho, serão abordados os conceitos utilizados em seu desenvolvimento. Na primeira seção, são apresentados os conceitos de teste de software com a definição das principais abordagens e, por último, é apresentada a definição de automação de testes e seus principais conceitos e técnicas.

3.1 Teste de Software

"As disciplinas de engenharia de software alinham atividades de projeto e construção com atividades que verificam produtos intermediários e finais de forma que os defeitos possam ser identificados e removidos" (PEZZÈ, 2008, p. 25). A atividade de teste é responsável por investigar o software a fim de fornecer informações sobre sua qualidade e presença de falhas ao contexto em que o software deve operar.

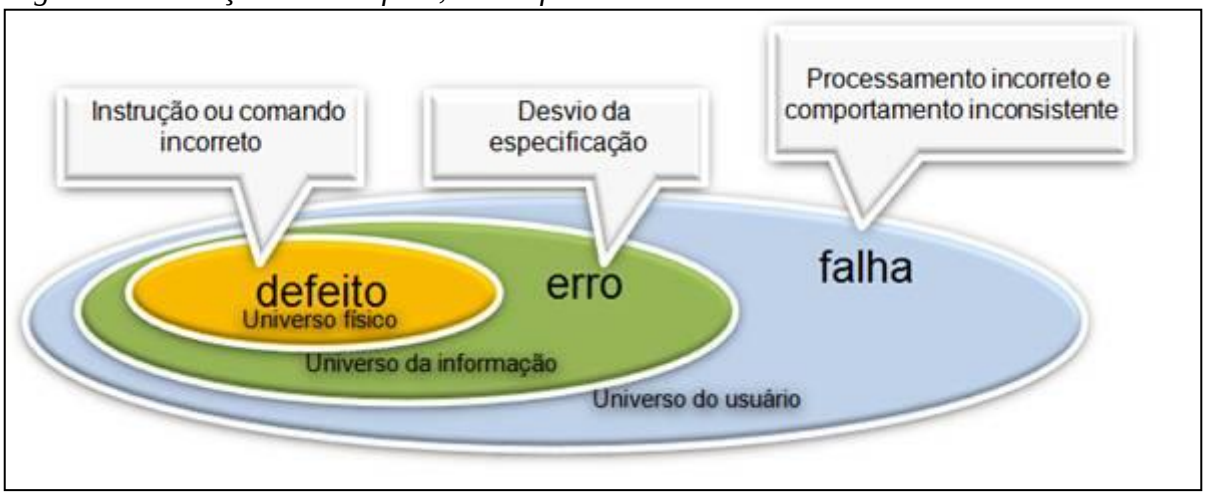
Essa atividade de teste, segundo Rios (2013), é um processo de execução de software ou parte dele para determinar se ele atingiu as especificações estabelecidas. O seu objetivo é revelar o máximo de presença de falhas no software, de modo que, depois de identificadas, as falhas sejam removidas.

Para melhorar o entendimento da atividade de teste de software, é necessário possuir um breve conhecimento sobre alguns conceitos, dentre eles: erro, defeito e falha. Estes conceitos são descritos a seguir e representados pela Figura 1, de acordo com a definição do Institute of Electrical and Electronics Engeneers (IEEE 610, 1990):

- Defeito é um ato inconsistente cometido por um indivíduo ao tentar entender uma determinada informação, resolver um problema ou utilizar um método ou uma ferramenta.
- Erro é uma manifestação concreta de um defeito em um artefato de software. Diferença entre o valor obtido e o valor esperado, ou seja, qualquer estado intermediário incorreto ou resultado inesperado na execução de um programa constitui um erro.

- Falha é o comportamento operacional do software diferente do esperado pelo usuário. Uma falha pode ter sido causada por diversos erros e alguns erros podem nunca causar falhas.

Figura 1 - Distinção entre defeito, erro e falha



Fonte: IEEE 610, 1990

O foco deste trabalho é em falhas, já que estas são detectadas através do comportamento do sistema. Desvios na especificação (erros) e erros no código (defeitos) não fazem parte do escopo desse trabalho.

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

> [...] teste de software é um processo que visa a execução de forma controlada, com objetivo de avaliar o seu comportamento baseado no que foi especificado. A execução dos testes é considerada um tipo de validação. (RIOS; MOREIRA FILHO, 2013, p. 10)
>
> [...] O teste proporciona o último elemento a partir do qual a qualidade pode ser estimada e, mais pragmaticamente, os erros podem ser descobertos [...] (PRESSMAN, 2011, p. 290)

A definição de teste de software descrita por Pressman (2011) será adotada por este trabalho, por descrever a atividade de teste de software com um objetivo claro.

Sommerville (2011) define quatro atividades básicas para um processo de construção de software: especificação, desenvolvimento, validação e evolução. A validação de software abrange um conjunto de técnicas utilizadas para investigar falhas em um sistema. Para cada etapa do desenvolvimento, diferentes tipos de testes devem ser executados para

permitir que novas funcionalidades sejam criadas e adicionadas sem adição de defeitos ao sistema.

As atividades de validação, também denominadas de verificação e validação (V&V) referem-se ao conjunto de atividades que garantem que o software implementa corretamente uma função específica (PRESSMAN, 2011). Validação se refere a um conjunto de atividades que garantem que o software construído corresponde aos requisitos do cliente. Seu objetivo é assegurar que o software esteja adequado às necessidades do usuário, ou seja, confirmar que o software esteja dentro das especificações. As atividades de V&V possuem elementos essenciais para sua formalização, alguns são essenciais para o desenvolvimento deste trabalho, os quais são descritos a baixo:

- Caso de teste é um conjunto de condições usadas para testar o software. Ele pode ser elaborado com o objetivo de identificar falhas no funcionamento do software, ou ainda, verificar se o software que foi construído possui seus requisitos plenamente atendidos. Craig e Jaskiel (2002) descrevem casos de teste como uma condição particular a ser testada e são compostos por valores de entrada, restrições para sua execução e um resultado ou comportamento esperado. Os casos de teste são usados neste trabalho como arcabouços para a construção dos testes automatizados.
- Procedimento de teste, segundo Craig e Jaskiel (2002), é uma descrição dos passos utilizados para a execução de um ou mais casos de teste. Neste trabalho, os procedimentos de teste são extraídos dos casos de teste, eles servem como base para construção dos testes automatizados.

Myers (2004) defende a ideia de que há princípios vitais para o teste de software. Um deles são os casos de teste, pois eles devem definir a saída esperada, de forma a reduzir a interpretação do critério de sucesso. A saída da execução do teste deve ser analisada. Outro princípio também relacionado aos casos de teste indica que se deve verificar não somente as condições inválidas de execução, como também as condições válidas.

Além dos elementos essenciais para formalizar as atividades de V&V, existem os níveis de testes. Segundo Rocha et al. (2001), o planejamento dos testes deve ocorrer em diferentes níveis e em paralelo ao desenvolvimento do software. Os principais níveis de teste de software são:

a. **Teste de Unidade:** segundo Bernardo (2011), 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 às funções para linguagens procedurais e métodos em linguagens orientadas a objetos.

O teste unitário tem como objetivo garantir que o código escrito esteja de acordo com as especificações antes de integrá-lo com outras unidades. Ele é importante para garantir que todo o código escrito para a unidade possa ser executado. Nesse nível de teste, é imprescindível o acesso ao código fonte e isso acontece durante a fase de implementação.

b. **Testes de Integração:** Somerville (2011) define que a fase de teste de integração tem como objetivo encontrar falhas provenientes da integração interna dos componentes de um sistema.

O teste de integração visa encontrar falhas associadas a interfaces entre os módulos quando esses são integrados para construir a estrutura do software que foi estabelecida na fase de projeto.

c. **Teste de Sistema:** o teste de sistema é o tipo de teste executado para garantir que o sistema funcione como um todo, ou seja, valida que a sua integração com outras interfaces está funcionando conforme o esperado.

O sistema já completamente integrado é verificado quanto a seus requisitos em um ambiente de produção. Está no escopo da técnica de teste de caixa-preta e dessa forma não requer conhecimento da estrutura (lógica) interna do sistema. É um teste mais limitado em relação aos testes de unidade e de integração, fases anteriores do processo de teste, pois se preocupa somente com aspectos gerais do sistema.

Segundo Softex (2011), 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.

d. **Teste de Aceitação:** segundo Softex (2011), o teste de aceitação é aquele realizado para assegurar ao usuário que o sistema irá funcionar de acordo com as suas expectativas.

Os testes de aceitação são realizados, geralmente, por um grupo pequeno de usuários do sistema, que simulam operações de rotina do sistema de modo a verificar se seu comportamento está de acordo com os requisitos, para permitir ao cliente determinar se aceita ou não o sistema.

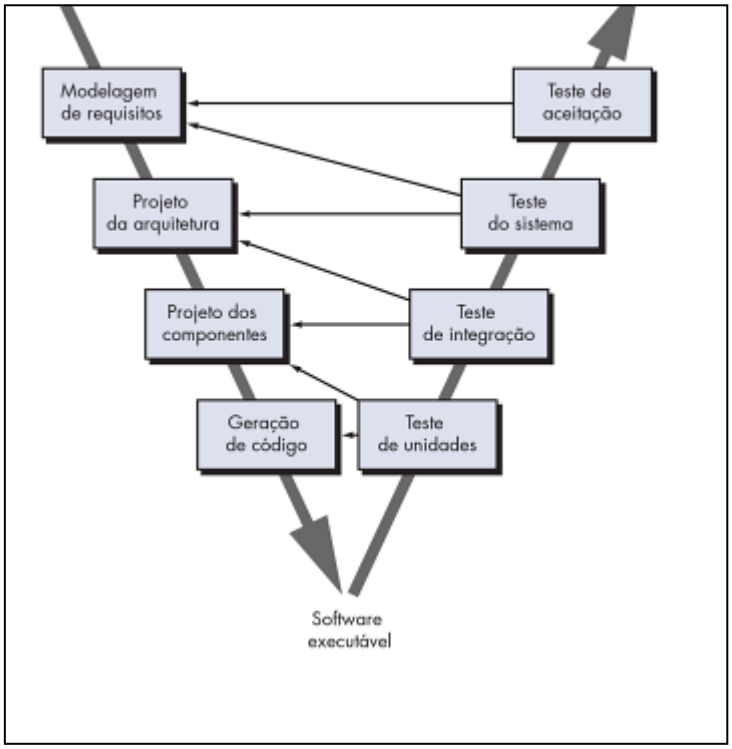
e. **Teste de Regressão:** Segundo Chittimalli, Harrold e Mary (2008), no desenvolvimento de um software, são realizadas muitas modificações por

diversas razões, como correção de erros, inclusão de novos requisitos, mudança de ambiente e melhoria de desempenho. Após as mudanças terem sido realizadas, o sistema precisa ser testado para garantir que ele está se comportando como pretendido e que as modificações não causaram nenhum impacto adverso na qualidade do software. Estes testes realizados após as mudanças são chamados de Testes de Regressão.

Para H. K. N (1989) e Beizer (1990), os testes de regressão são essenciais para garantir a qualidade e a confiabilidade de um sistema. Eles são caros, porém, quando executados, podem reduzir à metade o custo da manutenção do software. Agrawal, et al (1993) completa o pensamento, destacando que os testadores dispõem de pouco tempo para executar os testes de regressão devido à pressão para liberação de uma versão do sistema. Sendo assim, um teste de regressão completo não pode ser sempre realizado durante modificações e atualizações frequentes de um sistema, por exigir uma grande quantidade de esforço e por ocupar uma fração de tempo significante no ciclo de vida de um software.

A Figura 2 mostra como as atividades de planejamento de testes em diferentes níveis devem ocorrem em paralelo ao desenvolvimento do software.

Figura 2 - Modelo de Processo em V



Fonte: Pressman (2011, p. 60)

A conexão entre os lados esquerdo e direito do modelo em V implica que, quando encontradas falhas durante a verificação e a validação, o lado esquerdo do V pode ser executado novamente para corrigir e melhorar os requisitos, o projeto e a codificação, antes da execução das etapas de testes que estão no lado direito.

> [...] O modelo em V descreve a relação entre a garantia da qualidade e as ações associadas à comunicação, modelagem e atividades de construção iniciais. A medida que a equipe de software desce em direção ao lado esquerdo do V, os requisitos básicos do problema são refinados em representações progressivamente cada vez mais detalhadas e técnicas do problema e de solução. Uma vez que o código tenha sido gerado, a equipe se desloca para cima, ao lado direito do V, realizando basicamente uma série de testes (ações da garantia da qualidade) que validem cada um dos modelos criados à medida que a equipe se desloca para baixo, no lado esquerdo do V [...] (PRESSMAN, 2011, p. 60).

Para o desenvolvimento deste trabalho, serão considerados os conceitos de testes de sistema e regressão, falha, casos e procedimentos descritos acima, pois para atingirmos nosso objetivo de implantar uma estratégia de automação de testes funcionais em uma empresa de TI, visando solucionar os problemas de grande esforço na execução de testes de regressão e cobertura reduzida dos testes, esses conceitos são importantes para um bom entendimento.

3.2 Técnicas de Teste

Existem na literatura muitas maneiras descritas de como se testar software. Dentre essas inúmeras maneiras, existem as técnicas mais citadas para o teste em sistemas. Para esse trabalho, iremos citar as técnicas de teste funcional ou também chamado de teste de caixa preta, as técnicas de teste estrutural, comumente chamado de teste de caixa branca. A aplicação de uma das técnicas não exclui a aplicação da outra, pois, segundo Furlan (2009), existem algumas situações em que os testes funcionais não são capazes de detectar defeitos, pois eles preocupam-se somente em comparar se as saídas pelo sistema estão de acordo com o esperado, não levando em consideração a funcionalidade interna do software. Desta forma, podem existir procedimentos internos que não afetam as respectivas saídas e que não serão detectadas se for aplicada apenas o teste funcional.

3.2.1 Teste Estrutural (Teste de Caixa Branca)

Segundo Molinari (2005), os testes do tipo estrutural têm como objetivo garantir que todas as linhas de código e condições sejam executadas pelo menos uma vez e estejam corretas. Essa técnica trabalha diretamente com foco no código-fonte do componente de software para avaliar aspectos como: teste de condição, teste de fluxo de dados, teste de ciclos e teste de caminhos lógicos (PRESSMAN, 2005).

O teste de caixa branca estabelece os requisitos de teste com base na implementação, verificando se atende aos detalhes do código e solicitando a execução de partes ou de componentes elementares do programa. Os critérios pertencentes a essa técnica são classificados com base no fluxo de controle, o fluxo de dados e na complexidade.

De acordo com Inthurn (2001), o teste estrutural tem como objetivo verificar se a estrutura interna da unidade está correta. Esta verificação é efetuada através de casos de teste que visam percorrer todos os caminhos internos possíveis da unidade.

O teste de caixa branca divide-se em vários testes, onde podemos citar:

- O teste de condição verifica as condições lógicas contidas em um módulo do programa. Segundo Pezzè (2008), a cobertura de condições é útil para exercitar falhas na maneira como uma computação é decomposta em casos. A cobertura de condição considera essa decomposição em mais detalhes, forçando a exploração não apenas de ambos os resultados possíveis de uma expressão booleana controlando um desvio, mas também das diferentes combinações das condições individuais de uma expressão booleana composta.
- O teste de fluxo de dados exercita a lógica do programa. Essa técnica seleciona caminhos de teste de acordo com as localizações das definições e usos de variáveis no sistema.
- O teste de ciclos é uma técnica de teste de caixa branca que focaliza exclusivamente a validade de construções de ciclos. Quatro diferentes classes de ciclos podem ser definidas: ciclos simples, concatenados, aninhados e desestruturados.

Segundo Bartié (2002), a principal vantagem do teste estrutural é a alta eficiência na detecção de erros. Ele destaca que uma das desvantagens desta técnica é que ela é difícil de implementar e que demanda uma equipe especializada para sua realização, caso contrário os resultados podem ser demorados e não significativos.

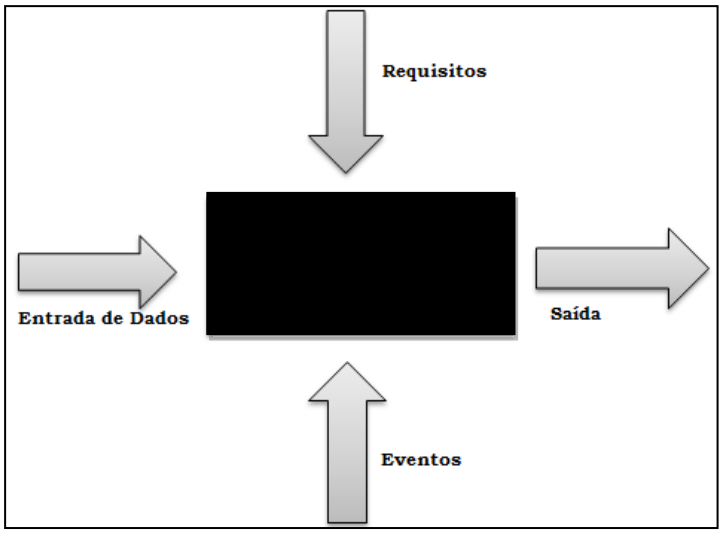
O teste de caixa branca não foi escolhido por necessitar de acesso ao código fonte, causando uma demanda de uma equipe técnica para a realização dos testes e pela dificuldade de sua implementação. Esse tipo de teste é projetado em função da estrutura do componente e permite uma averiguação mais precisa do comportamento dessa estrutura. O acesso ao código facilita o isolamento de uma função ou ação, o que ajuda na sua análise comportamental, mas não é o foco deste trabalho.

3.2.2 Teste Funcional (Teste de Caixa Preta)

Segundo Molinari (2005), o teste de caixa preta tem como objetivo garantir que todos os requisitos ou comportamentos da aplicação ou de um componente estejam corretos.

Teste de caixa preta é a técnica de teste em que o componente de software a ser testado é abordado como se fosse uma caixa-preta, ou seja, não se considera o seu comportamento interno (PRESSMAN, 2011). Dados de entrada são fornecidos, o teste é executado e o resultado é obtido e comparado a um resultado esperado previamente conhecido. Haverá sucesso no teste se o resultado obtido for igual ao resultado esperado. O componente de software a ser testado pode ser um método, uma função interna, um programa, um conjunto de programas e/ou componentes ou mesmo uma funcionalidade. A técnica de teste funcional é aplicável a todos os níveis de teste. A visão do teste de caixa preta pode ser vista na Figura 3.

Figura 3 – Teste Funcional



Fonte: Molinari (2005), tradução nossa

O teste funcional não se preocupa em verificar como ocorreu internamente os processamentos no software, mas se o algoritmo inserido no software produz os resultados esperados (BARTIÉ, 2002).

Ainda segundo Bartié (2002), o profissional responsável pela realização desta técnica deve possuir pleno conhecimento dos requisitos, suas características e comportamento esperado do sistema, para que seja possível realizar uma avaliação do software através dos resultados obtidos pela execução do software.

Segundo Myers (1979), o uso da técnica de teste funcional reduz o número de casos de testes necessários para se executar um teste confiável e produz casos de testes que indicam a presença ou ausência de classes de erros e não apenas erros associados ao teste em questão.

Alguns dos principais critérios associados a essa técnica são apresentados por Pressman (2001):

- **Particionamento de Equivalência**: esta técnica divide o domínio dos dados de entrada em classes ou partições de equivalência. O objetivo dessa divisão é que cada classe ou repartição represente uma possível falha para ser encontrada (Bartié, 2002).

  O interessante desta técnica é que se um elemento da classe de equivalência produz uma falha no sistema, todos os outros elementos da mesma classe produzem a mesma falha. Como os testes de software, na maioria das vezes, são exaustivos, essa técnica prevê a seleção de classes de acordo com os valores de entrada. Essa técnica deve garantir que, em cada classe identificada, maximize-se a quantidade dos valores de entradas quando possível e minimize os casos de testes necessários. Além de garantir que a quantidade dos valores de entrada, quando divididos em classes, seja significativa de tal maneira que, com alguma certeza, é garantido que se um elemento da classe detecta uma falha, qualquer outro elemento da classe pode detectar a mesma falha.
- **Análise do Valor Limite:** o objetivo desta técnica é verificar as condições que ficam extremamente perto dos valores limites do domínio de entrada. Segundo Burnestein (2003), após inúmeros testes realizados, os testadores perceberam que muitas falhas ocorrem nos limites das classes de equivalência, de tal modo que a análise do valor limite complementa a técnica de particionamento de equivalência. Ela exige que sejam selecionados elementos próximos às bordas,

de modo que os casos de testes cubram os valores limites superiores e inferiores.

Uma das diferenças entre as técnicas de análise do valor limite e particionamento de equivalência é que, ao invés de selecionar qualquer elemento pertencente a uma classe para realizar o teste, a técnica de análise do valor limite realiza teste em um ou mais elementos selecionados, de tal modo que os valores que estão mais próximos dos limites desta classe sejam os objetos de teste.

Para os testes realizado por este trabalho foi, utilizada majoritariamente a técnica de testes funcionais, particionamento de equivalência e análise do valor limite.

A seguir, são apresentados conceitos de automação de testes, seus principais benefícios e desafios.

3.3 Automação de Testes

Segundo Fewster e Graham (1999), um software deve ser testado para que haja uma validação de que ele funciona como deveria funcionar. Esse teste tem que ser eficaz na detecção de falhas, eficiente, executado rapidamente e mais barato quanto possível.

Ainda para os autores, automação dos testes significa redução do esforço na realização dos testes e aumento do número de testes que podem ser feitos em um curto período de tempo ou com um tempo determinado.

Nas próximas seções, serão discutidos os principais benefícios e dificuldades encontrados pela implementação de testes automatizados, as principais técnicas existentes na literatura e aspectos importantes sobre a implantação de estratégia de automação de testes em uma organização.

3.3.1 Benefícios da Automação dos Testes

Fewster e Graham (1999) destacam que a automação dos testes pode permitir que algumas tarefas ligadas a teste de software possam ser executadas de maneira mais eficiente do que se estivessem sendo feitas manualmente. Para os autores, os benefícios da automação são:

- Execução de testes de regressão em novas versões do sistema: esta é talvez a tarefa mais beneficiada pela automação, pois em um ambiente onde são realizadas muitas modificações no sistema, realizar testes manuais a cada alteração pode ser considerado uma atividade cansativa e com um longo

período de tempo dedicado. Quando os testes automatizados são implementados, o esforço envolvido para executar um conjunto de testes de regressão deve ser mínimo, dado que os testes existentes já foram automatizados. Neste caso, deve ser possível selecionar um conjunto de teste e iniciar sua execução com menor esforço manual comparado à atividade de testes manuais.

- Execução de mais testes com frequentemente: um dos benefícios mais claros da automação é a possibilidade de execução de mais testes em menos tempo, levando em consideração que os testes automatizados são realizados em menor tempo que os testes manuais, consequentemente torna-se possível a execução dos testes automatizados com maior frequência. Essa possibilidade de execução de testes mais vezes torna o software de maior confiança.
- Execução de testes difíceis de se realizar manualmente: a automação de testes possibilita a execução de testes de performance em larga escala em um menor tempo de execução comparado com o manual, além de testes que envolvem cálculos complexos e acessos de usuários múltiplos ao sistema. Ambos os testes podem ser difíceis de serem realizados por humanos, porém a ferramenta de automação pode substituir a execução do teste manual parcialmente.
- Otimização dos recursos: automação das tarefas repetitivas, gerando uma maior exatidão nos testes, além de melhorar o ânimo da equipe de testes. A automação possibilita mais tempo livre para os testadores dedicarem seus esforços em atividades de testes mais detalhadas e caras para serem automatizadas, aumentado assim a cobertura dos testes.
- Consistência dos testes repetitivos: a exatidão da execução de um teste automatizado é a mesma toda vez que esse é executado, essa exatidão dá um nível de consistência aos testes que seria difícil de se conseguir manualmente.
- Lançamento adiantado no mercado: como os testes automatizados podem ser executados mais vezes e mais frequentemente, o tempo de teste decorrido pode ser diminuído. Assim o tempo de lançamento do produto no mercado pode ser diminuído.
- Nível de confiança no produto aumentada: como a equipe de testes sabe da consistência e exatidão dos testes executados, bem como, as suas sucessivas execuções bem-sucedidas, pode haver uma confiança maior no produto de que não haverá o encontro de falhas pelo uso do produto no ambiente externo.

E ainda podemos adicionar um maior ganho no tempo para execução de testes exploratórios, já que os testes automatizados são executados.

A seguir, serão apresentados os desafios da automação de teste.

3.3.2 Desafios da Automação de Teste

Muitas empresas de TI realizam seus testes de forma manual. Essa técnica, em geral, não é a melhor forma de se realizar testes, pois ele demanda de muito tempo para sua realização e pode estar sujeita a falhas humanas. O ideal seria utilizar processos automatizados, mas a decisão de automatizar os testes deve ser avaliada com muita prudência, pois não existe solução definitiva e de fácil implementação. Segundo Molinari (2010), os principais desafios para automação de testes são:

- Expectativas irreais: muitas vezes a equipe de desenvolvimento de software, deixa se seduzir pelas promessas dos fabricantes de software de automação, diante das necessidades de resolver problemas imediatos ligados a teste de software. Muitas vezes, a solução encontrada vai de encontro ao o principal aspecto da automação: Não existe uma única solução através da qual tudo é possível de se resolver. Quando as equipes de desenvolvimento de software começam a automatizar seus testes, percebem que nem sempre automatizar é tão fácil assim, mas também não é algo intangível.
- Obtenção de suporte para o gerenciamento de ferramentas de automação de teste: quando a seleção das ferramentas para automação for realizada, deve-se levar em consideração a existência de suporte, seja pela definição de boas práticas durante seu uso ou pela definição de padrões de uso, além da existência de comunidades de discussão sobre a utilização da ferramenta. Isso é um critério importante para que haja transparência no processo de automação de testes, assim como no uso da ferramenta.
- Automação de testes é um novo projeto: muitas vezes as equipes de desenvolvimento de software não se dão conta de que a automação de testes é um novo projeto, pois na verdade a automação é um projeto derivado de um projeto de teste. Para a realização do novo projeto, deve-se ter na empresa uma equipe voltada para automação: suporte, criação de *scripts* e pesquisa de novas estratégias de automação. Sempre será necessário deixar algum tempo para a realização de novas pesquisas que enforquem novas estratégias e técnicas de automação.

- Envolver especialistas de testes experientes no processo de avaliação e uso de ferramentas de automação: é de suma importância envolver profissionais com vasta vivência em testes, esta é uma questão técnica que deve ser tratada como tal.

Durante a realização deste trabalho, os desafios destacados acima foram tratados da seguinte maneira:

- Expectativas irreais: No início da implantação da estratégia de automação de testes, foram exibido aos membros da equipe os critérios de automação dos casos de testes, assim como as atividades que fazem parte da estratégia e que iriam compor o processo adotado pela empresa em estudo, focando nas possibilidades reais de automação, deixando claro à equipe que a automação de alguns casos de testes não descartaria os testes manuais, além de não solucionar todos os problemas de desenvolvimento do software em construção.
- Suporte à ferramenta: Este desafio foi adicionado a um conjunto de critérios para a escolha da ferramenta de automação.
- Automação como um projeto: Devido a limitações da equipe de desenvolvimento e testes, uma única pessoa ficou responsável pela implantação e execução dos testes automatizados, assim como os manuais. Sendo de sua responsabilidade a organização do seu tempo para construção e manutenção dos casos de testes automatizados.
- Especialistas para os testes: A pessoa envolvida na construção dos casos de testes automatizados buscou aperfeiçoamento no assunto, com a participação de fóruns de discussão sobre automação, assim como, capacitação em cursos específicos de automação de testes.

Ainda segundo Molinari (2010), o teste automatizado tem uma produtividade maior que os testes manuais e consegue atingir em menor tempo aquilo que em geral é atingido pelos testes manuais. Mesmo assim, não se deve desprezar os testes manuais, mas sim reduzi-los.

3.4 Processo de Automação de Testes

Segundo Pressman (2011), o processo de software pode ser facilmente caracterizado por uma forma de metodologia para atividades, ações e tarefas necessárias para o desenvolvimento de software. Modelar esse processo e atividades que cumpram as

necessidades das organizações é apresentado como um desafio para as empresas (MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO, 2013). Lima (2014) afirma que é possível encontrar na literatura modelos de processos de automação de teste que propõem atividades necessárias à implantação de testes automatizados pelas organizações. Cabe deixar claro que, assim como Lima (2014), este trabalho usa o temo "processo" com o significado de uma estratégia de atividades necessárias para a implantação de testes automatizados.

Para a realização deste trabalho, foi escolhido o Modelo de Melhoria do Processo de Teste Brasileiro (MPT.Br), pois este é um modelo conhecido nacionalmente e possui base em normas reconhecidas internacionalmente, como o IEEE 829:2008 e a ISO/IEC 29119. Nele, consta uma área referente à automação de testes, a Automação da Execução de Testes (AET) utilizada neste trabalho. Não foi utilizado o MPT.Br por inteiro, pois o escopo deste trabalho só engloba a elaboração e adoção de uma estratégia de automação de testes divididas em atividades, artefatos e ações. Este não trata da implantação de um processo completo, nem de modificação do processo existente em um sentido mais geral, mas trata do estudo sobre os benefícios de adotar testes automatizados em comparação com manuais.

A AET possui um único objetivo, que é estabelecer e manter uma estratégia para automação e execução de testes, tendo como base a execução de um conjunto de práticas a seguir apresentadas:

- Definir objetivo do regime de automação: "*a organização deve definir objetivos para a automação da execução dos testes. Esses objetivos devem ser mantidos e avaliados periodicamente para que exista garantia que eles estão sendo alcançados*" (SOFTEX, 20011, p. 72).
- Definir critérios para seleção de casos de teste para automação: "*é necessário um procedimento de definição de critérios para a escolha de casos de testes para automação. Os critérios devem facilitar a análise de custo versus benefícios da automação de casos de teste*" (SOFTEX, 2011, p. 72).
- Definir um *framework* para automação de teste: "*é necessário que a organização estabeleça um framework para auxiliar nas tarefas de automação da execução do teste*" (SOFTEX, 2011, p. 72).
- Gerenciar incidentes de testes automatizados: "*é necessário que a organização defina um procedimento para identificação, análise e gerenciamento dos incidentes ocorridos na execução de testes automatizada. Os incidentes devem*

ser documentados e analisados, pois podem representar um defeito no software" (SOFTEX, 2011, p. 72).

- Verificar aderência aos objetivos de automação: "*a organização deve possuir um procedimento para monitorar e verificar se os objetivos da automação estão sendo atingidos. As não conformidades encontradas devem ser tratadas com ações corretivas*" (SOFTEX, 2011, p. 72).
- Analisar retorno sobre investimento na automação: "*é necessário que a organização verifique, periodicamente, se o investimento na automação de testes teve o retorno esperado. A forma da análise deve ser definida, podendo conter dados de ganhos da automação e despesas associadas à automação*" (SOFTEX, 2011, p. 72).

As práticas descritas acima, para esse trabalho, servirão como arcabouço para a construção da estratégia de automação de testes. Vale destacar que a organização alvo deste estudo não possuía nenhuma atividade de automação de testes que servisse de alicerce para essa pesquisa.

A prática "definir objetivos para automação" pode ser melhor analisada no final do capítulo introdutório como os objetivos alcançados por este trabalho. Já as práticas de "definição dos critérios se seleção dos casos de teste" e "definição do *framework"* são detalhadas nas seções 5.4 e 5.3.2 respectivamente.

A prática de "verificação de aderência aos objetivos de automação" serviu como base para a construção da estratégia de automatizar os testes sugeridos por este trabalho. "Gerenciar incidentes de testes automatizados" tornou-se uma atividade pertencente à estratégia sugerida por este trabalho para automação dos testes e a última prática proposta pelo MPT.Br, que é "analise do retorno sobre o investimento" pode ser melhor analisada na seção final deste trabalho.

A seguir são apresentadas técnicas de automação de testes.

3.5 Técnicas de Automação de Teste

Segundo Fantinato et al. (2004), as principais técnicas de automação de testes apresentadas na literatura são: *record and playback, scripts, data-driven* e *keyword-drivers*. Elas são apresentadas a seguir:

A técnica *record and playback* realiza os testes automatizados por meio da interface gráfica pela qual o usuário interage. Normalmente a ferramenta que se adequa a essa técnica fornece um recurso para gravar as ações do usuário durante o uso do sistema. Essas

ações posteriormente são traduzidas para *scripts* escritos na linguagem suportada pela ferramenta de automação para posteriormente ser executadas.

Para Fantinato et al. (2014), a técnica consiste em utilizar uma ferramenta de automação de testes para gravar as ações executadas pelo usuário, que interage com a interface gráfica do sistema e converte as ações em *scripts* de teste, os quais podem ser executados quantas vezes forem necessárias. Essa técnica é considerada simples e prática. Entretanto, apresenta algumas desvantagens para cenários com grandes conjuntos de casos de teste automatizados, tais como: alto custo, dificuldade de manutenção, baixa taxa de reutilização, curto tempo de vida e alta sensibilidade a mudanças no software a ser testado.

A escrita de *scripts* é outra técnica que consiste em um refinamento da técnica de *record and playback.* Ela se diferencia por utilizar um mecanismo para auxiliar a execução de testes que realizam as mesmas ações repetidamente, porém com dados diferentes. Através da programação, os *scripts* de teste gravados são alterados para que desempenhem um comportamento diferente do *script* original durante sua execução. Para que essa técnica seja utilizada, é necessário que a ferramenta de gravação de *scripts* de teste possibilite a edição dos mesmos. Dessa forma, os *scripts* de teste alterados podem contemplar uma maior quantidade de verificações de resultados esperados, que vão além das ações gravadas pelo testador (FANTINATO et al., 2014).

A técnica *data-driven* consiste em extrair dos *scripts* de teste os dados, que são específicos por caso de teste, e armazená-los em arquivos separados dos *scripts* de testes. Os *scripts* passam a armazenar os procedimentos de teste (lógica de execução) e as ações de teste sobre a aplicação, que normalmente são genéricos para um conjunto de casos de teste. A principal vantagem da técnica é a facilidade de adicionar, modificar ou remover dados de teste, ou até mesmo casos de teste inteiros, com pequena manutenção dos *scripts* (FANTINATO et al., 2014).

Segundo os estudos realizados durante a criação deste trabalho, podemos dizer que a técnica *keyword-drivers* foi criada para dar suporte aos testes de aceitação. Nesta abordagem os testes automatizados são realizados por meio da interface gráfica do sistema. Ainda Fantinato et al. (2014) relatam que os *scripts* de teste passam a conter apenas as ações específicas de teste sobre a aplicação, as quais são identificadas por palavras-chave. Essas ações são como funções de um programa, podendo inclusive receber parâmetros, que são ativadas pelas palavras-chave a partir da execução de diferentes casos de testes. O procedimento de teste é armazenado em um arquivo separado, na forma de um conjunto ordenado de palavras-chave e respectivos parâmetros. A principal vantagem da técnica é a

facilidade de adicionar, modificar ou remover passos de execução no procedimento de teste com necessidade mínima de manutenção dos *scripts* de testes.

Ainda segundo Fantinato et al. (2014), as ferramentas que possibilitam a automação dos testes, geralmente utilizam mais de uma das técnicas citadas acima. Durante esse trabalho, na justificativa da escolha da ferramenta para automação, citaremos a técnica, ou conjunto de técnicas utilizadas por cada ferramenta para a realização dos testes.

4 PROBLEMA E METODOLOGIA

A seguir, são apresentados na seção 4.1 aspectos da pesquisa-ação e na seção seguintes 4.2 a metodologia deste trabalho.

4.1 Pesquisa-Ação

Para Thiollent (1986) o método de Pesquisa-ação é um tipo de pesquisa social com base empírica que é concebida e realizada em estreita associação com uma ação ou com a resolução de um problema coletivo, no qual os pesquisadores e os participantes representativos da situação ou problema estão envolvidos de modo cooperativo ou participativo.

É importante que se reconheça a pesquisa-ação como um dos inúmeros tipos de investigação-ação, que é um termo genérico para qualquer processo que siga um ciclo no qual se aprimora a prática pela oscilação sistemática entre agir no campo da prática e investigar a respeito dela. Planeja-se, implementa-se, descreve-se e avalia-se uma mudança para a melhora de sua prática, aprendendo mais, no decorrer do processo, tanto a respeito da prática quanto da própria investigação.

Embasado nos estudos do Thiollent (1986), a pesquisa-ação possui as seguintes etapas: pesquisa exploratória, formulação do problema, discussão com o grupo de trabalho, coleta dos dados e análise e interpretação dos dados. A seguir essas são melhor apresentadas e enquadradas neste estudo.

4.1.1 Pesquisa exploratória

Nessa fase, foi realizado o estudo do ambiente da empresa em estudo, foi analisado o processo de testes adotado pela empresa em um período de 15 dias. Houve também discussões com os membros do projeto para coletar informações de visões diferentes sobre as atividades de testes seguidas pelo projeto. Na seção 5.1, são apresentados os resultados da análise do processo de testes adotado pela empresa.

4.1.2 Formulação do problema

Após a fase exploratória, foi possível obter informações importantes para a definição do problema. O problema levantado por esse trabalho é: Como implantar uma estratégia de automação de testes funcionais em uma empresa de TI, visando solucionar os problemas de grande esforço na execução de testes de regressão e cobertura reduzida dos

testes? Para solucionar esses problemas identificados, os seguintes pontos precisam ser tratados: diminuição do esforço na execução de testes de regressão e aumento da cobertura dos testes realizados. Assim, esse trabalho propõe a definição de um conjunto de atividades de automação de testes (bem como a definição de papéis, ferramentas e técnicas) a serem definidas e adicionadas ao processo utilizado pela empresa, para permitir a resolução do problema identificado na empresa, considerado por este trabalho como uma estratégia de automação de testes.

4.1.3 Discussão com o grupo de trabalho

Nessa etapa, foi realizada uma discussão com o gerente do projeto, o líder técnico e a equipe de desenvolvimento, com o objetivo de recolher as propostas e contribuições para compor a solução do problema.

4.1.4 Coleta dos Dados

Diversas técnicas são adotadas para a coleta de dados na pesquisa-ação. No entanto, as mais utilizadas nesse trabalho foram a entrevista coletiva e individual com as pessoas envolvidas no problema e a observação do pesquisador. Estas entrevistas serviram para o pesquisador se basear nos impactos que o processo sugerido poderia causar nas atividades já realizadas pela equipe, bem como, conhecer melhor o processo adotado pela empresa. Também foi realizada uma pesquisa de documentos e diretrizes estabelecidas para a atividade dentro da empresa. Em paralelo foram construídos todos os *scripts* de testes automatizados.

4.1.5 Análise e interpretação dos dados

Essa pesquisa privilegiou a discussão em torno dos resultados obtidos e é melhor apresentada na seção 6.

4.2 Metodologia

Controlar a qualidade de sistemas de software é um grande desafio devido à alta complexidade dos produtos e às inúmeras dificuldades relacionadas ao processo de desenvolvimento, que envolve questões humanas, técnicas, burocráticas, de negócios e políticas.

Idealmente, os sistemas de software devem não apenas fazer corretamente o que o cliente precisa, mas também fazê-lo de forma segura, eficiente e escalável. Ele deve ainda ser flexível e de fácil manutenção e evolução. Esse trabalho de pesquisa parte dos seguintes problemas: Grande esforço na execução de testes de regressão e cobertura reduzida dos testes. Para resolver tais problemas o presente trabalho teve como principal objetivo implantar uma estratégia de automação de testes funcionais em uma empresa de TI, visando solucionar tais problemas. Os resultados específicos relacionados a esse trabalho são: identificar os mecanismos de automação de teste mais utilizados em engenharia de software, definir uma estratégia para automação dos testes para o projeto, introduzir as atividades de automação de teste definidas na estratégia de teste e verificar o impacto da automação dos testes na cobertura dos testes de regressão, a partir de um comparativo com o processo automatizado e o manual.

Todas as atividades da estratégia de automação de testes foram executadas pelo autor desta pesquisa, no papel de analista de testes responsável pelos testes automatizados. A seguir, são apresentados os passos executados para a realização do trabalho e mais adiante, a ordem em que eles foram executados:

- Analisar o processo de testes adotado pela empresa: Neste ponto, foi realizado um estudo detalhado das atividades do processo de teste adotado pela empresa como nome fictício Software QA. O intuito desse estudo foi coletar informações sobre o processo de testes seguido, sendo possível detectar quais atividades do processo de testes podem ser melhoradas ou removidas para adequar-se à nova estratégia de automação de testes. Esta parte foi realizada por observação do autor na empresa Software QA durante uma *sprint*[2].
- Definir uma estratégia de automação de testes para o projeto: Nessa etapa, foi realizada a definição das atividades do processo de automação, bem como, a especificação dos produtos de cada atividade e seus insumos, indicação dos papéis e pessoas envolvidas na execução das atividades e as ferramentas

[2] Sprint representa um intervalo de tempo dentro do qual um conjunto de atividades deve ser executados. (Disponível em http://www.desenvolvimentoagil.com.br/scrum/ acessado dia 1/01/1016)

necessárias para realização das atividades estabelecidas. Essas definições foram embasadas pela experiência do autor e uma pesquisa detalhada na literatura sobre processos de teste de software envolvendo automação de testes.

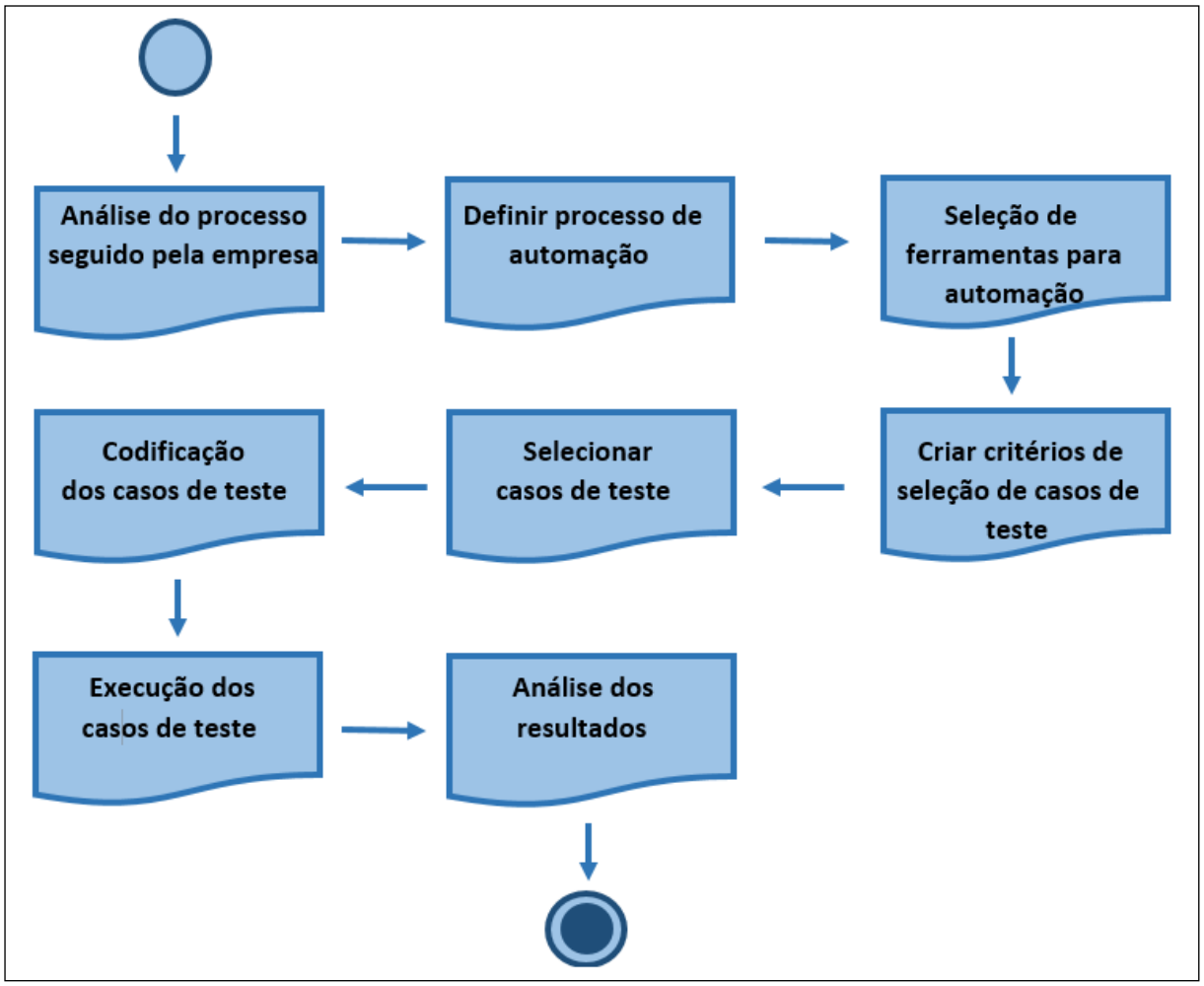
O resultado dessa etapa foi a definição de uma estratégia de automação de testes a ser implantada na empresa de TI em estudo com o objetivo de diminuir o retrabalho dos testes de regressão e aumentar a cobertura dos testes realizados.

- Seleção da ferramenta para automação de teste: O objetivo deste ponto do trabalho foi analisar as ferramentas para automação de teste que se adequaram às necessidades da estratégia estabelecida e que dão suporte às atividades estabelecidas. Ao final do estudo, foi escolhida uma ferramenta para automatizar os testes, alinhada com os requisitos do processo e da aplicação a ser testada.
- Definir critérios de seleção para os casos de testes para automação: Nessa etapa, foram criados os critérios para seleção dos casos de testes a serem automatizados.
- Selecionar casos de testes para automação: Com os critérios de seleção de casos de testes já definidos, esta parte do trabalho deu-se por submeter todos os casos de testes elaborados pelo testador durante a *sprint* corrente aos critérios já definidos.
- Codificação dos casos de teste: Após a seleção dos casos de teste a serem automatizados e a escolha da ferramenta para automatizar os testes, esta etapa teve como objetivo a escrita das classes de testes. Nela, foram implementados os casos de testes automatizados em linguagem JAVA, com auxílio das ferramentas Selenium e JUnit e do ambiente de desenvolvimento Eclipse.
- Executar testes automatizados: Após os testes já estarem codificados, este ponto tem como objetivo a execução dos casos de testes. Os testes foram executados em ambiente controlado com ajuda do ambiente de desenvolvimento Eclipse, e seus resultados são capturados e armazenados para posteriormente serem analisados, com ajuda do JUnit.
- Análise dos Resultados: Após a execução dos testes automatizados, os resultados foram capturados, e, nesta etapa do projeto, foi realizada a análise dos resultados, tendo como principais critérios, a cobertura dos testes automatizados versus a cobertura dos testes manuais e a redução do retrabalho

na realização de testes de regressão. Os critérios secundários para análise foram: redução de riscos de novas falhas em áreas estáveis do software, resultantes da inserção de novas funcionalidades e a agilidade para testar a aplicação em diferentes ambientes de teste.

A seguir, a ordem em que as atividades citadas acima foram realizadas é apresentada na figura 4. O círculo representa o início do processo, os retângulos com curvas representam as atividades, o duplo círculo representa o final da sequência das atividades e as setas representa a ordem em que as atividades foram realizadas.

Figura 4 - Fluxograma da metodologia deste trabalho



Fonte - Elaborada pelo autor

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

Para execução destas atividades no presente trabalho, foi utilizada uma experimentação em situação real, na qual os pesquisadores intervêm conscientemente. Os participantes não são reduzidos a “cobaias” e desempenham um papel ativo. As variáveis, de seu lado, não são isoláveis, posto que todas elas interferem no que está sendo observado. Esta experimentação chama-se pesquisa-ação e foi explicada na seção 4.1. A seguir é descrito o detalhamento do desenvolvimento deste trabalho.

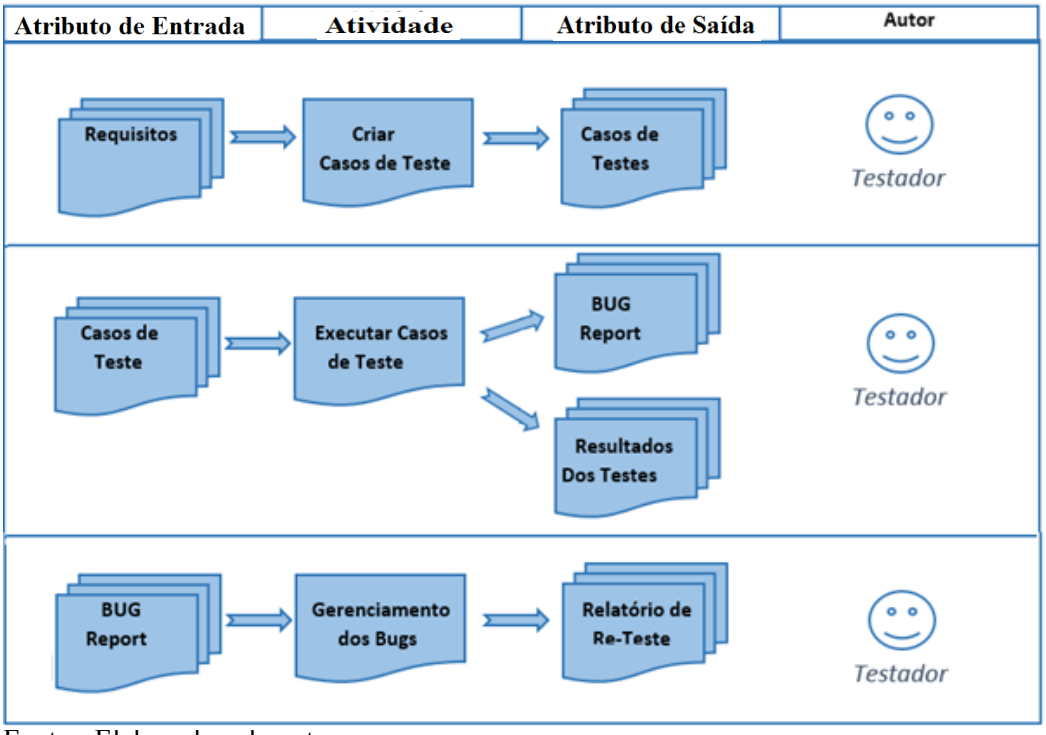
5 DESENVOLVIMENTO

5.1 Análise do processo de testes adotado pela empresa

A empresa Software QA segue um processo de desenvolvimento baseado no *framework Scrum*, que é uma metodologia ágil para gestão e planejamento de projetos de software. No scrum, os projetos são divididos em pequenos ciclos chamados *sprints*, a duração de cada *sprint* é de aproximadamente 15 dias, em cada *sprints* são realizadas as fases de desenvolvimento e testes. Estar fora do escopo deste trabalho, a análise do processo geral adotado pela empresa, pois o trabalho está focado apenas na atividade de teste e não propõe qualquer alteração nas outras atividades.

As atividades de testes analisadas são apresentadas na Figura 5. Os retângulos curvados representam as atividades, já os múltiplos retângulos representam os artefatos de entrada e saída de cada atividade, as setas entalhadas indicam a direção em que os artefatos de entrada e saída são usados pelas atividades.

Figura 5 - Atividades de Testes adotadas pela Software QA.



Fonte - Elaborada pelo autor

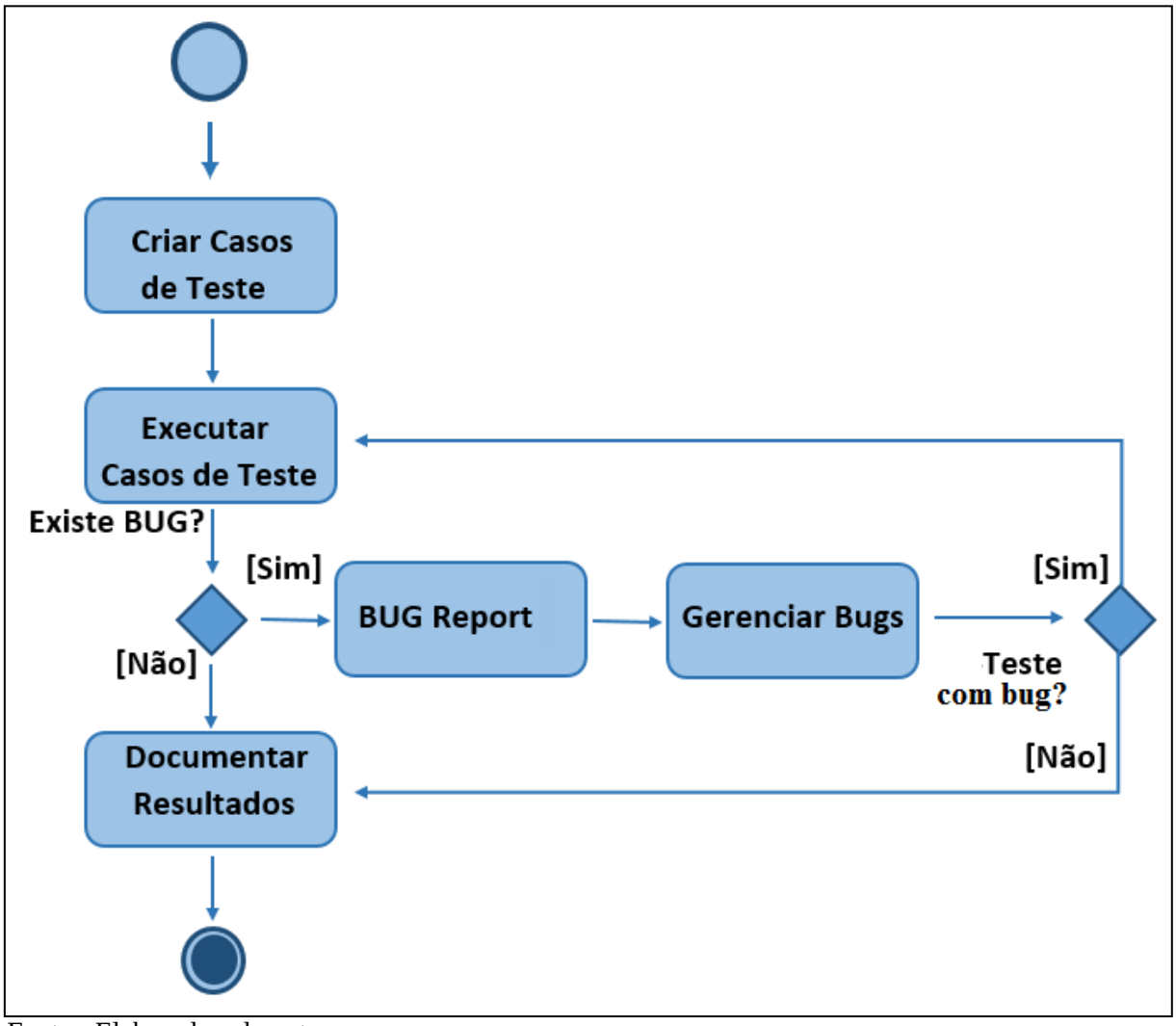
- Criar casos de testes: O responsável por realizar esta atividade é o testador. Nesta atividade, o testador possui os requisitos da funcionalidade a ser testada como artefato de entrada da atividade, baseada nestes requisitos ele elabora os casos de testes. No final da execução desta atividade, o testador possui um documento com os casos de testes que serão executados, quando a

funcionalidade for liberada para teste. Este documento com os casos de testes compõe o artefato de saída desta atividade.

- Executar casos de testes: O testador, responsável por executar essa atividade, possui os casos de testes como artefatos de entrada dessa atividade. Ele deve executar os casos de testes no sistema e caso os casos de testes que detectem bugs. Ele deve criar um *bug report* que é uma chamada aos desenvolvedores para solucionar tal problema. Outro artefato de saída desta atividade é o resultado da execução dos testes, que é um documento que indica quais casos de testes foram executados, quais não detectaram bugs e quais detectaram.
- Gerenciar *bugs*: Após a execução dos testes, alguns bugs podem surgir, e é de responsabilidade do testador gerenciar esses bugs detectados. Nessa atividade, ele possui como artefato de entrada os *bugs reports*, assim ele pode acompanhar quais bugs devem ser gerenciados. O gerenciamento dos bugs ocorre durante todo o ciclo de vida do desenvolvimento do software e se inicia quando o primeiro bug é identificado. Também faz parte desta atividade realizar os testes novamente desses bugs, que acontecem quando a equipe de desenvolvimento resolve o bug identificado e disponibiliza a solução para o testador executar novamente o caso de teste que detectou o bug. Caso o bug esteja realmente corrigido, ou seja, a execução do caso de teste não detectou nenhum bug, ele é dito como resolvido e o testador documenta esse resultado no seu relatório de re-teste. Caso contrário, o caso de teste foi executado e continua a detectar o bug, ele é dito como reaberto, e o testador documenta o caso no seu relatório de re-teste que é um artefato de saída desta atividade e abre novamente um bug report.

O relacionamento entre as atividades e os fluxos de decisões seguidos pelo testador durante a execução é apresentado na Figura 6. O círculo representa o início do fluxo e o duplo círculo representa o fim, os retângulos representam as atividades e as setas a direção e ordem entre elas, já o losango representa uma tomada de decisão durante a realização do fluxo.

Figura 6 - Fluxo de atividades de teste.



Fonte - Elaborada pelo autor

A estratégia de testes adotada inicia na tarefa de criação dos casos de testes, esses casos de testes são documentados e armazenados em um ambiente colaborativo de domínio exclusivo da empresa Software QA. Os casos de testes utilizam a técnica de testes funcionais e são baseados nos requisitos do usuário. Quando o testador possuir uma implementação disponível para teste, ele executa os casos de teste com o objetivo de detectar bugs na implementação. Essa execução dos testes funcionais ocorre no ambiente de teste. Caso sejam detectados *bugs*, o testador abre um *bug report* informando que existe um bug na implementação disponibilizada para teste. Assim, a equipe de desenvolvimento decide quando e qual dos desenvolvedores deve resolver o bug. Com a detecção do *bug*, outra atividade é iniciada, a atividade de gerenciamento de *bug*. Nela, o testador gerencia os *bugs* existentes e verifica quais estão sendo corrigidos e disponibilizados no ambiente de teste. A empresa Software QA utiliza um ambiente controlado para esse gerenciamento. Quando um bug é dito resolvido e disponibilizado no ambiente de teste, o testador executa novamente o caso de teste

que detectou o bug. Caso seja constatada a correção do bug, ele é fechado e seu resultado é documentado. Caso contrário, o *bug report* é reaberto.

Durante a análise do processo seguido pela empresa, foi possível detectar que ela não utiliza nenhuma atividade de testes automatizados. Também ficou claro que o uso de diferentes ambientes de desenvolvimento para versionamento de código pela empresa ajuda na proliferação de bugs. A empresa utiliza três ambientes; desenvolvimento, teste e produção. Quando uma nova funcionalidade deve ser implementada, ela é construída no ambiente de desenvolvimento, ao qual somente os desenvolvedores possuem acesso. Quando a implementação é dita concluída pelos desenvolvedores, eles disponibilizam essa implementação no ambiente de teste, onde somente o testador possui acesso. Nela o testador executa seus casos de testes. Quando o testador atestar a funcionalidade livre de bugs na execução dos cenários de testes, ela é disponibilizada no ambiente de produção, ao qual somente os usuários finais possuem acesso.

No decorrer deste trabalho, alguns pontos de falhas foram detectados pelo autor deste trabalho no processo de atualização dos ambientes:

- Não existem atividades de testes automatizados no fluxo de inserção de novas funcionalidades nos ambientes.
- Quando uma funcionalidade é “disponibilizada” para o ambiente de teste, ela pode conter erros, dificultando o controle do ambiente.
- Não existe atividade de teste de regressão na atualização dos ambientes.
- Não existe atividade de teste durante as atualizações “disponibilização” das funcionalidades em todos os ambientes.
- Não existe um ambiente controlado livre de bugs detectados durante a execução dos casos de teste.

Estes pontos de falhas analisados, provocam problemas no gerenciamento do ambiente de teste, bem como, impactos negativos na qualidade do sistema entregue aos usuários finais da aplicação.

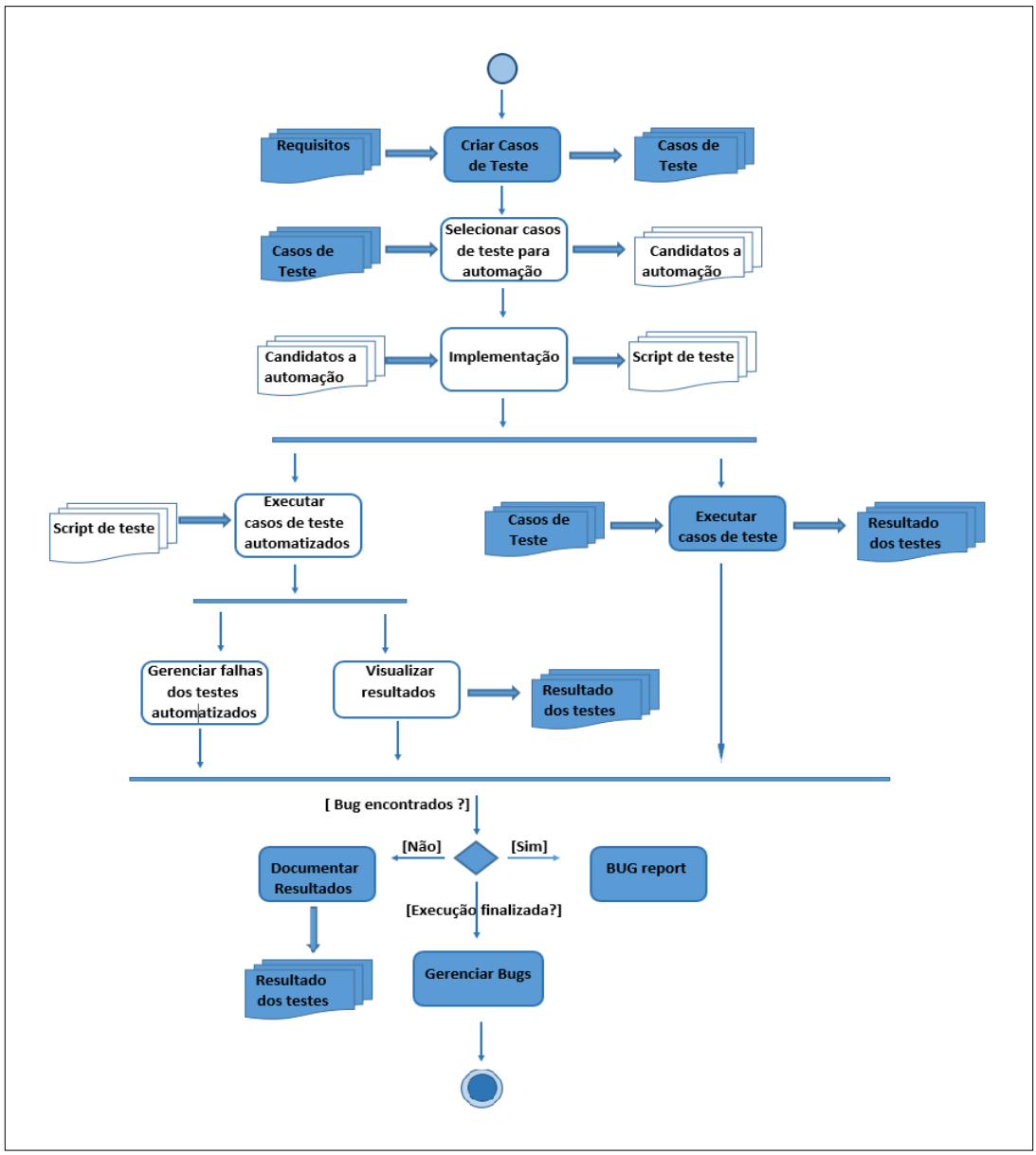
Como forma de resolver esses problemas identificados, esse trabalho propõe um conjunto de atividades para incluir a automação de testes na atualização dos ambientes da empresa que é apresentado a seguir.

5.2 Definição da Estratégia para automação de testes

A estratégia para automação dos testes, sugerida por este trabalho como fator de contribuição desta pesquisa, foi organizada de forma a não incluir impactos negativos no processo já adotado no projeto, levando em consideração que trata-se de um projeto real, e mudanças no seu processo poderiam representar riscos ao projeto. Com base nisso, as atividades de automação foram conduzidas em paralelo com às atividades de testes manuais. Essa decisão também permite comparar os resultados obtidos com a automação em relação ao ambiente de testes manuais.

Todas as atividades de testes automatizados foram executadas pelo analista de teste, sendo o mesmo responsável por este trabalho. A seguir, é apresentada na figura 7, a estratégia de automação dos testes sugerida.

Figura 7 - Estratégia de testes sugerida por este trabalho.



Fonte - Elaborada pelo autor

As atividades representadas pelos retângulos com cantos arredondados sem preenchimento são atividades ligadas à automação dos testes, que ocorrem em paralelo às atividades do processo de testes manuais adotados pela empresa em estudo, e estas são representadas pelos retângulos com cantos arredondados com preenchimento na cor azul.

Estas atividades de automação de testes sugeridas pela estratégia de automação foram desenvolvidas segundo a área de processo AET – Automação da Execução do Teste, definido pelo Modelo de Melhoria do Processo de Testes Brasileiro (MPT.Br).

A atividade de seleção de casos de teste para automação é melhor analisada na seção 5.4 onde são descritos os critérios de seleção de casos de teste. O artefato de saída casos de testes candidatos à automação, representado pelos múltiplos retângulos, consiste em um documento com os casos de testes existentes e sua avaliação sobre os critérios de seleção. Neste trabalho, pode-se analisar este documento na seção 5.5.

As atividades de implementação dos testes, execução e visualização dos resultados são apresentadas nas seções 5.6, 5.7 e 6, respectivamente.

Gerenciar falhas de testes automatizados é uma atividade responsável pela identificação, análise e gerenciamento dos incidentes ocorridos na execução de teste automatizados. Essas falhas podem ocorrer por diversos motivos, alguns deles podem ser: sistema em teste indisponível, componente de acesso removido, atualização dos navegadores, entre outros. As falhas são documentadas e analisadas, pois podem representar um defeito no software bem como nos *scripts* de teste. Eles são descritos na subseção 5.7.1.

Durante a análise do processo seguido pela empresa, foi identificada pelo autor deste trabalho, a necessidade de sugerir, junto à estratégia de automação dos testes, um conjunto de atividades para execução dos casos de teste no fluxo de trabalho do versionamento de código e controle dos ambientes de desenvolvimento. A equipe utilizava três ambientes onde o fluxo de alteração nos códigos é muito grande. Essas alterações, por inclusão de código ou correção de *bugs*, na maioria das vezes, não possuíam testes e assim possuía uma grande chance de causar a proliferação de *bugs* nos ambientes. Na seção 5.1, onde a estratégia de automação dos testes é apresentada, analisamos e mostramos como a equipe trabalha no versionamento de código e controle dos ambientes de desenvolvimento além de citarmos pontos de falhas. Com base nesses pontos e, como parte da contribuição deste trabalho, a seguir são descritas atividades, com o objetivo de minimizar a proliferação de bugs nos ambientes:

- **Atualizar *branch*:** Após implementação de uma nova funcionalidade ou correção de um *bug* o desenvolvedor deve submeter, caso haja *suíte* de teste, o seu "*BRANCH*" referente a funcionalidade ou *bug* corrigido. Nesta atividade, o desenvolvedor executa um conjunto de casos de teste, chamado de *suíte* de teste, especifico referente a sua implementação. Caso seu *branch* passe nos testes, ele deve fazer um *pull-request*, que é uma submissão aos membros da equipe para avaliação do código desenvolvido.

  Caso contrário, se os casos de teste da suíte não passarem, o desenvolvedor não deve realizar o *pull-request* para ambiente de desenvolvimento para não contaminá-lo com *bug's*. Ele deve no seu *branch* corrigir os *bugs* e submeter novamente à *suíte*.
- **Atualizar o ambiente de teste:** Após seu *branch* ser aprovado pela suíte de teste e o pull-request ser aceito para o ambiente de desenvolvimento, deve-se realizar um *cherry-pick* para QA e fazer um novo *deploy*. Esse *cherry-pick* faz com que a atualização possua uma indicação (ponteiro) mostrando de onde a atualização partiu. Com o *cherry-pick* feito, o ambiente de testes estará atualizado com as novas implementações que passaram nos testes de regressão daquela funcionalidade.
- **Ambiente de testes:** Quando uma funcionalidade ou correção de *bug* for disponibilizada no ambiente de QA, a equipe de teste deve executar os testes manuais não contemplados pelos testes automatizados e em seguida executar uma bateria de testes automatizados do sistema como um todo. O objetivo dos testes manuais é detectar *bug's* não encontrados pela execução dos testes automatizados e o objetivo da execução da suíte de testes é verificar se houve regressão quanto à inserção da nova funcionalidade ou correção de bug no ambiente.
- **Atualizar ambiente *alfa*:** Caso não tenha existido problema de regressão no sistema com a nova implementação ela deve fazer parte do ambiente *alfa*, esse ambiente só recebe implementação, quando forem executados todos os testes automatizados e os testes manuais sem a detecção de erros.

  Caso contrário, se a bateria de testes, quando executada, possuir falhas, ou podendo ser os casos manuais, essas devem ser analisadas e documentadas pela equipe de QA e, em seguida, enviadas para à equipe de desenvolvimento

para correção. Nesse caso, os desenvolvedores devem criar um novo *branch* a partir do ambiente de desenvolvimento para correção do *bug* e não atualizar *alfa* com a funcionalidade.

- **Atualizar produção:** Quando houver necessidade de atualização do ambiente de produção a equipe deve clonar o ambiente *alfa* e disponibilizar como novo ambiente de produção.

A sugestão de um ambiente *alfa* também faz parte das contribuições deste trabalho. O ambiente *alfa* contém as implementações livres de *bug's* encontrados até o momento, por esse motivo ele é necessário para o controle dos ambientes e atualização do ambiente de produção.

A seguir são apresentados os critérios definidos para a seleção da ferramenta de automação a ser utilizada.

5.3 Ferramenta de Apoio à Automação do Testes

As ferramentas para automação dos testes abrangem um vasto leque de atividades, e podem ser aplicadas em todas as fases do ciclo de vida do desenvolvimento de um sistema (PERRY, 2006). Bastos *et al*. (2007) defendem a necessidade da utilização de uma ferramenta de teste quando se existe uma forte pressão para melhorar a qualidade do software em questão, isso quando o projeto não possuir condições da realização de testes tradicionais. Villas Boas (2003) completa, dizendo que as ferramentas são importantes para reduzir a intervenção humana na atividade de teste, e que sua utilização aumenta a produtividade e influência na confiabilidade do sistema testado.

Essas ferramentas de apoio a automação e execução dos testes automatizados fazem parte do ambiente de teste. Anderson Bastos *et al.* (2012) dizem que este ambiente não é apenas uma configuração de hardware, mas toda a estrutura onde o teste será executado. Considera-se os seguintes atributos de ambiente: escopo do sistema, equipe, volume de dados, origem dos dados e interface.

Segundo Bartié (2002), a utilização de ferramentas de testes possui diversos benefícios, dentre eles, podemos citar a redução do tempo na execução de atividades que envolvem o ciclo de vida do teste.

As próximas seções descrevem as características analisadas para escolha da ferramenta de automação utilizada por este trabalho e o ambiente de testes automatizados construído e utilizado por este trabalho.

5.3.1 Escolha da ferramenta para automação dos testes

Durante a realização desse estudo foi possível identificar uma variedade de ferramentas de automação de testes. A melhor ferramenta para automação depende da estratégia de automação à qual ela será inserida e o ambiente da empresa. Essa seção avaliará as seguintes características: *Record e playback*, teste web, mapeamento de objetos na tela, extensibilidade, suporte ao ambiente e integração. Foram avaliadas as seguintes ferramentas: Compuware QARum[3], Mercury QuickTest Professional[4], Rational Function Test[5], Borland Segue SilkTest[6] e Selenium[7]. As características avaliadas das ferramentas foram levantadas de acordo com a observação do autor deste trabalho no ambiente ao qual a ferramenta escolhida será implementada. A descrição de cada característica será abordada a seguir:

5.3.1.1 *Record & PlayBack*

Essa característica é essencial para a automação de testes, ela é o método pelo qual a ferramenta de automação irá gravar os *scripts*, observar e reproduzi-los.

Esta categoria possui os seguintes detalhes a serem considerados:

- Facilidade de reprodução e gravação dos testes.
- Facilidade de leitura dos *scripts* pela ferramenta.

5.3.1.2 Teste Web

Baseado no desenvolvimento de software para web, essa característica ressalta a importância das ferramentas de automação possuírem suporte para as páginas web. Os critérios desta categoria são:

- Existe funções que permitem verificar a existência de carregamento de elementos da página, assim como a própria página?
- Pode-se testar a funcionalidade dos *links*?
- Existe diferenciação entre os tipos e as localizações de objeto?
- Existe diferenciação entre os campos da página?

[3] http://www.compuware.com/
[4] http://www.mercury.com/us/products/quality-center/functionaltesting/quicktest-professional/
[5] http://www-03.ibm.com/software/products/en/functional
[6] http://www.borland.com/Products/Software-Testing/Automated-Testing/Silk-Test
[7] http://www.seleniumhq.org/

5.3.1.3 Mapeamento de Objetos

Esta característica é essencial para identificação e mapeamento de objetos na tela do sistema, é essencial que a ferramenta de automação consiga controlar esses objetos da *interface* do sistema.

5.3.1.4 Extensibilidade

Essa característica está relacionada com a extensão da ferramenta de automação.

Se a ferramenta não suporta uma determinada funcionalidade, ela suporta a criação de uma nova? Esse tipo de pergunta deve ser respondido nesta característica.

5.3.1.5 Suporte ao Ambiente

A quais ambientes a ferramenta dá suporte? Essa é uma pergunta importante, pois se a ferramenta não suporta o ambiente no qual será implantado, isto pode ser um grande problema.

5.3.1.6 Integração

Perguntas como: Como acontece a integração da ferramenta de automação com outras ferramentas? A ferramenta permite a execução de testes a partir de *suítes* de teste? devem ser respondidas nesta característica.

Para cada característica que foi avaliada pelo autor do trabalho é atribuída uma nota de 1 a 4:

- 1 = nenhum suporte.
- 2 = suporte básico apenas.
- 3 = suporte bom, mas incompleto.
- 4 = suporte satisfatório.

A Tabela 2 apresenta a avaliação das ferramentas a partir das características definidas. Onde o total de pontos é apresentado. Vale ressaltar que essa é uma avaliação subjetiva, baseada na experiência do autor. Assim, a ferramenta escolhida foi a Selenium WebDriver, ela é explicada em detalhes na seção 5.3.2.

Tabela 2 - Comparação entre as ferramentas de automação.

Ferramentas	***Record* & *Playback***	**Teste de Web**	**Mapeamento de Objetos**	**Extensibilidade**	**Suporte de Ambiente**	**Integração**	**Total**
Functional Teste	4	4	3	4	4	3	19
SilkTest	4	3	2	3	3	3	18
Selenium	4	4	4	4	4	4	24
QuickTest	3	4	4	3	3	3	20
QARum	3	4	4	4	3	3	21

Fonte: Elaborada pelo o autor

Após a análise das ferramentas, a ferramenta que obteve melhor desempenho nos aspectos avaliados foi a ferramenta Selenium com um total de 24 pontos.

5.3.2 Selenium

Segundo SeleniumHQ (2012), Selenium é um conjunto robusto de ferramentas de código aberto que apoia o desenvolvimento rápido de testes automatizados, oferecendo um rico conjunto de funções de teste de aplicações baseadas na web. Estas operações são muito flexíveis, permitindo muitas opções para a localização de elementos da interface do usuário e comparar os resultados dos testes esperados contra o comportamento obtido. É uma ferramenta de fácil uso e eficiente para desenvolver casos de teste, permitindo os testes de aceitação ou funcional, regressão e de desempenho. A Tabela 3 ilustra as características do Selenium observadas pelo autor durante a sua utilização na realização deste trabalho.

Tabela 3 - Características da ferramenta Selenium.

É um conjunto de quatro ferramentas open source: Selenium IDE, Selenium RC, Selenium Web Driver e Selenium GRID.
Suporte às linguagens Java, C#, Perl, PHP, Python, Ruby, entre outras.
É possível executar testes em qualquer navegador com suporte a JavaScript.
Suporta diversos sistemas operacionais.
Ambiente integrado para elaboração de casos de teste.
É uma ferramenta de código aberto (open source).
Permite gravação e reprodução das gravações feitas.
Permite debug dos scripts de teste e utilização de breackpoints.
Permite a realização de testes de sistema e testes de regressão.

Possui uma comunidade ativa.

Fonte - Elaborado pelo autor

O Selenium é composto por quatro principais ferramentas:

- **Selenium IDE:** É um ambiente de desenvolvimento integrado para construção de casos de teste. Ele opera como uma extensão do Firefox e provê uma interface amigável para o desenvolvimento e execução de conjuntos de testes. O Selenium IDE é uma ferramenta do tipo *record-and-playback*, ou seja, ela captura as ações executadas pelo testador e gera um *script* que permite a re-execução das ações feitas, automatizando assim, o teste. (SANTOS *et al*. 2009)

- **Selenium RC (Remote Control):** Possibilita uma maior flexibilidade ao testador, permitindo a construção de lógicas de teste mais complexas, a partir do uso de uma linguagem de programação. Para isso, ele provê uma API *(Application Programming Interface)* e bibliotecas para cada uma das linguagens suportadas: HTML, Java, C#, Perl, PHP, Python e Ruby. (SANTOS *et al.* 2009)

- **Selenium WebDriver**: O Selenium WebDriver é uma ferramenta open source que permite automatizar a execução de cenários de testes via código-fonte. Ele é uma junção do antigo Selenium RC com o Selenium IDE. Ele pode ser integrado ao ambiente Java, .Net, Rails e outras ferramentas de automatização de testes como o JUnit.

- **Selenium GRID:** Permite distribuir os testes em múltiplas máquinas, reduzindo assim o tempo gasto na execução de um conjunto de testes. Ele é ideal para escalonar conjuntos de testes grandes ou conjuntos que devem ser executadas em múltiplos ambientes. O Selenium Grid atua executando múltiplas instâncias do Selenium RC em paralelo de forma transparente, fazendo com que os testes não precisem se preocupar com a infra estrutura utilizada. (SANTOS *et al.* 2009)

Após a seleção da ferramenta Selenium WebDriver, é necessário montar o ambiente de execução dos testes. A seção a seguir apresenta os passos para criação desse ambiente.

5.3.3 Montagem do Ambiente

Segundo Bastos *et al*. (2012), a preparação do ambiente de testes isolado, organizado, representativo e mensurável assegura a descoberta de erros reais. Para que os testes automatizados pretendidos por esse trabalho fossem executados, foi necessário construir um ambiente de testes.

Segundo Emerson Rios e Trayahú Moreira (2013), ao definirmos a ambiente de teste, devemos considerar os seguintes aspectos:

- Sistema Operacional.
- Arquitetura do sistema.
- Identificação dos componentes.
- Linguagem de Programação.
- Conectividade entre os ambientes.

Após levarmos em consideração os aspectos descritos por Emerson e Trayahú (2013) descreveremos a seguir o ambiente de testes construído para execução dos testes automatizados gerados por este trabalho nos aspectos pessoal, hardware, software, documentação e ambiente de trabalho.

5.3.3.1 Pessoal

As atividades de desenvolvimento do ambiente de testes, criação dos casos de testes, execução dos casos de testes, documentação dos resultados e acompanhamento dos *reports* de *bug* tiveram como responsável o analista de teste. Ele é o único envolvido na execução de atividades ligadas a testes na empresa em estudo. Ele também é o autor responsável por esse trabalho. Para a construção do software a ser testado, a empresa dispunha de 6 desenvolvedores e um líder técnico.

5.3.3.2 Hardware

A plataforma Java é utilizada neste trabalho, pois ela permite desenvolver aplicativos utilizando qualquer uma das linguagens criadas para a plataforma, além de o sistema a ser testado estar em desenvolvimento nesta plataforma. Uma grande vantagem da plataforma é a de não estar presa a um único sistema operacional ou hardware, pois seus programas rodam através de uma máquina virtual que pode ser emulada em qualquer sistema.

O universo Java é um vasto conjunto de tecnologias, composto por três plataformas principais que foram criadas para segmentos específicos de aplicações:

- Java SE (Java Platform, Standard Edition). É a base da plataforma. Inclui o ambiente de execução e as bibliotecas comuns.
- Java EE (Java Platform, Enterprise Edition). A edição voltada para o desenvolvimento de aplicações corporativas e para internet.
- Java ME (Java Platform, Micro Edition). A edição para o desenvolvimento de aplicações para dispositivos móveis e embarcados.

Para a construção do sistema a ser testado foi utilizada a plataforma Java EE. Além da plataforma Java, são utilizadas outras tecnologias para o desenvolvimento: MongoDB e nodeJs.

5.3.3.3 Software

O sistema sob teste por esse trabalho permite que seus clientes, seja qual for o segmento de atuação e o porte, analisem dados de diferentes fontes como sistemas internos, redes sociais e lojas de aplicativos, por meio de módulos e modelos de dados pré-configurados. Este modelo permite que os clientes iniciem sua jornada para a análise massiva de dados, sem a necessidade de grandes investimentos em hardware, software e serviços profissionais. Ele apresenta uma base sólida e uma proposta de construção de um software completo que cumpra todas as etapas do ciclo de vida da análise de dados, como coleta, agregação, análise e visualização. Com o sistema, o cliente pode agregar dados de seus sistemas internos para buscar respostas baseadas em projeções e cruzamento de informações de diversas fontes. O sistema oferece, também, aplicativos que proporcionam respostas eficazes aos principais questionamentos da empresa, como, por exemplo, o comportamento de seus consumidores nas redes sociais, o grau de satisfação de seus clientes e, até mesmo, o monitoramento das lojas virtuais de aplicativos móveis.

Para testar esse sistema, foi selecionado a ferramenta Selenium WebDriver. No ponto 5.3.1 foram detalhados os critérios e a avaliação do Selenium como fermenta para automação. Já no ponto 5.3.2 foram mostradas as caraterísticas desta ferramenta.

Além do Selenium, são utilizadas outras ferramentas para auxílio na automação de sistema, que são descritas a seguir:

- Junit é um *framework* que possibilita a criação de testes em Java. O objetivo do *framework* é facilitar a criação de casos de teste, possibilitando ao desenvolvedor executar seu caso de teste e no final verificar o estado atual com o esperado. Ele ainda previne que o desenvolvedor escreva testes duplicados e permitir escrever

testes que retenham seu valor ao longo do tempo, ou seja, que possam ser reutilizáveis.

Ele ainda oferece um conjunto de métodos assertivos, que ajudam o desenvolvedor a verificar o resultado final da execução do seu teste, ou seja, comparar o estado do sistema antes e depois da execução do caso de teste, ou simplesmente verificar o estado de um determinado atributo ou objeto após a execução do cenário de teste. Além dos métodos assertivos, o JUnit oferece classes para execução de suítes de testes e uma amigável interface onde o resultado dos testes é acompanhado. A seguir, é apresentado na Figura 8 um caso de teste automatizado, nele são utilizados o Selenium WebDriver em conjunto com JUnit. As anotações *before*, *test* e *after* são restritas do JUnit, essas anotações facilitam a execução dos casos de teste. Já o método assertTrue, também restrito ao JUnit, possibilita uma comparação lógica entre dois elementos.

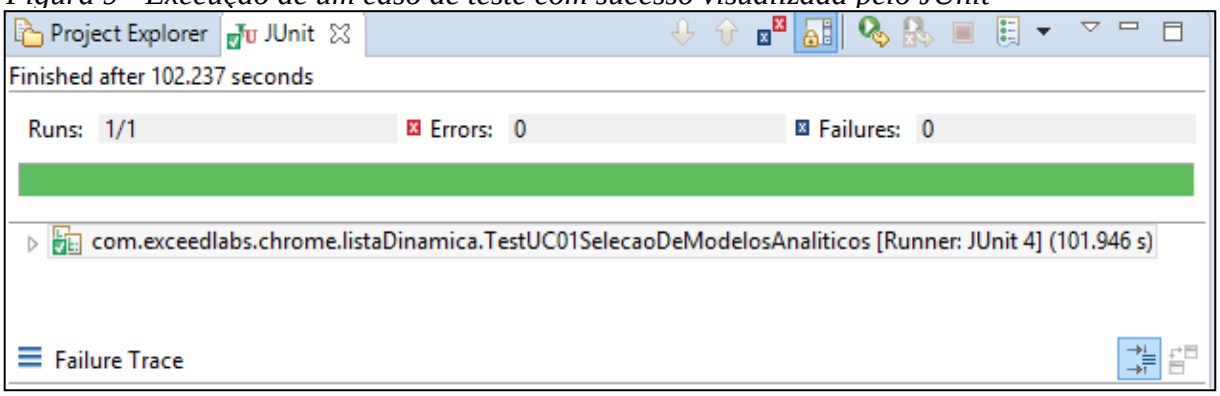
Figura 8 - Classe de teste implementada com Selenium e JUnit em linguagem JAVA

```
18
19 public class TestUC01SelecaoDeModelosAnaliticos {
20     private WebDriver driver;
21     private Login login;
22     private WebDriverWait wait;
23     private PaginaListaDinamica paginaListaDinamica;
24     private boolean resultDaBuscaDoModeloAnalitico;
25
26     @Before
27     public void inicializar() {
28         login = new Login();
29         Driver webDriver = new Driver();
30         this.driver = webDriver.getDriver();
31         this.wait = new WebDriverWait(driver, 100);
32         this.paginaListaDinamica = new PaginaListaDinamica();
33     }
34
35     @Test
36     public void selecaoModelosAnalitcos() throws InterruptedException {
37         // Passos:
38         // Passo 1: Realizar Login
39         login.login(driver);
40         // Passo 2: Ir para a página de Lista de contatos
41         paginaListaDinamica.paginaAdicaoListaDinamica(wait, driver);
42         // Passo 3: Peencher Dados da Lista Dinaâmica
43         paginaListaDinamica.preencherDadosListaDinamica(driver, wait, "Lista",
44                 "UC01 - Selecionar Modelo Analitico");
45         // Passo 4: Buscar modelo analitico
46         resultDaBuscaDoModeloAnalitico = paginaListaDinamica
47                 .selecaoModeloAnalitico(driver, wait, "Emails");
48         // Passo 5: Verificando se o modelo analitico procurado foi encontrado.
49         assertTrue(resultDaBuscaDoModeloAnalitico);
50     }
51
52     @After
53     public void finalizrar() throws InterruptedException {
54         driver.close();
55     }
56 }
57
```

Fonte – Elaborada pelo autor.

Além dos métodos assertivos e as anotações pertencentes ao JUnit, ele ainda possibilita uma interface intuitiva para acompanhamento da execução dos casos de teste. Ao executar um caso de teste temos duas possibilidades, a primeira delas é que, durante a execução do caso de teste é encontrado um erro, podendo esse ser considerado um *bug*. A segunda é que todos os passos são executados e não é detectada nenhuma anomalia na execução dos passos, ou seja, o resultado do teste foi favorável. No JUnit é possível acompanhar esse resultado via interface. Abaixo pode ser visto, na Figura 9, a execução de um caso de teste e a representação da barra verde informando que o caso de teste foi executado com sucesso, caso contrário, o JUnit apresenta uma barra na cor vermelha.

Figura 9 - Execução de um caso de teste com sucesso visualizada pelo JUnit



Fonte - Elaborada pelo autor

- O Eclipse é um Ambiente de Desenvolvimento Integrado (IDE) extremamente poderoso e flexível para diversas finalidades: desenvolvimento Java, Android, C++, modelagem de processos, desenho de relatórios e assim por diante. Neste trabalho, ela serviu para desenvolvimento dos *scripts* de testes automatizados em linguagem Java, utilizando as APIs do Selenium WebDrivere JUnit.
- WebDriver é uma ferramenta de código aberto para testes automatizados de aplicações web por meio de muitos navegadores. Neste trabalho, foram utilizados os WebDrivers dos navegadores Firefox, Internet Explorer e Chrome. A escolha dos três navegadores corresponde aos requisitos da aplicação.
- Java é uma linguagem de programação orientada a objetos de alto nível. O programa desenvolvido é executado em uma *Java Virtual Machine* que está disponível em um grande número de dispositivos, possibilitando assim à linguagem um grande poder em portabilidade.

A interação entre as ferramentas é clara. Como base do ambiente, temos o Java EE sendo utilizado para o desenvolvimento dos casos de testes automatizados, logo acima dessa plataforma, temos a IDE Eclipse que executa suas operações na plataforma Java EE, ela permite inserir as bibliotecas do Selenium WebDriver e JUnit para construção dos *scripts* de testes em linguagem Java, e instanciação dos navegadores onde os *scripts* de testes serão executados. A seguir são apresentadas na Tabela 4 as versões das ferramentas utilizadas.

Tabela 4 - Ferramentas e Versões

Ferramenta	**Versão**
Selenium WebDriver	2.48.2
Java EE	7.0
Eclipse	Release 2 (4.4.2)
JUnit	4.11
Chrome WebDrive	2.20
Firefox WebDrive	42.0
Intenter Explorer WebDriver	11.0.18

Fonte - Elaborada pelo autor

5.3.3.4 Documentação

A documentação consta de casos de testes construídos para contemplar os testes nas *sprints*, os casos de testes são criados pelo analista de testes e disponibilizados em ambiente acessível a todos os membros da equipe.

5.3.3.5 Ambiente de Trabalho

Para a realização do estudo foram utilizadas as instalações da própria empresa. Ela conta com salas climatizadas e terminais de trabalho individuais para os membros das equipes dispostas em baias. O ambiente confortável facilitou a realização deste estudo.

5.4 Critérios de Seleção de Casos de Teste

Segundo o RUP, o caso de teste é a definição de um conjunto específico de valores de entrada, condição de execução e resultados esperados, identificados com a finalidade de avaliar um determinado aspecto de um item de teste alvo.

Já Heumann (2001) define casos de teste como um conjunto de entradas de teste, condições de execução e resultados esperados para um objetivo particular: exercer um caminho particular do programa ou verificar o cumprimento de um requisito específico, por exemplo.

A proposta de um caso de teste, segundo os autores do RUP e Hermann(2001) é identificação e comunicação de condições que seriam implementadas no teste. Casos de teste são necessários para verificar a implementação bem-sucedida e aceitável dos requisitos do produto.

Devido a problemas ligados a prazos reduzidos e diminuição de custos na organização e construção do software, foi necessário estruturar uma estratégia de automação de testes de modo que possa entregar o software no tempo e dentro do orçamento, e igualmente satisfaça as exigências do cliente. Uma alternativa utilizada para esse fim, é a seleção de casos de testes para serem automatizados. Sabemos ainda da importância da priorização dos casos de testes que serão automatizados, pois, os casos de teste que poderiam revelar falhas podem ser rejeitados, reduzindo a eficácia do ciclo de teste.

Existem na literatura, alguns trabalhos relacionados a critérios de seleção de casos de teste que podem ser citados aqui. Um deles é o trabalho de Lima (2014), onde ele lista um conjunto de oito critérios de seleção de casos de teste. Já um segundo, é o trabalho de Rothermel (2003), que considera quatro técnicas que reusam a suíte de testes da versão original do software: *retest-all, regression test selection, test suite reduction, e test case prioritization.* Nenhuma delas foi utilizada por esse trabalho por não possuir aderência ao objetivo principal do projeto em estudo, além dos seguintes aspectos: variáveis de ambientes, organização dos requisitos e elaboração e documentação dos casos de testes.

Dessa forma, como contribuição dessa pesquisa, foram definidos critérios de seleção e escalas para os critérios utilizados. Vale lembrar que para a construção dos critérios levou-se em consideração o trabalho de Softex (2011) nos seguintes aspectos:

- Efetividade.
- Exemplaridade.
- Economia.

Este trabalho adicionou outros critérios e inseriu escalas de avaliação para eles, levando em consideração a experiência do autor sobre o projeto, adquirida durante a construção deste trabalho

- Integração.
- Importância.

A seguir os critérios serão melhor explicados, assim como suas escalas de avaliação.

5.4.1 Efetividade

Os testes são selecionados com o objetivo de possuir uma probabilidade razoável de encontrar erros.

Tabela 5 - Classificação de casos de teste segundo o critério de efetividade.

Escala Qualitativa	Escala Numérica	Descrição
Probabilidade Alta	2	Casos de testes que possuem uma certa complexidade de desenvolvimento, ou podendo incluir novas implementações (refatoração) em componentes já existentes.
Probabilidade baixa	1	Casos de testes que possuem novas implementação (refatoração) em componentes já existentes.

Fonte - Elaborada pelo autor.

5.4.2 Exemplaridade

Os testes são selecionados com o objetivo de serem práticos na construção dos *scripts*, logo é levada em consideração a dificuldade em implementar o caso de teste selecionado pela quantidade de passos que serão implementados no *script*.

Tabela 6 - Classificação de casos de teste segundo o critério de exemplaridade.

Escala Qualitativa	Escala Numérica	Descrição
Facilidade de construção	3	Casos de testes que possuem um número pequeno de passos entre 03 e 04 e podem não apresentar dificuldade na implementação.
Dificuldade média de construção	2	Casos de testes que possuem um número pequeno de passos entre 05 e 06 e podem apresentar dificuldades na implementação.

Alta dificuldade de construção	1	Casos de testes que possuem um número de passos superior a 06 passos e certamente apresentarão dificuldades na implementação.

Fonte - Elaborada pelo autor.

5.4.3 Economia

Os testes são selecionados com o objetivo de possuir um pequeno ou razoável custo de desenvolvimento, levando em consideração o tempo estimado para constrição do *script*.

Tabela 7 - Classificação de casos de teste segundo o critério de economia.

Escala Qualitativa	Escala Numérica	Descrição
Tempo curto de construção	3	Casos de testes que possuem estimativa de tempo de construção menor que 40 minutos.
Tempo médio de construção	2	Casos de testes que possuem estimativa de tempo de construção maior que 40 minutos e menor e igual a 60 minutos.
Tempo alto de construção	1	Casos de testes que possuem estimativa de tempo superior a 60 minutos.

Fonte - Elaborada pelo autor.

5.4.4 Integração

Os testes são selecionados com o objetivo de permitir que os casos de testes que verificam a interação entre componente sejam executados automaticamente.

Tabela 8 - Classificação de casos de teste segundo o critério de integração.

Escala Qualitativa	Escala Numérica	Descrição
Alta Interação	3	Casos de testes que possuem comunicação com componentes externos e internos ao sistema.
Média Interação	2	Casos de testes que possuem comunicação somente com componentes externos.

Baixa Interação	1	Casos de teste que possuem comunicação somente com componentes internos.

Fonte - Elaborada pelo autor

5.4.5 Importância

Os testes são selecionados com o objetivo de automatizar os casos de testes elencados como importantes, ou seja, são essências para a utilização do sistema pelos clientes.

Tabela 9 - Classificação de casos de teste segundo o critério de importância.

Escala Verbal	Escola Numérica	Descrição
Muito importante	3	Casos de testes que possuem um elevado grau de importância aos usuários do sistema e estão contidas no fluxo principal da execução das funções do sistema.
Importante	2	Casos de testes que possuem uma importância aos usuários do sistema, mas não estão contidos no fluxo principal da execução das funções do sistema.
Pouco importante	1	Casos de testes que possuem pouca importância aos usuários do sistema.

Fonte - Elaborada pelo autor.

5.5 Seleção dos Casos de Teste

A partir dos critérios de seleção de casos de teste, detalhados na seção 5.4 deste trabalho, 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 de cada *sprint* abordando todas as funcionalidades que faziam parte da *sprint backlog*, que é um conjunto de atividades a serem implementadas. Os casos de testes foram submetidos ao conjunto de critérios de seleção e em cada caso de teste foi atribuído um valor pertencente à escala numérica dos critérios submetidos. Ao final, foi realizada uma soma dos valores atribuídos aos critérios. Essa pontuação é mostrada na Tabela 10.

A partir dessa avaliação foi possível selecionar os casos de testes candidatos à automação com pontuação maior que 8, possibilitando estabelecer uma prioridade para o desenvolvimento dos casos de testes, implementando primeiro os casos de testes com maior valor para a estratégia de automação. Todos os casos de teste definidos como candidatos

foram automatizados, ao todo 93 casos de testes, possuindo uma porcentagem de 81,57% de todos os casos de testes, restando 18,25 % dos casos de testes não candidatos à automação.

Tabela 10 - Classificação dos casos de teste candidato a automação.

Sprint	**UC**	**TC**	**Descrição**	**Efetividade**	**Exemplaridade**	**Economia**	**Integração**	**Importância**	**Total**	**Resultado**
	[UC01] Criar Web Marketing	TC01	Campos preenchidos	3	1	2	3	3	12	**CANDIDATO**
		TC02	Campos em branco	2	2	1	1	2	8	NÃO CANDIDATO
		TC03	Tags de substituição no assunto	2	2	2	2	2	10	**CANDIDATO**
		TC04	Importe de URL	3	2	2	2	2	11	**CANDIDATO**
		TC05	Suprimindo contatos	2	3	2	2	2	11	**CANDIDATO**
		TC06	Suprimindo lista de contatos	2	3	2	2	2	11	**CANDIDATO**
		TC07	Selecionando remetente	2	2	1	1	2	8	NÃO CANDIDATO
		TC08	Adicionando remetente	3	1	1	1	3	9	**CANDIDATO**
	[UC02] Enviar Web Marketing	TC09	Enviar Web Marketing	3	1	3	3	3	13	**CANDIDATO**
		TC10	Enviar depois de salvo	2	3	2	1	3	11	**CANDIDATO**
		TC11	Enviar um já enviado	2	2	1	1	1	7	NÃO CANDIDATO
		TC12	Tags de substituição no assunto	3	2	1	3	3	12	**CANDIDATO**
		TC13	Importação de URL	2	1	2	2	3	10	**CANDIDATO**
		TC14	Suprimindo contatos	2	2	2	2	3	11	**CANDIDATO**
		TC15	Suprimindo lista de contatos	2	2	2	2	3	11	**CANDIDATO**

01		TC16	Selecionando remetente	2	2	1	1	2	8	NÃO CANDIDATO
		TC17	Adicionando remetente	2	3	2	2	3	12	**CANDIDATO**
		TC18	Editado	3	3	2	2	3	13	**CANDIDATO**
	[UC03] Envio de E-mail de Teste	TC19	Envio com campos preenchidos	3	2	2	2	3	12	**CANDIDATO**
		TC20	Envio com campos de e-mail vazio	2	1	3	1	1	8	NÃO CANDIDATO
		TC21	Envio com tags de substituição no assunto	3	3	3	2	2	13	**CANDIDATO**
		TC22	Importação de URL	3	2	3	2	2	12	**CANDIDATO**
	[UC04] Remover Web Marketing	TC23	Remover salvo	2	1	3	2	3	11	**CANDIDATO**
		TC24	Remover enviado	2	3	2	1	1	9	NÃO CANDIDATO
		TC25	Remover um pendente de envio	2	2	2	1	1	8	NÃO CANDIDATO
	[UC05] Editar Web Marketing	TC26	Editar com campos preenchidos	3	3	3	1	3	13	**CANDIDATO**
		TC27	Editar com campos em branco	2	2	3	1	1	9	NÃO CANDIDATO
		TC28	Editar tags de substituição no assunto	2	1	2	2	3	9	**CANDIDATO**
		TC29	Editar importação de URL	2	1	3	1	3	10	**CANDIDATO**
		TC30	Editar usuário suprimido	2	1	3	1	3	10	**CANDIDATO**
		TC31	Editar lista de contato suprimido	2	1	3	1	3	10	**CANDIDATO**
		TC32	Editar remetente	2	1	2	1	3	9	**CANDIDATO**

	[UC05] Cancelar Envio do Web Marketing	TC33	Status pendente	3	1	2	2	3	11	**CANDIDATO**
		TC34	Status salvo	2	2	2	1	1	8	NÃO CANDIDATO
		TC35	Status enviado	2	2	2	1	1	8	NÃO CANDIDATO
02	[UC06] - Salvar Lista Dinâmica	TC36	Selecionar modelos analíticos	2	2	1	3	3	11	**CANDIDATO**
		TC37	Visualização dos dados	3	1	1	3	3	11	**CANDIDATO**
		TC38	Mapeamento correto dos dados	2	2	2	2	3	11	**CANDIDATO**
		TC39	Mapeamento incorreto dos dados	2	2	1	1	2	8	NÃO CANDIDATO
		TC40	Resumo dos dados	2	1	1	3	3	10	**CANDIDATO**
		TC41	Salvar lista	3	3	3	3	3	15	**CANDIDATO**
	[UC07] Visualizar Lista Dinâmica	TC42	Visualizar lista salva	2	2	2	2	3	11	**CANDIDATO**

	[UC08] - Editar Lista Dinâmica	TC43	Campo nome preenchido	2	2	2	2	3	11	**CANDIDATO**
		TC44	Campo nome em branco	2	1	1	1	1	6	NÃO CANDIDATO
	[UC09] – Excluir Lista Dinâmica	TC45	Excluir lista	2	2	2	2	3	11	**CANDIDATO**
	[UC10] – Adicionar Remetente	TC46	Campo e-mail em branco	2	1	3	1	2	9	NÃO CANDIDATO
		TC47	Campo e-mail preenchido	3	2	2	2	3	12	**CANDIDATO**

03	[UC11] - Editar Remetente	TC48	Campo e-mail em branco	2	1	1	1	1	6	NÃO CANDIDATO
		TC49	Campo e-mail preenchido	3	2	2	2	3	12	**CANDIDATO**
	[UC12] - Remover Remetente	TC50	Remover	2	2	3	2	3	12	**CANDIDATO**
	[UC13] – Criar Lista	TC51	Campos preenchidos	3	3	3	2	3	14	**CANDIDATO**
		TC52	Campos em branco	2	1	1	2	2	8	NÃO CANDIDATO
		TC53	Inserindo CSV	2	2	2	2	3	11	**CANDIDATO**
		TC54	Cancelar criação da lista	3	2	1	1	2	9	**CANDIDATO**

04		TC55	Configurando cabeçalho	2	2	2	1	3	10	**CANDIDATO**
		TC56	Visualizar contatos	2	3	2	1	2	10	**CANDIDATO**
		TC57	Mapeamento dos dados	2	2	2	2	3	11	**CANDIDATO**
		TC58	Qualidade dos dados	2	2	2	2	3	11	**CANDIDATO**
		TC59	Resumo da criação	2	2	2	2	2	10	**CANDIDATO**
		TC60	Salvar	3	3	3	3	3	15	**CANDIDATO**
	[UC14] – Editar Lista Estática	TC61	Editar sobrescrevendo lista	2	1	1	2	2	8	**CANDIDATO**
		TC62	Editar adicionando contato na lista	2	1	1	2	2	8	**CANDIDATO**
		TC63	Editar CSV	2	1	1	3	2	9	**CANDIDATO**
	[UC15] – Visualizar Lista	TC64	Visualizar lista	2	2	2	2	2	10	**CANDIDATO**
	[UC16] – Remover Lista	TC65	Remover lista	2	2	3	2	3	12	**CANDIDATO**

	[UC17] – Salvar Survey	TC66	Campos obrigatórios	2	3	3	2	3	13	**CANDIDATO**
		TC67	Cancelar criação	2	1	1	1	2	7	NÃO CANDIDATO
		TC68	Textos das tags de substituição do assunto	2	2	1	1	1	7	NÃO CANDIDATO
		TC69	Adicionar questões	3	3	2	1	2	11	**CANDIDATO**
		TC70	Remover questão	2	2	2	1	2	9	**CANDIDATO**
		TC71	Clone de questões	2	1	2	1	2	8	**CANDIDATO**
		TC72	Questões como campo obrigatório	2	2	2	2	2	10	**CANDIDATO**
		TC73	HTML e URL no cabeçalho da página de resposta	2	3	2	2	3	12	**CANDIDATO**
		TC74	HTML e URL no rodapé da página de resposta	2	3	2	2	3	12	**CANDIDATO**
		TC75	HTML na página de agradecimento	2	3	2	2	3	12	**CANDIDATO**
		TC76	URL de redirecionamento na página de resposta	2	3	2	2	3	12	**CANDIDATO**
		TC77	Layout da página de resposta fluído	2	2	1	2	3	10	**CANDIDATO**
		TC78	Layout da página de resposta paginado	2	2	1	2	3	10	**CANDIDATO**
		TC79	Múltiplas respostas	2	3	3	1	2	11	**CANDIDATO**
		~~TC80~~	~~Selecionar remetente~~	~~2~~	~~3~~	~~2~~	~~1~~	~~2~~	~~10~~	~~**CANDIDATO**~~
		TC81	Inserir remetente	2	3	3	1	3	12	**CANDIDATO**
		TC82	Tags de substituição no assunto de e-mail	2	2	1	2	3	10	**CANDIDATO**
		TC83	Selecionar remetente	2	3	2	1	2	10	**CANDIDATO**
		TC84	Inserir contato suprimido	2	2	1	2	3	10	**CANDIDATO**
		TC85	Inserir lista de contatos suprimidos	2	2	1	2	3	10	**CANDIDATO**
		TC86	Inserir link de resposta	2	2	1	2	3	10	**CANDIDATO**

05		TC87	URL no corpo do survey	2	3	1	2	2	10	**CANDIDATO**
	[UC18] – Enviar Survey	TC88	Cancelar envio	2	2	2	2	3	11	**CANDIDATO**
		TC89	Enviar com questões como campo obrigatório	2	2	1	1	2	8	**CANDIDATO**
		TC90	Enviar com HTML e URL no cabeçalho da página de resposta	2	3	1	2	3	11	**CANDIDATO**
		TC91	Enviar com HTML e URL no rodapé da página de resposta	2	3	1	2	3	11	**CANDIDATO**
		TC92	Enviar com HTML na página de agradecimento	2	3	1	2	3	11	**CANDIDATO**
		TC93	Enviar com URL de redirecionamento na página de resposta	2	3	1	2	3	11	**CANDIDATO**
		TC94	Enviar com layout da página de resposta fluído	2	2	1	1	3	9	**CANDIDATO**
		TC95	Enviar com layout da página de resposta paginado	2	2	1	1	3	9	**CANDIDATO**
		TC96	Enviar com questões de múltiplas respostas	2	2	1	1	3	9	**CANDIDATO**
		TC97	Enviar com remetente	2	2	1	2	3	10	**CANDIDATO**
		TC98	Enviar com tags de Substituição no assunto de e-mail	2	2	2	2	3	11	**CANDIDATO**
		TC99	Enviar com contato Suprimido	2	2	2	1	2	9	**CANDIDATO**
		TC100	Enviar com lista de Contatos Suprimidos	2	2	2	1	2	9	**CANDIDATO**
		TC101	Enviar com link de Resposta	2	2	2	1	3	10	**CANDIDATO**

		TC102	Enviar com URL no corpo do survey	2	3	1	1	1	8	NÃO CANDIDATO
		TC103	Enviar e-mail de teste	3	3	2	2	3	13	**CANDIDATO**
		TC104	Enviar depois de salvo	3	3	2	2	3	13	**CANDIDATO**
		TC105	Enviar depois de editado	3	3	2	2	3	13	**CANDIDATO**
	[UC19] – Envio e-mail de teste	TC106	Envio de e-mail de teste	2	3	2	2	3	12	**CANDIDATO**
		TC107	Campo de e-mail de teste em branco	2	2	1	1	1	7	NÃO CANDIDATO
	[UC20] – Suprimir lista	TC108	Campo lista suprimida em branco	2	2	1	1	1	7	NÃO CANDIDATO
		TC109	Campo lista suprimida preenchido	3	3	2	1	3	12	**CANDIDATO**
	[UC21] – Suprimir Contatos	TC110	Campo contatos suprimido em branco	2	2	1	1	1	7	NÃO CANDIDATO
		TC111	Campo contato suprimido preenchido	3	3	1	2	3	12	**CANDIDATO**

	[UC21] – Excluir Survey	TC112	Remover salvo	3	2	2	3	3	13	**CANDIDATO**
		TC113	Remover enviado	2	2	2	2	3	11	**CANDIDATO**
		TC114	Remover um pendente de envio	2	3	1	2	3	11	**CANDIDATO**

Fonte - Elaborada pelo autor

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

5.6 Desenvolvimento dos Casos de Testes

Os casos de testes foram construídos em linguagem Java, com uso do Selenium WebDriver. A captura dos objetos para execução dos casos de teste, baseia-se nos métodos disponíveis do Selenium WebDriver. Para verificação dos resultados, utilizaram-se os métodos assertivos do JUnit. A seguir, é apresentado na figura 10, um *script* de testes utilizando esta abordagem.

Figura 10 - Scripts de teste para adicionar remetente

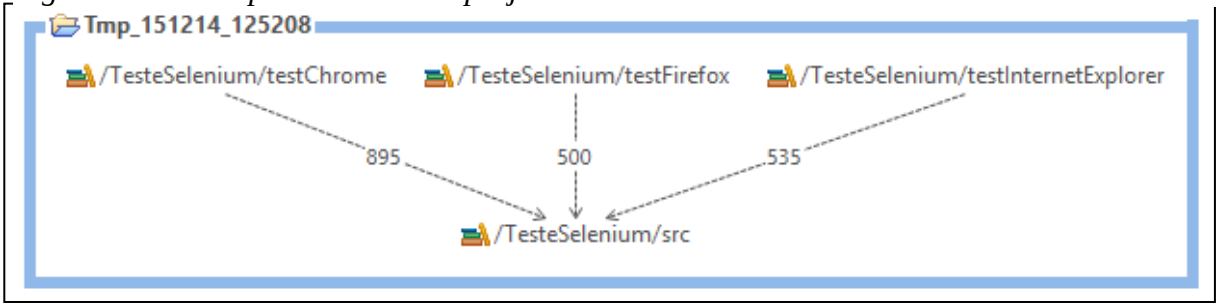
```
31      @Before
32      public void inicializar() {
33          login = new Login();
34          driverIE = new DriverIE();
35          driver = driverIE.getDriver();
36          paginaAdicaoRemetente = new PaginaDeAdicaoDeRemetente();
37          adicionarRemetente = new AdicionarRemetente();
38          excluirRemetente = new ExcluirRemetente();
39          wait = new WebDriverWait(driver, 100);
40
41      }
42
43      @Test
44      public void adicionarRemetenteTest() throws InterruptedException {
45          // Passos:
46          // 1. Fazer Login
47          login.login(driver);
48          // 2. Ir a página de Adição de Remetente
49          paginaAdicaoRemetente.paginadeAdicicaoDeRemetente(driver, wait);
50          // 3: Inserir informação do Usuário
51          idUserAdicionado = adicionarRemetente.adicionarRemetente(driver, wait);
52          // 4: Verifica se o elemento criado esta na lista
53
54          List<WebElement> ListRemetentes = driver.findElements(By.tagName("td"));
55          boolean remetenteInserido = false;
56          for (WebElement remetenteItem : ListRemetentes) {
57              if (remetenteItem.getText().equals(
58                      "teste selenium" + idUserAdicionado)) {
59                  remetenteInserido = true;
60              }
61          }
62          assertTrue(remetenteInserido);
63      }
64
65      @After
66      public void finalizar() {
67          excluirRemetente.excluirRemetenteID(idUserAdicionado, driver, wait);
68          driver.close();
69      }
70
71  }
72
```

Fonte - Elaborado pelo autor

O método *inicializar*, seguido da anotação *Before* na linha 31, inicializa todos os objetos, classes e drivers usados pelo método de execução dos passos de teste. No método *adicionarRemetenteTest*, constam os passos de execução dos testes e os métodos *asserts* (linha 62) e, por fim, o método *finalizar*, que permite a realização de operações no final da execução dos testes. No exemplo, o remetente inserido no método de teste é removido logo no final para manter a base de dados limpa.

As classes de testes do projeto foram desenvolvidas com apoio do ambiente de desenvolvimento integrado Eclipse. A seguir são apresentados os principais módulos do projeto e suas dependências, na Figura 11.

Figura 11 - Principais modelos do projeto.



Fonte - Elaborada pelo autor

O módulo src é composto pelas *classes page objects* que representam as páginas web do sistema e são responsáveis pela execução dos passos do caso de teste, desde a navegação entre páginas e inserção de dados, até a verificação dos resultados. Já os outros módulos representam os passos de execução dos casos de teste e a entrada de dados para os casos de teste, além de instanciação dos respectivos navegadores para teste. Na Figura 12, é apresentado um fragmento de uma classe de acesso aos componentes da página, com auxílio do Selenium pertencentes ao pacote src. Nele, podemos ver um método *inserirCSV*, que é invocado pelos passos de testes presentes nos módulos testeChrome, testeForefox e testeIntenetExplorer. Este recebe como parâmetro um *webDriverWait* que é objeto do Selenium para configuração de pausas na execução dos *scripts*, ele é utilizado para "esperar" os elementos da página serem carregados antes de serem acessados por outros métodos. O outro parâmetro é caminhoCSV que é um caminho de um arquivo a ser carregado no método. O método *drive.findElement*, particular do Selenium, captura o componente da interface. O método sendKeys insere o valor repassado pelo método que o chama para o componente capturado.

Figura 12 - Classe de teste.

```
155
156     public void inserirCSV(WebDriver driver, WebDriverWait wait,
157             String caminhoCSV) throws InterruptedException {
158         // csv
159         driver.findElement(By.name("csvFile")).sendKeys(caminhoCSV);
160         // próximo
161         WebElement botaoNext = driver.findElement(By
162                 .xpath(".//*[@id='contactListAdded']/div/div/ul/li[3]/button"));
163         botaoNext.click();
164         Thread.sleep(5000);
165
166     }
```

Fonte - Elaborado pelo autor

5.7 Execução dos Testes

Após a criação dos *scripts* de testes, foi iniciada a execução dos *scripts* de testes. Eles foram executados nos navegadores Google Chrome, Mozila Firefox e Internet Explorer. A execução dos testes automatizados ocorreu no decorrer das 05 *sprint*s.

Na primeira *sprint*, onde os únicos casos de testes construídos referiam-se ao módulo de Web Marketing, foram executados 25 casos de teste automatizados, comportados em 13 classes de teste, ou seja, em algumas classes de teste é executado mais de um caso de teste, este agrupamento ocorreu devido à similaridade dos testes selecionados. Estes constituem uma suíte de teste do módulo Web Marketing, exibido na Figura 13.

Figura 13 - Suíte de testes do modelo Web Marketing.

```
21
22 @RunWith(Suite.class)
23 @SuiteClasses({
24     TestUC01AddWebMarketing.class,
25     TestUC02SalvarWebMarketing.class,
26     TestUC03TagsSubstituicaoAssunto.class,
27     TestUC05EmailTest.class,
28     TestUC06ImporteURL.class,
29     TestUC08RemoverWebMarketing.class,
30     TestUC09EnviarDespoisDeSalvoWebMarketing.class,
31     TestUC10ContatoSuprimidoWebMarketing.class,
32     TestUC11ListaSuprimidaWebMarketing.class,
33     TestUC12AdicionandoRemetenteWebMarketing.class,
34     TestUC13EnviarWebMarketing.class,
35     TestUC14EditarWebMarketing.class,
36     TestUC15CancelarEnvioWebMarketing.class
37     })
38 public class SuiteTestesWebMarketingChrome {
39
40 }
41
```

Fonte - Elaborada pelo autor

Na segunda *sprint*, os casos de testes executados são referentes ao módulo do sistema Lista Dinâmica. Foram codificados 8 casos de teste automatizados constituindo 8 classes de teste. A seguir, é apresentado na Figura 14, a classe de suíte de teste referente ao módulo Lista Dinâmica executado no navegador Google Chrome.

Figura 14 - Suíte de teste do módulo lista dinâmica.

```
@RunWith(Suite.class)
@Suite.SuiteClasses({

    //chrome
    TestUC01SelecaoDeModelosAnaliticos.class,
    TestUC02ChecarAVisualizadoDosDados.class,
    TestUC03MapeamentoDeDados.class,
    TestUC04VerificarDadosResumo.class,
    TestUC05CriarLista.class,
    TestUC06VisualizarDadosDaLista.class,
    TestUC07ExcluirLista.class,
    TestUC08EditarLista.class

})
public class TestListDynamicChrome {

}
```

Fonte - Elaborada pelo autor

Na terceira *Sprint*, foram executados os casos de testes referentes aos módulos do Remetente. Foram codificados 3 casos de teste em 3 classes de teste. Abaixo, na Figura 15, é possível visualizar a suíte de teste do navegador Google Chrome.

Figura 15 - Suíte de testes do módulo remetente.

```
@RunWith(Suite.class)
@Suite.SuiteClasses({

        // Chrme
        TestUC01AdicionarRemetente.class,
        TestUC03EditarRemetente.class,
        TestUC04RemoverRemetente.class

        })
public class TestRemetenteChrome {

}

```

Figura - Elaborada pelo autor

Na *sprint* quatro, onde os casos de testes executados foram os referentes ao módulo de Lista Estática, foram codificados um total de 14 casos de teste candidatos a automação, resultando em 13 classes de teste, tendo a classe *criarListaEstatica* mais de um teste. A seguir, é apresentado, na Figura 16, a suíte de teste do módulo no navegador Google Chrome.

Figura 16 - Suíte de testes do módulo lista estática.

```
@RunWith(Suite.class)
@SuiteClasses({
    //Casos de Teste Chromer
    UC01CriarListaEstaticaChrome.class,
    UC02CamposObrigatoriosPaginaAddLista.class,
    UC03CancelarCriacaoListaEstaticaPaginaAdicionarLista.class,
    UC04CriarListaSemCabecalho.class,
    UC05VisualizarContatosNaCriacao.class,
    UC06MapearContatosDoMesmoTipo.class,
    UC07QualidadeDosDadosDuplicadoEmBranco.class,
    UC08SalvarListaEstica.class,
    UC09VerificarConteudoTelaResumo.class,
    UC10RemoverLista.class,
    UC11VisualizarLista.class,
    UC12EditarListaContatoSobrescrevendoLista.class,
    UC13EditarListaContatoAdicionandoContato.class,
    })
public class TestsListStatic {

}
```

Fonte - Elaborada pelo autor.

Na última *sprint* analisada por este trabalho, os testes executados foram dos componentes referentes a Survey, foram codificadas 43 classes de teste contemplando os 42 cenários candidatos a automação. A seguir, é apresentada, na Figura 17, a suíte de teste do módulo Survey no navegador Google Chrome.

As suítes de testes apresentadas até o momento são referentes aos módulos dos sistemas contemplados pelas *sprints* e implementadas durante a construção deste trabalho. Foram construídas suítes de testes dos navegadores Google Chrome, Mozila Firefox e Intenet Explorer, para facilitar a execução de uma determinada suíte de teste em um navegador específico.

Figura 17 - Suíte de testes do módulo survey.

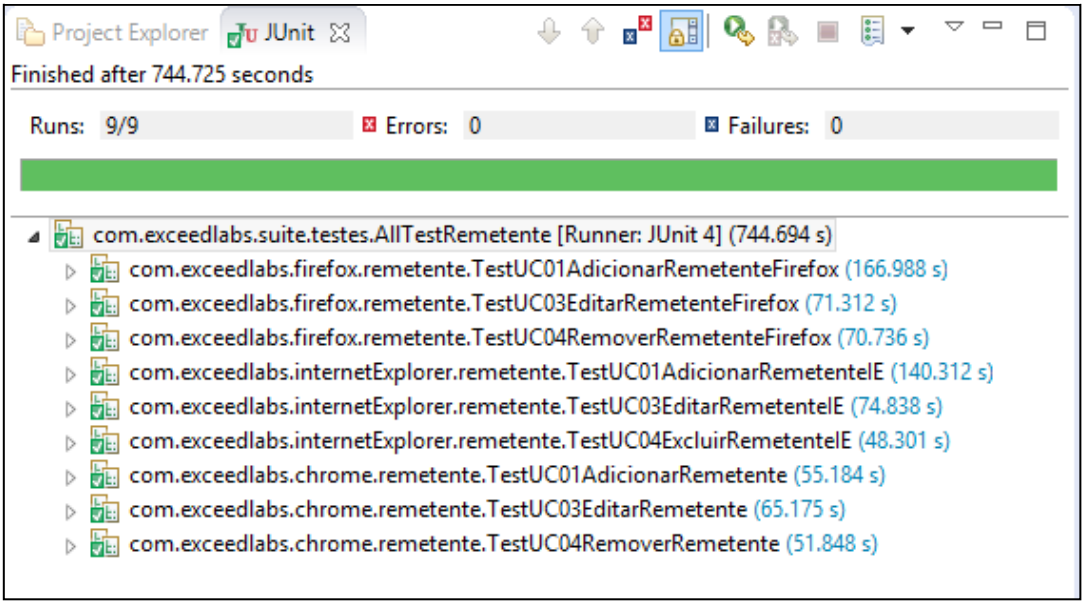
```
@RunWith(Suite.class)
@SuiteClasses({
    UC01SurveyCamposObrigatoriosPaginaConfiguracao.class,
    UC06AdicionarQuestaoMultiplaEscolha.class,
    UC07AdicionarQuestaoDropDown.class,
    UC08AdicionarQuestao5InputFields.class,
    UC09AdicionarQuestaoRating.class,
    UC10AdicionarQuestãoTrueFalse.class,
    UC11AdicionarQuestaoDatepickerField.class,
    UC12AdicionarQuestaoTimePecker.class,
    UC13AdcionarQuestaoComment.class,
    UC14AdicionarQuestaoOneChoice.class,
    UC15RemoverQuestao.class,
    UC16AdicionarQuestao.class,
    UC17ClonarQuestao.class,
    UC18VerificarQuestaoComoCampoObrigatorio.class,
    UC19InserirHTMLnoCabecalho.class,
    UC20InserirURLnoCabecalho.class,
    UC21InserirHTMLnoRodape.class,
    UC22InserirURLnoRodape.class,
    UC23InserirHTMLdeAgradecimento.class,
    UC24InserirURLdeRedirecionamento.class,
    UC25VerificarSurveyLayoutFluido.class,
    UC26VerificarSurveyLayoutPaginado.class,
    UC27VerificarConfiguracaoSurveyMultiplasRespostas.class,
    UC28VerificarConfiguracaoSurveyUnicaResposta.class,
    UC31EscolherRemetente.class,
    UC32AdicionarRemetente.class,
    UC33InserirTagsDeSubstituicaoNoAssunto.class,
    UC34VerificarTextoDasTagsDeSubstituicao.class,
    UC35SelecionarListaDeContato.class,
    UC36InserirConatoSuprimido.class,
    UC37InserirListaConatoSuprimida.class,
    UC38EnvioSurveyComLinkPaginaResposta.class,
    UC39EnviarEmailDeTeste.class,
    UC40EnviarSurveyComImportURL.class,
    UC41SalvarEnviarSurveyDepoisDeSalvo.class,
    UC42EnviarSurveyComContatoSuprimido.class,
    UC43EnviarSurveyComListaSuprimida.class,
    UC44EnviarSurveyComTagsNaAssunto.class,
    UC45EnviarSurveyComLayoutConfiguradoUm.class,
    UC46EnviarSurveyComLayoutConfiguradoDois.class,
    UC47RemoverSurvey.class,
    UC48EditarSurvey.class
})
public class TestSurvey {

}
```

Fonte - Elaborada pelo autor

Além dessas, foi construída uma suíte de teste geral por componente, onde é possível executar os casos de testes de todos os navegadores. Um exemplo disso pode ser visto na Figura 18, onde é apresentada a suíte de teste do módulo Remetente com os casos de testes dos navegadores contemplados. Nela, podemos ver a execução dos casos de teste e seu resultado individual da execução, assim como o tempo em que os casos de testes foram executados e o tempo total de execução da suíte.

Figura 18 - Execução da suíte de teste remetente.



Fonte - Elaborada pelo autor

É importante destacar que a execução das suítes de testes não ocorreu só nas suas respectivas *sprints*. Quando necessária, a execução de testes de regressão de todas as suítes de testes implementadas até o momento foram executadas, assim como nas atividades da estratégia de automação dos testes oriundo deste trabalho. Na capitulo 6, é feita uma análise dos resultados sobre essas execuções e, a seguir, é apresentado, na Tabela 11, o número de vezes em que as suítes de testes automatizados foram executadas, seu tempo de execução e número de bugs detectados. Essas *suítes* executadas constam de casos de testes automatizados dos navegadores Internet Explorer, Mozila Firefox e Google Chrome.

Tabela 11 - Número de execução das suítes

Sprint	Nº de execuções	Tempo de execução	Nº de Bugs detectados
1	6	12hs	23
2	4	10hs	15
3	4	8hs	10
4	5	8hs	11
5	4	13hs	25
Total	**23**	**51hs**	**84**

Fonte - Elaborada pelo autor

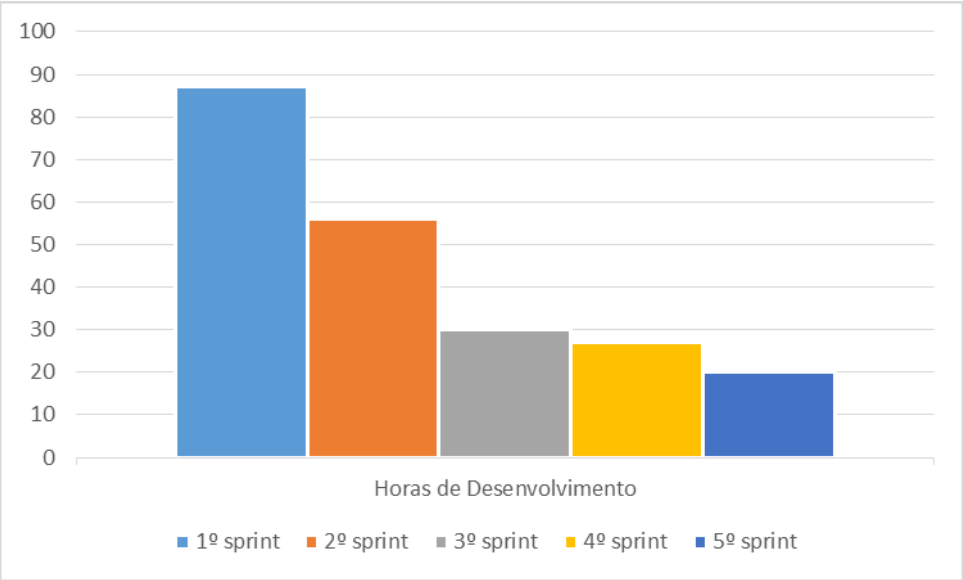
6 ANÁLISE DOS RESULTADOS

6.1 Análise quantitativa

A análise dos resultados do teste foi feita a partir dos resultados registrados no JUnit. A partir da ferramenta, foi possível visualizar a execução das suítes de testes, mostrando o resultado para cada caso de teste e sumarizando os resultados para cada suíte. Essa visualização fornece ao testador um retorno geral sobre a quantidade de bugs detectados no 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.

Com a execução da atividade "implementação", foi possível perceber um ganho expressivo em produtividade na criação dos *scripts*, após a criação dos *page objects*, responsáveis pelo acesso às páginas web, permitindo a criação dos casos de testes com JUnit demandasse apenas a chamada dos métodos criados com novos valores de entrada. Na Figura 19, onde é mostrado um gráfico de representação das horas de construção dos *scripts* pelas *sprint*s é possível perceber o declínio das horas utilizadas para a implementação. Porém esse resultado não cabe somente à implantação do padrão *page objects*, é importante frisar que, ao longo das *sprints*, o analista de teste foi ganhando mais experiência na atividade desempenhada, logo diminuiu seu tempo de implementação.

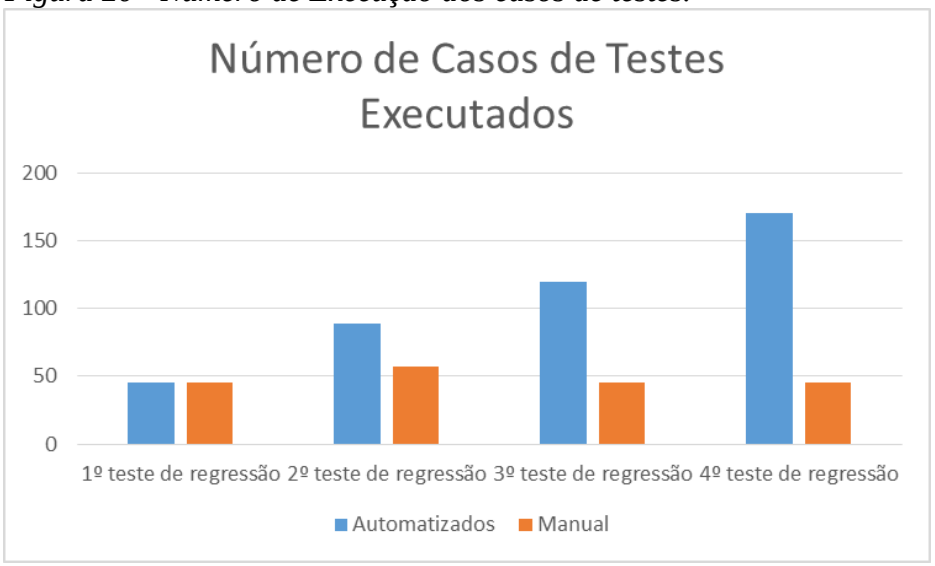
Figura 19 – Horas de construção dos scripts de testes.



Fonte - Elaborada pelo autor.

A execução automatizada permitiu que os testes fossem executados em todos os navegadores requisitados pela especificação do sistema, apenas alterando a chamada dos *webdrivers*. Ao todo foram automatizados 279 casos de testes. Podemos observar os resultados na cobertura dos testes na Figura 20.

Figura 20 - Número de Execução dos casos de testes.



Fonte - Elaborada pelo autor

Os casos de testes automatizados foram executados em todos os navegadores. Podemos observar que o ganho na cobertura dos testes justifica a adoção da automação de testes no projeto, pois conforme o sistema for evoluindo, irá aumentar também o número de cenários de teste a serem verificados, e, consequentemente, o esforço na sua execução irá aumentar. Essa característica reforça ainda mais a adoção de testes automatizados.

Após a execução dos testes, é necessário verificar os casos de teste que não foram executados com sucesso. Essa verificação ocorre com a ajuda das falhas exibidos pelo JUnit durante e no final da execução dos casos de testes automatizados. Quando possível, os erros eram identificados e corrigidos, e os testes eram 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 mostraram-se atividades importantes, 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. Além de atualização dos componentes da interface, a atualização dos navegadores impedia a execução dos casos de testes, tendo a necessidade de atualizar os *scripts* para serem executados conforme a atualização dos navegadores.

Após a execução de cada suíte de teste ou conjunto de suítes de testes, foi possível gerar relatórios e métricas sobre a execução dos testes, através dos resultados exibidos pelo Junit. 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 uma execução de todas as suítes de teste em um único navegador em uma versão do sistema e que pode ser visto na tabela 12 com mais detalhes.

Os casos de testes aprovados são aqueles que não apresentaram *bugs* na sua execução. Já os reprovados, são casos de testes que, na sua execução, apresentaram falhas. Estes foram reportados para a equipe de desenvolvimento e todos foram devidamente corrigidos. Através dessa matriz, é possível ter uma visão global dos resultados de execução dos testes no sistema, permitindo acompanhar o número de bugs do sistema e, caso necessário, dimensionar ações de melhoria dentro do projeto para funcionalidades em específico.

Tabela 12- Matriz de execução das suítes de teste.

Caso de Teste	Componente	Resultado
TC01 Campos preenchidos	Web Marketing	**Aprovado**
TC03 Tags de substituição no assunto	Web Marketing	**Aprovado**
TC04 Importe de URL	Web Marketing	**Reprovado**
TC05 Suprimindo contatos	Web Marketing	**Aprovado**
TC06 Suprimindo lista de contatos	Web Marketing	**Aprovado**
TC08 Adicionando remetente	Web Marketing	**Aprovado**
TC09 Enviar	Web Marketing	**Aprovado**
TC10 Enviar depois de salvo	Web Marketing	**Aprovado**
TC12 Tags de substituição no assunto	Web Marketing	**Reprovado**
TC13 Importação de URL	Web Marketing	**Aprovado**
TC14 Suprimindo contatos	Web Marketing	**Aprovado**
TC15 Suprimindo lista de contatos	Web Marketing	**Aprovado**
TC17 Adicionando remetente	Web Marketing	**Aprovado**
TC18 Editado	Web Marketing	**Aprovado**
TC19 Envio com campos preenchidos	Web Marketing	**Aprovado**
TC21 Envio com tags de substituição no assunto	Web Marketing	**Aprovado**
TC22 Importação de URL	Web Marketing	**Reprovado**
TC23 Remover salvo	Web Marketing	**Reprovado**

TC26 Editar com campos preenchidos	Web Marketing	**Aprovado**
TC28 Editar tags de substituição no assunto	Web Marketing	**Aprovado**
TC29 Editar importação de URL	Web Marketing	**Aprovado**
TC30 Editar usuário suprimido	Web Marketing	**Aprovado**
TC31 Editar lista de contato suprimido	Web Marketing	**Aprovado**
TC32 Editar remetente	Web Marketing	**Reprovado**
TC33 Status pendente	Web Marketing	**Aprovado**
TC36 Selecionar modelos analíticos	M. Analítico	**Reprovado**
TC37 Visualização dos dados	M. Analítico	**Reprovado**
TC38 Mapeamento correto dos dados	M. Analítico	**Aprovado**
TC40 Resumo dos dados	M. Analítico	**Reprovado**
TC41 Salvar lista	M. Analítico	**Reprovado**
TC42 Visualizar lista salva	M. Analítico	**Aprovado**
TC43 Campo nome preenchido	M. Analítico	**Aprovado**
TC45 Excluir lista	M. Analítico	**Aprovado**
TC47 Campo e-mail preenchido	Remetente	**Aprovado**
TC49 Campo e-mail preenchido	Remetente	**Aprovado**
TC50 Remover	Remetente	**Aprovado**
TC51 Campos preenchidos	Lista Estática	**Aprovado**
TC53 Inserindo CSV	Lista Estática	**Reprovado**
TC54 Cancelar criação da lista	Lista Estática	**Aprovado**
TC55 Configurando cabeçalho	Lista Estática	**Reprovado**
TC56 Visualizar contatos	Lista Estática	**Reprovado**
TC57 Mapeamento dos dados	Lista Estática	**Aprovado**
TC58 Qualidade dos dados	Lista Estática	**Reprovado**
TC59 Resumo da criação	Lista Estática	**Aprovado**
TC61 Editar sobrescrevendo lista	Lista Estática	**Aprovado**
TC62 Editar adicionando contato na lista	Lista Estática	**Aprovado**
TC63 Editar CSV	Lista Estática	**Aprovado**
TC64 Visualizar lista	Lista Estática	**Reprovado**
TC65 Remover lista	Lista Estática	**Aprovado**
TC66 Campos obrigatórios	Survey	**Aprovado**
TC69 Adicionar questões	Survey	**Reprovado**

TC70 Remover questão	Survey	**Aprovado**
TC71 Clone de questões	Survey	**Aprovado**
TC72 Questões como campo obrigatório	Survey	**Reprovado**
TC73 HTML e URL no cabeçalho da página de resposta	Survey	**Aprovado**
TC74 HTML e URL no rodapé da página de resposta	Survey	**Aprovado**
TC75 HTML na página de agradecimento	Survey	**Aprovado**
TC76 URL de redirecionamento na página de resposta	Survey	**Aprovado**
TC77 Layout da página de resposta fluído	Survey	**Aprovado**
TC78 Layout da página de resposta paginado	Survey	**Aprovado**
TC79 Múltiplas respostas	Survey	**Aprovado**
TC81 Inserir remetente	Survey	**Aprovado**
TC82 Tags de substituição no assunto de e-mail	Survey	**Reprovado**
TC83 Selecionar remetente	Survey	**Aprovado**
TC84 Inserir contato suprimido	Survey	**Aprovado**
TC85 Inserir lista de contatos suprimidos	Survey	**Aprovado**
TC86 Inserir link de resposta	Survey	**Reprovado**
TC87 URL no corpo do survey	Survey	**Aprovado**
TC88 Cancelar envio	Survey	**Aprovado**
TC89 Enviar com questões como campo obrigatório	Survey	**Reprovado**
TC90 Enviar com HTML e URL no cabeçalho da página de resposta	Survey	**Aprovado**
TC91 Enviar com HTML e URL no rodapé da página de resposta	Survey	**Aprovado**
TC92 Enviar com HTML na página de agradecimento	Survey	**Aprovado**
TC93 Enviar com URL de redirecionamento na página de resposta	Survey	**Aprovado**
TC94 Enviar com layout da página de resposta fluído	Survey	**Aprovado**

TC95 Enviar com layout da página de resposta paginado	Survey	**Aprovado**
TC96 Enviar com questões de múltiplas respostas	Survey	**Aprovado**
TC97 Enviar com remetente	Survey	**Aprovado**
TC98 Enviar com tags de Substituição no assunto de e-mail	Survey	**Reprovado**
TC99 Enviar com contato Suprimido	Survey	**Aprovado**
TC100 Enviar com lista de Contatos Suprimido	Survey	**Aprovado**
TC101 Enviar com link de Resposta	Survey	**Aprovado**
TC103 Enviar e-mail de teste	Survey	**Reprovado**
TC104 Enviar depois de salvo	Survey	**Aprovado**
TC105 Enviar depois de editado	Survey	**Aprovado**
TC106 Envio de e-mail de teste	Survey	**Aprovado**
TC109 Campo lista suprimida preenchido	Survey	**Aprovado**
TC110 Campo contatos suprimido em branco	Survey	**Aprovado**
TC111 Campo contato suprimido preenchido	Survey	**Aprovado**
TC112 Remover salvo	Survey	**Aprovado**
TC113 Remover enviado	Survey	**Aprovado**
TC114 Remover um pendente de envio	Survey	**Aprovado**

Fonte - Elaborada pelo autor

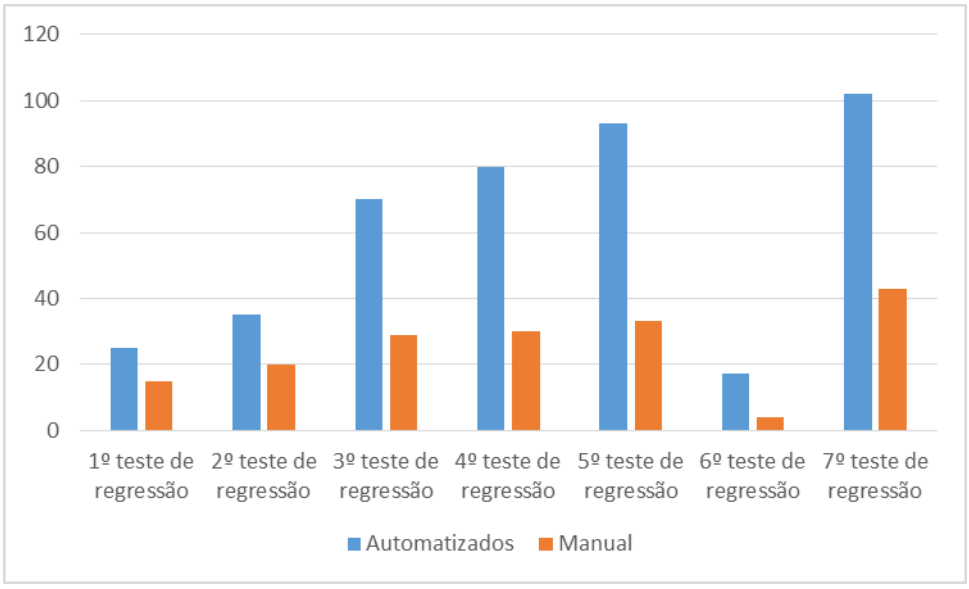
A eficácia dos testes automatizados em descobrir as falhas no sistema também apresentou resultados positivos. A maior parte das falhas no sistema relacionadas às verificações previstas nos *scripts* de testes, e identificadas na execução manual, também foram capturadas na execução automática. É importante ressaltar que a execução manual, conseguiu identificar problemas no sistema que não foram cobertos pelos casos de testes automatizados, como, por exemplo, problemas de deslocamento de elementos na interface, quebra de *layout*, seleção de vários componentes de maneira exploratória e navegação no sistema através do teclado. 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 por seu cenário possuir obrigatoriedade de interação com o usuário.

A Figura 21 apresenta um comparativo entre o número de falhas encontradas com a execução dos testes manuais e automatizados ao longo de 7 testes de regressão que

aconteceram ao longo das *sprints*. Analisando o gráfico, é possível perceber um aumento do número de falhas encontradas com a inclusão de testes automatizados durante os testes. 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 testes de regressão onde os testes manuais não podem ser totalmente executados.

Sobre o controle dos ambientes de desenvolvimento com execução dos casos de testes automatizados, durante suas atualizações e implantação do ambiente *alfa* a quantidade de bugs identificados no ambiente de produção diminuiu em 97%. Os 3% dos bugs identificados no ambiente de produção são de cenários não levantados pelo testador no momento da criação dos cenários de testes. Sobre a sugestão do controle dos ambientes podemos analisar de forma satisfatória sua implementação, pois o número de bugs pertencentes a versões do sistema entregues aos usuários finais foi reduzida dramaticamente.

Figura 21 - Comparativo entre o número de falhas encontradas com a execução dos testes manuais e automatizados.



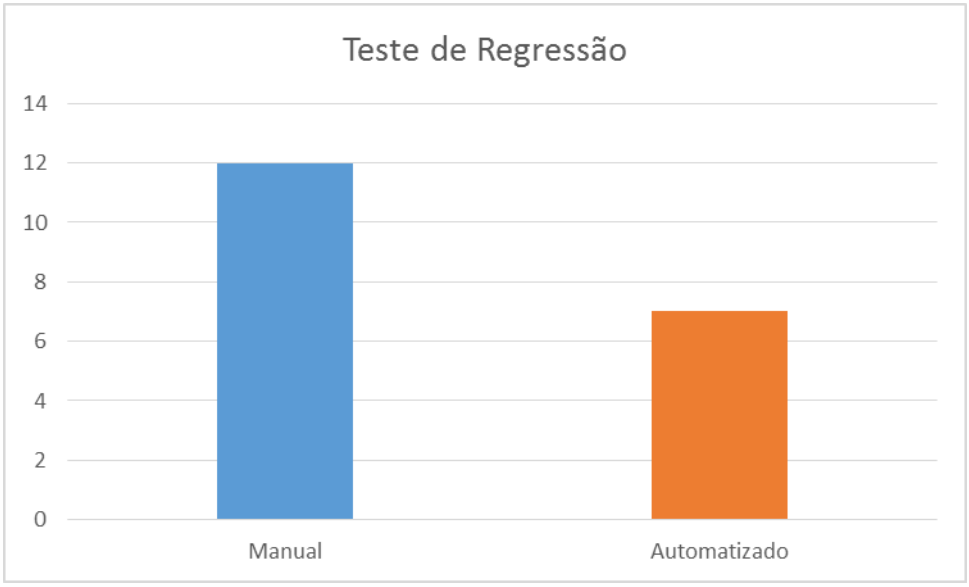
Fonte – Elaborado pelo autor.

Ao comparar a eficiência dos testes manuais em relação aos testes automatizados, foi possível notar que os testes automatizados são mais eficientes para os casos de teste que verificaram um maior número de entradas, com regras de validação mais complexas ou casos de testes com passos repetitivos. Já para cenários desse tipo, a execução manual, possui um maior esforço, além de ser propensos a erros humanos, já que as regras são verificadas de maneira visual. Porém 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.

Sobre o tempo de execução dos testes automatizados, o analista de testes considerou os testes automatizados mais rápidos que os testes manuais, ao executar um teste de regressão. Na execução do último teste de regressão analisado por este trabalho, podemos analisar seu resultado na Figura 22 da execução de todas as suítes de teste no navegador Google Chrome.

Figura 22 - Número de horas gastas na execução de um teste de regressão com casos de testes manuais e automatizados.



Fonte - Elaborado pelo autor

Sobre a diminuição do tempo de execução dos testes de regressão com a estratégia de automação, podemos inferir que a economia de 5 horas durante cada execução causa uma economia de R$ 65,00, levando em consideração a hora de um testador na empresa em estudo.

A figura 22 apresenta um comparativo em números de horas gastas na execução de um teste de regressão com somente testes manuais e execução de testes automatizados em conjunto com os testes manuais.

Podemos concluir com as análises apresentadas que a cobertura dos testes aumentou e que os testes de regressão tiveram seu tempo diminuído com a implantação da estratégia de automação, além de aumentar a confiança do sistema e da equipe com a redução da quantidade de bugs disponibilizados aos usuários finais do sistema com a implantação do ambiente de controle *Alfa*.

6.2 Análise qualitativamente

Para completar nossa análise da implantação das atividades de automação de testes do processo, assim como, seu impacto no dia a dia da equipe, realizou-se uma pesquisa com quatro desenvolvedores, um líder técnico e um gerente.

As seguintes perguntas foram realizadas:

1. Nome e Cargo do entrevistado.
2. Como era os testes antes da implantação da estratégia de automação testes?
3. Como são os testes hoje?
4. É possível perceber melhoria provenientes da implantação da estratégia de automação dos testes? Quais?
5. Cite pontos negativos da estratégia de automação dos testes.
6. Cite pontos de possíveis melhoria da estratégia de automação dos testes.

No final da pesquisa, foi possível perceber que todos os entrevistados possuíam conhecimento de como as atividades de testes manuais aconteciam ao longo do desenvolvimento do software. Quando perguntados como era os testes antes da implantação da estratégia de automação testes? Obtivemos as seguintes respostas:

- “Execução dos cenários de testes no ambiente de testes executado pelo testador.”
- “Tínhamos a criação de cenários de testes com base dos requisitos, quando a implementação era disponibilizada para o ambiente de testes, o testador executa os casos de testes manualmente com o objetivo de encontrar bugs”.
- “Toda a execução dos testes era feita de forma manual, incluindo os testes de regressão”.
- “O testador executava seus testes manuais no ambiente de testes, caso existisse bugs ele reportava para a equipe de desenvolvimento”.
- “Execução de testes funcionais, executados pelo testador no ambiente de testes de forma manual”.
- “Quando uma funcionalidade era desenvolvida, ela era submetida para o ambiente de testes. Neste ambiente, o testador executava seus cenários para certificar que tudo está certo”.

Sobre estas respostas, podemos inferir que os membros da equipe detinham conhecimento de que a execução dos casos de testes funcional ocorria de forma manual e era executada somente no ambiente de testes.

Quando perguntados como são os testes hoje? Os membros da equipe responderam das seguintes maneiras:

- “Com a automação de alguns cenários de testes, a execução dos testes tornou-se mais rápida”.
- “Facilidade de execução de cenários de teste pelo testador e um aumento na cobertura dos testes”.
- “Minhas funcionalidades são testadas com mais precisão em todos os ambientes de desenvolvimento”.
- “Os cenários de testes candidatos à automação são automatizados. Quando uma funcionalidade ou correção de um bug é dita como resolvida, ela é testada com os casos de testes automatizados no ambiente de desenvolvimento. Caso os testes não encontrem erro, a funcionalidade é disponibilizada no ambiente de testes onde o testador executa os cenários de testes automatizados e manuais. Se nada for encontrado, essa funcionalidade é disponibilizada no ambiente *alfa* e logo depois pode ser adicionada ao ambiente de produção”.
- “Hoje temos um processo de testes, com atividades de testes manuais e testes automatizados, ambos os testes (manuais e automatizados) são executados de acordo com processo sugerido pelo QA e adotado pela empresa”.
- “Sinto que a execução dos testes de regressão é mais rápida e o controle dos ambientes de desenvolvimento tornou-se essencial para a disponibilização de um ambiente controlado para os clientes”.

Todos os entrevistados relataram melhorias provenientes da implantação de atividades de automação no processo, algumas delas são: execução dos testes de regressão mais rápidos, confiabilidade na execução dos testes com precisão da automação, maior cobertura do sistema por parte dos testes e controle dos ambientes de desenvolvimento com a execução das suítes de testes automatizados.

Quando perguntados sobre os pontos negativos da automação, nenhum dos entrevistados soube opinar. Acreditamos que a ausência dessa informação ocorre por os entrevistados não estarem imersos na execução das atividades de automação de testes, só observando, na maioria dos casos, os artefatos gerados pelas atividades.

Já quando questionados sobre possíveis melhorias na estratégia de automação dos testes, os entrevistados levantaram os seguintes pontos: integração da execução dos casos de testes com ferramentas de gerenciamento para melhorar o entendimento e compartilhamento

dos resultados, otimização dos tempos de execução dos testes automatizados e extensão da automação dos casos de testes para os casos de testes hoje não candidatos à automação.

7 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho apresentou um relato de experiência na implantação de testes funcionais automatizados em uma empresa de TI em um projeto real, tendo como base o Modelo de Melhoria do Processo de Testes Brasileiro (MPT.Br), mais especificamente a área referente a automação de testes, a Automação da Execução de Testes (AET) para montagem da estratégia de automação de testes. Foram definidas atividades, artefatos, papéis necessários ao desenvolvimento e execução de testes automatizados, tendo como base a ferramenta de automação de testes Selenium WebDriver.

Com a implantação da estratégia de automação dos testes, foi possível perceber os benefícios da automação no processo de construção do sistema. Um dos principais benefícios percebidos foi a redução do tempo de execução dos testes de regressão. Com essa redução do tempo na execução dos testes de regressão, é possível perceber uma redução direta de custo no desenvolvimento e manutenção de softwares, a partir do menor tempo necessário à execução de testes de regressão, dado que estes poderão ser executados de forma automática e não mais totalmente manual.

Outro benefício importante da implantação da estratégia de automação dos testes foi um aumento da cobertura dos testes, pois como os testes automatizados levam menos tempo para serem executados, sobra mais tempo para o testador executar testes exploratórios aumentando ainda a chance de se detectar *bug's*. Além dos benefícios já citados, é importante destacar a diminuição de *bug's* no ambiente de produção com a implantação do ambiente *alfa* e os conceitos utilizados por este trabalho vivenciados na sala de aula aplicados diretamente no mercado de trabalho.

Além dos principais benefícios já citados podemos destacar os seguintes:

- Alteração do processo seguido pela empresa em estudo, na introdução de atividades de desenvolvimento e execução de testes automatizados,
- Melhoria no processo de teste executado pela empresa de TI estudada, diminuindo a quantidade de erros e consequentemente aumentando a confiança do software desenvolvido.

Portanto, podemos concluir que a implantação da estratégia de automação dos testes 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, é possível incorporar todos os pontos levantados na pesquisa realizada com os membros da equipe quando questionados sobre possíveis pontos de melhoria, a possibilidade de extensão dos casos de teste automatizados para outras funcionalidades do sistema e dos casos de testes não selecionados pelos critérios de seleção. Otimização do tempo de execução dos casos de testes, podendo ser otimizado pela execução dos testes de regressão em paralelo. Diante disto, planeja-se descobrir casos de dependência e independência entre testes, para executar testes independentes em paralelo. Espera-se que a oferta de máquinas com múltiplos núcleos, comuns hoje em dia, possa acelerar muito o tempo para observação de erros e assim melhorar esta prática que é fundamental para diminuir o tempo de execução dos testes de regressão. Por fim, a integração da execução dos casos de testes com ferramenta de gerenciamento de testes, é possível de ser implementada com a ajuda da ferramenta TestLink[8].

[8] Disponível em http://testlink.org/

REFERÊNCIAS

BARTIÉ, Alexandre. **Garantia de Qualidade de Software: adquirindo maturidade organizacional.** Rio de Janeiro: Elsevier, 2002, 3a Edição, 291p.

BASTOS, Aderson; et al. **Base de conhecimento em teste de software**. São Paulo: Martins, 2007.

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.

BURNESTEIN, I. **Practical Software Testing: A Process-oriented Approach**. 2003. 87 Springer, New York.

CARVALHO, Eliana Lencioni. **Testes de Software e Automação de Testes**. 2010. Disponível em: < http://www.tmtestes.com.br/uploads/conteudo/49/Artigo_1-teste_%20Automacao_SW.pdf> Acesso em: 07 maio 2015.

CERVANTES, Alex. Exploring the use of a test automation framework. In: **Aerospace conference**, 2009 IEEE. IEEE, 2009. p. 1-9.

CHANG, Juei; RICHARDSON, Debra J. **Structural specification-based testing: Automated support and experimental evaluation. In: Software Engineering**—ESEC/FSE'99. Springer Berlin Heidelberg, 1999. p. 285-302.

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

CRAIG, R.D., JASKIEL, S. P. **Systematic Software Testing**, Artech House Publishers, Boston, 2002.

FEWSTER M.; GRAHAM D. **Software Test Automation – Effective use of test execution tools**. ACM Press/Addison-Wesley Publishing Co.,New York, 1999.

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.

FURLAN, Fabiane Pifer. Visualização de Informação como Apoio ao Planejamento do Teste de Software.2009. 140f. Dissertação (Mestrado) - Universidade Metodista de Piracicaba, Piracicaba, SP: 2009.

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

INTHURN, Cândida. **Qualidade & Teste de Software**. Florianópolis: Visual Books Ltda, 2001.

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.** 2014. 59f. Monografia (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Bacharelado em Engenharia de Software, Quixadá, 2014.

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO. Evolução da Qualidade de Software no Brasil de 1994-2010 baseada nas pesquisas e projetos do PBQP Software (Relatório técnico do MCTI). Paraná: UFP, 2012. Disponível em: <http://www.mct.gov.br/upd_blob/0222/222128.pdf>. Acesso em: 31 maio. 2013

MYERS, Glenford J. The Art of Software Testing. A Willy-Interscience Pub, 1979. 2 ed. Nova Jérsei: John Wiley & Sons, 2004.

MOLINARI, L. **Testes de Software Produzindo Sistemas Melhores e Mais Confiáveis**. 2a Edição. São Paulo-SP. 2005.

MOLINARI, L. **Inovação e Automação de Testes de Software**. 1a Edição. São Paulo-SP. 2010.

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.

PERRY, W. E. **Effective Methods for Software Testing**, 3 ed, Wiley Publishing, Indianapolis, IN, 2006. 591p.

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

PRESMAN, R. S. **Software engineering - a practitioner's approach**. 5th. ed. McGrawHill, 2001.

PRESSMAN, R. S., "**Software Engineering: A Practitioner's Approach**", McGraw-Hill, 6th ed, Nova York, NY, 2005.

PRESSMAN, Roger S. **Engenharia de software**. McGraw Hill Brasil, 2011.

RAPPS, S., WEYUKER, E.J. **Data Flow analysis techniques for test data selection**, In: International Conference on Software Engineering, p. 272-278, Tokio, Sep. 1982.

RIOS, Emerson; MOREIRA FILHO, Trayahú. **Teste de software**. 3 ed. rev. e atual. Rio de Janeiro: Alta Books Editora, 2013. 304p.

ROCHA, A. R. C.; MALDONADO, J. C.; WEBER, K. C. et al. **Qualidade de software** – **Teoria e prática**, Prentice Hall, São Paulo, 2001.

ROTHERMEL, Gregg; ELBAUM, Sebastian; MALISHEVSKY, Alexey; KALLAKURI, Praveen; QIU, Xuemei. **On Test Suite Composition and Cost- 47 Effective Regression Testing**, Agosto de 2003

SAPNA, P. G.; MOHANTY, Hrushikesha. **Clustering test cases to achieve effective test selection**. In: Proceedings of the 1st Amrita ACM-W Celebration on Women in Computing in India. ACM, 2010. p. 15.

SANTOS, I. S.; SANTOS NETO, P. **Automação de Testes Funcionais com o Selenium**, Escola Regional de Computação (Ceará, Maranhão e Piauí). 2009. p. 27-52.

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: 10 jan. 2014.

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

SOMMERVILLE, Ian. **Engenharia de Software**, 9 Edição - São Paulo: Person Prenticial Hall, 2011.

SeleniumHQ. **Selenium Documentation**. (2012) Disponível em: http://seleniumhq.org/docs/. Acesso em jun. 2012.

TUSCHLING, O. **Software Test Automation**, 2008. Disponível em: http://www.stickyminds.com/getfile.asp?ot=XML&id=14908&fn=XDD14908filelistfilename1%2Epdf Acesso em: 09 mar. 2015.

THIOLLENT, Michel. **Metodologia da pesquisa-ação**. São Paulo: Cortez: Autores Associados, 1986. 108p

VILAS BOAS, A. L. de C. **Qualidade e Avaliação de Produto de Software**. Ed. Lavras/MG: FAEPE/UFLA, 2007.