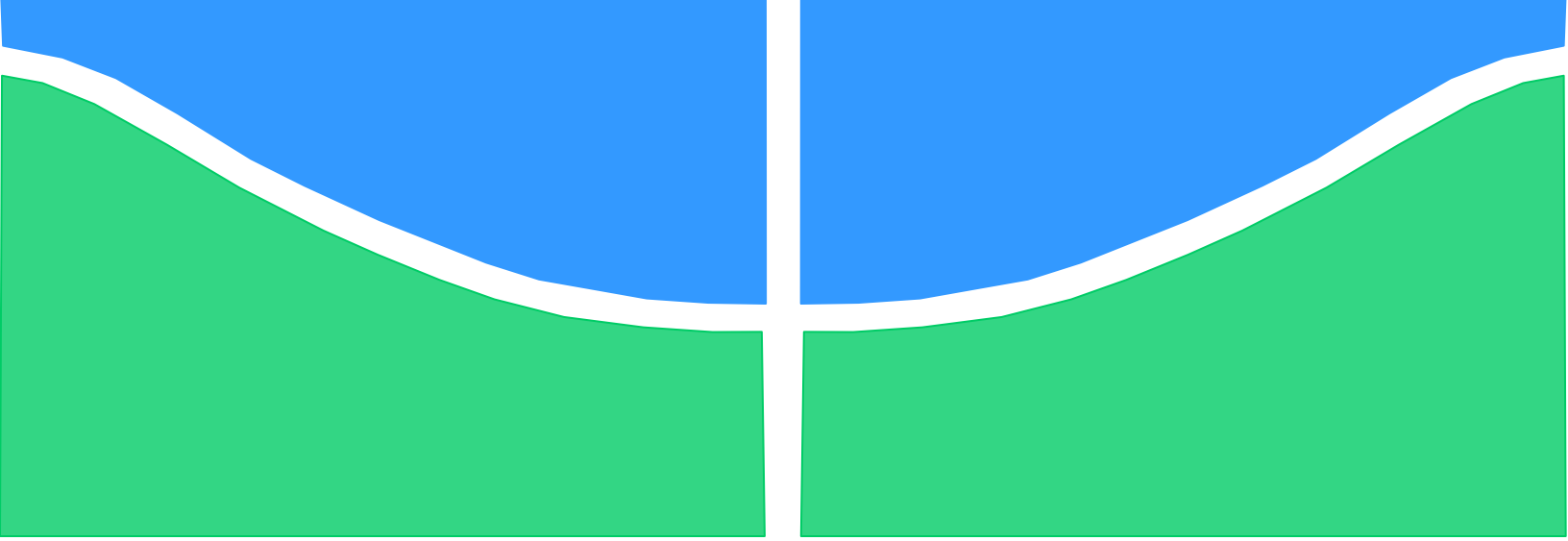


Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Curso de Engenharia de Software

Estratégias para realizar testes funcionais de interface com o usuário – visão de uma equipe de testes

Autor: João Guilherme Santana Araruna
Orientador (a): Dra. Carla Silva Rocha Aguiar

Brasília, DF
2017

Estratégias para realizar testes funcionais de interface com o usuário – visão de uma equipe de testes

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB
Faculdade do Gama - FGA

Orientador (a): Dra. Carla Silva Rocha Aguiar

Brasília, DF
2016

Estratégias para realizar testes funcionais de interface com o usuário – visão de uma equipe de testes

João Guilherme Santana Araruna

Monografia submetida como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software da Faculdade UnB Gama - FGA, da Universidade de Brasília, em 18/07/2017 apresentada e aprovada pela banca examinadora abaixo assinada:

Profa. Dra. Carla Silva Rocha Aguiar, UnB/ FGA
Orientador

Prof. Me. Ricardo Ajax Dias Kosloski
Convidado

Prof. Dr. Fábio Macedo Mendes
Convidado

Brasília, DF
2017

AGRADECIMENTOS

Agradeço primeiramente a Deus por ter me dado saúde para chegar até aqui. Agradeço a toda a minha família e em especial aos meus pais Wesley e Ana Paula que me deram todo apoio e estrutura durante o curso. Ao meu primo Nilton Araruna, hoje já engenheiro de software, e aos amigos de curso mais próximos Bruno Contessotto, Eduardo Brasil, Nilton Araruna, Rafael Fazzolino, Thabata Granja e Thiago Kairala que me acompanharam e me apoiaram em diversas matérias do curso.

Agradeço a todos os professores do curso que contribuíram para o conhecimento que tenho hoje, em especial aos professores Ricardo Ajax e Carla Rocha pela paciência e dedicação entregue ao trabalho desenvolvido.

SUMÁRIO

SUMÁRIO 5

RESUMO 9

1 Introdução 11

1.1 Contexto 11

1.2 Problema 13

- Objetivo geral 16
- Objetivos específicos 16

2 Metodologia 18

3 Testes de software 20

3.1 Importância e definição de teste 20

3.2 Níveis de teste 20

- Teste de unidade 20
- Teste de integração 21
- Teste de módulo 21
- Teste de Sistema 21
- Teste de Aceitação 21

3.3 Tipos de teste 21

- Teste unitário 21
- Teste de aceitação 22
- Teste de confiabilidade 22
- Teste de desempenho 22
- Teste de recuperação 22
- Teste de regressão 22

- Teste de segurança 23
- Teste de configuração 23
- Teste de estresse 23
- Teste de interface gráfica 23
- Testes funcionais 23

3.4 Organização dos testes 24

3.5 Processo de testes 26

3.5.1 Planejamento 26

3.5.2 Execução 28

3.5.2.1 Descrição dos passos 29

3.5.3 Avaliação dos resultados 30

3.6 Considerações Finais do Capítulo 31

4 Automação dos testes funcionais de interface com o usuário 32

4.1 Estratégias de automação 32

4.1.1 Gravação 33

4.1.1.1 CRIANDO UM TESTE AUTOMATIZADO VIA GRAVAÇÃO 34

4.1.2 Codificação 36

4.1.2.1 CRIANDO UM TESTE AUTOMATIZADO VIA CODIFICAÇÃO 37

4.1.3 Linguagem natural 40

5 Aplicação, resultados e avaliações 43

5.1 Survey 43

5.1.1 Considerações temáticas 44

5.1.1.1 GRAU DE AMEAÇA DE ITENS 44

5.1.1.2 ITENS PARA AVALIAR CONHECIMENTO 45

5.1.1.3 ITENS PARA AFERIR ATITUDES E OPNIÕES 45

5.1.1.4 Item para obter informações factual 45
5.1.2 Considerações sobre uso de escalas 45
5.1.2.1 ESCALA ORDINAL 45
5.1.2.2 Escala intervalar 46
5.1.2.3 Escala de razão 46
5.1.2.4 Escala likert 46
5.1.3 Objetivo 46
5.2 Equipe de teste 46
5.2.1 Atividades de teste 47
5.3 Questionário 47
5.3.1 Questões utilizadas - Objetivos 47
5.3.2 Resultados obtidos 48
5.3.2.1 QUANTO AOS PERFIS E TEMPO DE EXPERIÊNCIA 48
5.3.2.2 QUANTO À QUESTÃO 1 49
5.3.2.3 QUANTO À QUESTÃO 2 50
5.3.2.4 QUANTO À QUESTÃO 3 51
5.3.2.5 QUANTO À QUESTÃO 4 53
5.3.2.6 QUANTO À QUESTÃO 5 53
5.3.2.7 QUANTO Á QUESTÃO 6 55
5.4 Avaliação do resultado obtido na questão 6 55
5.5 Entrevista 57
5.6 Considerações finais 60
6 ESTUDO DE CASO 62
6.1 OBJETIVOS 62
6.2 HIPÓTESES 62

6.3 INDICADORES 62
6.4 PROCEDIMENTOS E COLETA DE DADOS 64
6.4.1 APLICAÇÃO DA ESTRATÉGIA DE CODIFICAÇÃO NOS PROJETOS 64
6.4.2 APLICAÇÃO DA ESTRATÉGIA DE GRAVAÇÃO 68
6.4.3 APLICAÇÃO DA ESTRATÉGIA DE LINGUAGEM NATURAL 69
6.4.4 EXECUÇAO MANUAL DOS TESTES 69
6.5 RESULTADOS 70
6.5.1 CUSTO 70
6.5.2 CONFIABILIDADE 72
6.5.3 REUSO 74
6.6 ANÁLISE DOS RESULTADOS 78
7 CONCLUSÃO 81
8 REFERÊNCIAS 83

RESUMO

Este trabalho foi associado a questões sobre as vantagens para uma organização do mercado privado em automatizar os testes funcionais dos softwares por ela sendo desenvolvidos. Neste sentido foram feitos aprofundamentos sobre a própria área de testes de software como níveis, tipos de testes e processos para a execução desta importante atividade para a organização. Outro foco de concentração dos estudos foi sobre métodos de automatização de testes funcionais usando ou a gravação do comportamento das funcionalidades do software ou pela elaboração de códigos. A pesquisa foi subdividida em duas grandes fases. A primeira fase, conduzida neste estudo de TCC teve como foco levantar, por meio de um *survey* e entrevistas, a opinião dos especialistas de testes da própria organização sobre a importância deste tipo de automatização, além de identificar quais são os aspectos vantajosos para a empresa com um procedimento deste tipo. Os resultados foram utilizados em um estudo de caso para avaliar os aspectos identificados como relevantes nesta questão obtidos neste trabalho de TCC.

Palavras-chave: Testes de software, automação, testes funcionais de interface com o usuário.

ABSTRACT

This study was associated with questions related to the advantages to automate the functional tests for an organization of the private sector. In this way was done deepening studies about the software testing area itself as tests levels, tests kinds and a general software testing process to accomplish this important activity. Another important focus was about the functional software testing automation by means of recording the functionalities behavior or coding. The Research was divided in two steps. The first phase, conducted this TCC study focused raise, through a survey and interviews, the opinion of the organization's own test experts about the importance of this type of automation, and identify what are the beneficial aspects for the company with such a procedure. The results were used in case study to evaluate the issues identified as relevant in this matter obtained in this work TCC.

Keywords: Software testing, automation, graphical user interface testing (GUI testing).

1 INTRODUÇÃO

Este primeiro capítulo expõe qual o contexto em que o trabalho está inserido, o problema a ser resolvido, a justificativa do problema, os objetivos gerais e específicos e a descrição de cada capítulo.

1.1 CONTEXTO

O nível de qualidade de um software está diretamente ligado ao sucesso da sua fase de desenvolvimento (Gopal, 2010). Para avaliar se o desenvolvimento foi feito de maneira correta e se o sistema apresenta um nível e qualidade desejado pelo cliente, é necessário a realização de testes. Galin(2004) faz um apanhado de conceitos sobre o significado genérico do termo qualidade de software onde são apresentadas definições clássicas como:

> *"O grau no qual um sistema, componente ou processo satisfaz requisitos especificados ou o grau no qual eles satisfazem as expectativas do usuário" (IEEE, 1990).*

No seu apanhado Galin (2004) acrescenta outras definições de autores consagrados na área, onde a qualidade também significa ausência de deficiências, segundo Juran, e conformidade com os requisitos estabelecidos pelos usuários (clientes), segundo Crosby. Desta forma, ao satisfazer as necessidades dos clientes as funcionalidades provêm a sua satisfação sobre o produto entregue.

Ao detalhar conceitos de qualidade de software, a norma ISO/IEC 25000 (ISO/IEC-25000, 2005) os subdivide nos seguintes grandes aspectos: A qualidade do produto e a qualidade em uso. A qualidade em uso envolve os benefícios esperados pelo usuário ao usar o software solicitado. Desta forma, o atendimento aos requisitos definidos tende a gerar produtos de alta qualidade que, por sua vez serão fornecerão benefícios ao serem usados pelos seus usuários. Além disso, a melhoria na qualidade do processo usado na construção do software é uma maneira de fomentar a qualidade do produto de software que será entregue de forma que os aspectos se completam influenciando uns aos outros em termos de qualidade de software.

Continuando no esclarecimento do seu raciocínio, Galin (2004) cita Pressman que, por sua vez, defende que a qualidade também está relacionada aos requisitos funcionais do software, aos padrões de qualidade definidos nos acordos de prestação de serviços de desenvolvimento e ao uso de boas práticas de engenharia de software. Neste sentido as definições de Pressman, segundo Galin (2004), provêm direcionamentos operacionais indicando a necessidade de se testar o grau

no qual os requisitos estabelecidos para a aplicação sejam satisfeitos, sejam eles funcionais ou de outras naturezas.

Neste contexto surgem os conceitos de verificação e validação do software sendo construído onde:

- A verificação é uma tarefa que tem o objetivo de confirmar ou não a fidelidade do produto entregue após uma das fases de desenvolvimento à especificação dos requisitos que foram feitas anteriormente, ou seja, avaliar se o produto foi construído corretamente (Maldonado, 2007; Naik, 2011).
- A validação está voltada para os critérios de aceitação do cliente, ou seja, tem o objetivo de garantir que o sistema se comporta de acordo com as especificações que o cliente descreveu ao levantar as características e funcionalidades do sistema (Maldonado, 2007; Galin, 2003).

Neste caminho surge o conceito de teste de software como uma forma de avaliar (verificar ou validar) se as características estabelecidas pelos requisitos do software foram atendidas para, consequentemente, satisfazer as expectativas do usuário no atingimento de maiores níveis de qualidade do software entregue (Maldonado, 2007). Neste caso,

"O teste é o processo de execução de um programa com a intenção de encontrar erros. " (Myers, 2012)

Os testes podem ser organizados em unidades chamadas de casos de teste. Um caso de teste descreve os passos ordenados para a execução do teste e também apresentam o que é esperado do sistema após o usuário seguir essas instruções (Rätamann, 2003). Os tipos, níveis e casos de testes são declarados em um documento conhecido como roteiro de testes.

As atividades necessárias para realizar os testes poder ser abordadas sob a ótica de um subprocesso existente no processo de desenvolvimento de software. Neste caso, o subprocesso de teste pode ser definido pelos passos de Planejamento, Geração dos casos de teste, Execução e Avaliação dos resultados obtidos (IEEE, 2014).

Ahmed(2015) afirma que os testes são atividades que exigem esforço e dedicação dos envolvidos para que a qualidade do software esteja aceitável.

"O teste é uma forma importante para garantir e melhorar a qualidade do sistema de software. No entanto, é uma atividade demorada e de trabalho intensivo. " (Mao, 2014) e, por este motivo, automatizar a atividade de teste é uma boa política ao longo do ciclo de vida de

desenvolvimento de software, pois pode minimizar erros de execução dos testes envolvidos devido a falhas humanas dos próprios testadores (Hushalini, 2014).

Os testes funcionais de interface com o usuário são normalmente feitos manualmente e estão dentro do nível de teste de sistema e de aceitação com o objetivo de, respectivamente, verificar se as funcionalidades estão funcionando corretamente e validar se as funcionalidades existem conforme foram definidas pelo usuário.

Além disso, caso sejam solicitadas mudanças ao longo do desenvolvimento, ambos os tipos de teste devem ser repetidos para avaliar se a funcionalidade está de acordo com suas especificações. Unindo essa ideia de repetição dos testes com as ideias de Mao, 2014 e Hushalini, 2014 citadas anteriormente, é razoável pensar em automatizar os testes funcionais de interface com o usuário ao longo de um desenvolvimento de software.

1.2 PROBLEMA

A organização estudada neste trabalho de TCC é uma empresa do mercado privado que desenvolve softwares de acordo com as demandas contratados pelos seus clientes, consumidores deste tipo de serviço. Neste sentido, o produto de software demandado à empresa deve satisfazer os requisitos funcionais definidos. Os requisitos devem não somente funcionarem corretamente, mas também estarem de acordo com as definições estabelecidas significando que eles devem ser verificados e validados. Verificar os requisitos funcionais solicitados pelos usuários é da responsabilidade da equipe de testes mantida pela organização, e é feita por meio dos testes funcionais de interface do usuário. Os resultados dos testes estarão associados à qualidade do produto gerado que, por sua vez, será o alvo principal de avaliação por parte do cliente para a aceitação do produto entregue.

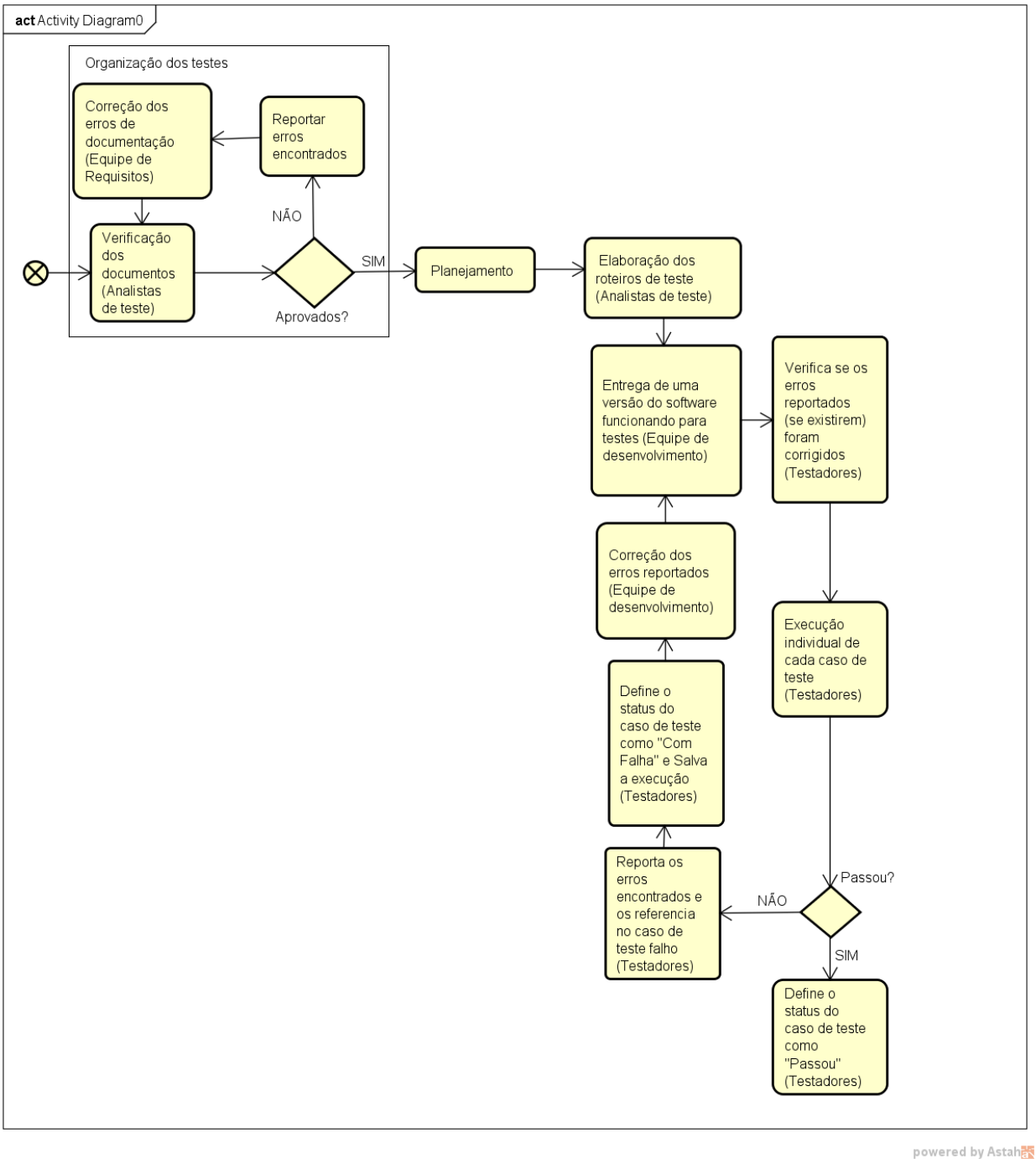
A equipe de testes da organização estudada neste TCC é constituída por um Gerente de teste, 5 Analistas de teste e 4 testadores. Existe um gerente da equipe de testes que trata juntamente com os gerentes de projeto questões relacionadas à sincronização dos cronogramas da equipe de teste com os cronogramas dos projetos de software em andamento na organização.

A equipe de testes descrita no parágrafo anterior possui e segue um processo de teste definido para a organização onde, inicialmente antes mesmo da etapa de planejamento, é feita uma verificação dos documentos que apoiarão a elaboração do plano de testes (Caso de uso, regras de negócio, descrição de interface, etc). Caso existam problemas com estes documentos, eles são

reportados aos seus responsáveis (equipe de desenvolvimento de requisitos) para serem reparados. Após os reparos os documentos são usados para a elaboração do plano de testes que, por sua vez, contém também os casos de testes a serem executados.

Com base neste planejamento, o plano de testes é realizado, os resultados são registrados para serem avaliados em seguida. Para a execução do plano de testes estabelecido, já deve estar disponível a versão executável do software, além de todos os itens necessários para esta execução (roteiro de testes, casos de testes, etc). Os testadores executam os testes reportando os erros encontrados durante a execução. Após o encerramento das execuções os erros reportados devem ser corrigidos pela equipe de desenvolvimento do projeto de software para serem então submetidos novamente à equipe de testes. A equipe repete os testes necessários para avaliar a correção dos erros anteriormente apontados antes de seguir em frente no seu plano de testes. Caso os erros estejam corrigidos os testadores fecham o relatório de erros encontrados ou retornam o relatório de erros identificados e o ciclo se repete até que todos os erros identificados sejam resolvidos. A imagem deste fluxo descrito pode ser vista na Figura 1.

Figura 1: Processo de testes da equipe de testes da organização estudada. Fonte autor.



Para facilitar a execução dos testes funcionais de interface com o usuário, a equipe de testes realiza estudos sobre como proceder para automatizá-los. Para automatizar este tipo de testes existem basicamente duas estratégias distintas. Uma delas é a gravação que grava as ações do usuário com a aplicação reproduz/executa a gravação que foi feita. Outra consiste em codificar

cada passo realizado pelo usuário para automatizar sua interação com o sistema. Desta forma este trabalho de TCC foi feito com base no seguinte problema:

Na visão da equipe de testes da empresa estudada, quais são as vantagens de proceder com a automatização dos testes funcionais de interface com o usuário e por qual estratégia a automatização deve ser feita?

- **OBJETIVO GERAL**

Para este trabalho o objetivo geral pode ser caracterizado como:

Analisar as percepções dos testadores sobre as estratégias de automatização dos testes funcionais de interface com o usuário.

Neste caso são considerados como aspectos: os métodos concorrentes (gravação do comportamento da funcionalidade ou a elaboração de códigos, ambos visando automatizar os testes funcionais de interface com o usuário); os aspectos vantajosos deste tipo de procedimento conforme a visão dos especialistas da equipe de testes (ex.: melhoria dos prazos, custos, confiabilidade dos testes, dentre outros que possam ser descobertos).

Os resultados deste trabalho serão utilizados em um estudo de caso para avaliar se a automação dos testes funcionais de interface com o usuário implica nas vantagens apontadas pelos especialistas da equipe de teste da organização.

- **OBJETIVOS ESPECÍFICOS**

Para este Trabalho de Conclusão de Curso pretende-se cumprir os objetivos específicos abaixo descritos, afim de cumprir com o objetivo geral estabelecido:

1. Caracterizar os níveis e tipos de teste, suas necessidades e importância no desenvolvimento de software;
2. Caracterizar as estratégias de automação de testes funcionais de usuário, considerando a automatização via código e a gravação do comportamento da funcionalidade;
3. Definir uma estratégia para coletar em campo na organização estudada as opiniões dos seus especialistas em testes sobre qual método usar para automatizar os testes funcionais de interface com o usuário (dentre dois possíveis: gravação e elaboração de código);
4. Aplicar a estratégia para a coleta das opiniões dos especialistas da organização

estudada;

5. Avaliar os resultados consolidando os achados.

- **Organização do Trabalho**

Esta seção contém a organização do trabalho, ela apresenta os capítulos desenvolvidos e seus respectivos temas.

O estudo começa no capítulo três o qual aborda a definição dos testes e como seu planejamento é feito. Em seguida é destacada a importância dos testes funcionais em um ambiente de desenvolvimento de software.

O capítulo quatro descreve as possíveis estratégias de automatizar os testes funcionais de interface com o usuário. Sendo assim serão caracterizadas nesse capítulo a automação via código e via gravação, que grava o comportamento de determinada funcionalidade.

O capítulo cinco apresenta e executa as estratégias adotadas para obter as opiniões dos profissionais da área de qualidade e testes sobre testes funcionais e sua automação, e ao final os dados obtidos são analisados e apresentados.

2 METODOLOGIA

Esta seção detalha principalmente a metodologia utilizada no trabalho de TCC, de forma geral uma pesquisa pode ser classificada quanto à sua natureza, abordagem, objetivos, além de Procedimentos técnicos (Moresi, 2003; Gil, 2008).

Segundo esses autores quanto à sua natureza, uma pesquisa pode ser classificada como sendo uma pesquisa básica ou aplicada. No caso desta pesquisa como um todo, ela pode ser classificada como sendo de natureza *aplicada*, por ter como objetivo a geração de conhecimentos práticos a serem aplicados em uma realidade circunstancial, ou seja, dirigidos à solução de problemas específicos envolvendo interesses locais de uma determinada organização de desenvolvimento de software do mercado privado.

Segundo Gil (2008) a pesquisa como um todo, pode ser classificada quanto aos seus objetivos como sendo de características exploratórias, pois pretende identificar na opinião dos especialistas em testes da organização quais as vantagens para a organização em automatizar esses tipos de testes.

De acordo com definições estudadas em Moresi (2003) e Gil (2008), a pesquisa pode ser considerada como usando uma abordagem *mista*, tanto qualitativa, quanto quantitativa. A característica quantitativa estará representada nas pesquisas de campo pela consolidação das respostas dos *Surveys* usados no levantamento das opiniões dos especialistas da organização sobre a automatização dos testes funcionais de interface com o usuário. A característica qualitativa deve-se ao fato da coleta de dados ser feita no ambiente natural onde o pesquisador é o elemento chave, analisando indutivamente os dados obtidos, afim de compreender os fenômenos observados, foco básico deste trabalho.

Quanto aos procedimentos técnicos que serão usados devem ser ressaltados os seguintes:

- As pesquisas de campo por meio de *surveys* baseados em questionários e entrevistas estruturadas, onde as pessoas envolvidas na pesquisa (especialistas em testes da organização estudada) serão interrogadas diretamente a respeito do fenômeno de interesse (tipos de automatização dos testes funcionais de interface com o usuário);
- Bibliográfica, por se utilizar de recursos tais como artigos, livros e publicações de base científica, desde que acessíveis ao pesquisador e telematizada pelo seu uso intensivo de computador, internet.

Os procedimentos para coletar os dados para pesquisa são o questionário e a entrevista. O

questionário será respondido pelos profissionais da equipe de testes via internet utilizando o formulário do google docs. Quanto a entrevista, ela será do tipo estruturada já que segundo Gil (2008), as entrevistas estruturadas possuem um número definido de perguntas e a ordem em que elas são feitas ao entrevistado é fixa. As perguntas presentes tanto no questionário quanto na entrevista terão como objetivo obter as opiniões dos profissionais de testes sobre as vantagens de se automatizar os testes funcionais de interface com o usuário e qual a melhor a estratégia de automação dentre gravação e codificação.

Figura 2: Seleção metodológica. Fonte autor.

Natureza	Pesquisa Básica	Pesquisa Aplicada	
Objetivos	Pesquisa Exploratória	Pesquisa Descritiva	Pesquisa Explicativa
Abordagem	Pesquisa Qualitativa	Pesquisa Quantitativa	Pesquisa Explicativa
Métodos Investigativos	Experimento	Bibliográfico	Documental
	Estudo de Caso	Ex-post facto	Levantamento
	Pesquisa com Survey	Pesquisa-Ação	Participante
Técnicas de Coleta de Dados	Observação Simples	Observação sistemática	Observação participante
	Questionário	Documentos	Entrevista Estruturada
	Entervista semi-estruturada	Entrevista Informal	

3 TESTES DE SOFTWARE

Este capítulo visa explicar as estratégias de testes utilizadas no desenvolvimento de software e descrever as práticas utilizadas no processo de testes, relacionando-os com os testes funcionais para destacar a importância deles durante a verificação do software.

3.1 IMPORTÂNCIA E DEFINIÇÃO DE TESTE

Hoje em dia dificilmente realizamos nossas atividades diárias sem a participação ou apoio de algum software. Infelizmente falhas ocorrem durante o uso desses sistemas e podem prejudicar o rendimento das nossas tarefas e prejuízos financeiros. As práticas relacionas ao teste de software visam eliminar ou reduzir ao máximo as inconsistências que um sistema pode apresentar e consequentemente aumentar sua qualidade (Larkman, 2012).

Teste é uma prática que se apoia nos conceitos de verificação e validação para identificar e prevenir erros, ou seja, ela é crucial para que as entregas feitas preencham a expectativa de qualidade do cliente (Naik, 2011; Maldonado, 2007)

3.2 NÍVEIS DE TESTE

A facilidade de corrigir erros está diretamente ligada com a ideia de encontra-los previamente o mais breve possível, essa situação pode ser apoiada pela utilização dos níveis de teste (Stephens, 2015). O autor define o teste como possuindo 6 níveis: Teste de Unidade, Teste de Integração, Testes Automatizados, Teste de Interface do Componente, Teste de Sistema e Teste de Aceitação.

- **TESTE DE UNIDADE**

Neste nível o foco é o código da aplicação. Dessa forma os testes também são feitos por meio de codificação e procuram erros em pequenos pedações de código. Em geral, os códigos de teste são voltados para as ações de um único e específico método mas também podem ser criados códigos de testes para testar as partes de um método. Dessa forma devem ser codificados testes que verificam em um determinado método seus fluxos felizes e exceções tratadas (Stephens, 2015, Ammann, 2008).

- **TESTE DE INTEGRAÇÃO**

Nível de teste apoiado pelos testes de regressão. Aqui o objetivo é verificar se as partes do sistema (ex.: banco de dados, código, interface, recursos externos) estão interagindo de maneira correta a medida que as funcionalidades são inseridas no código.

- **TESTE DE MÓDULO**

Nível de teste focado em verificar a estrutura, comportamento e interação dos módulos ou componentes individuai. Módulos são formados de unidades, por exemplo dentro de um módulo você pode reunir classes, métodos, pacotes que serão usados durante o desenvolvimento do software.

- **TESTE DE SISTEMA**

Aqui o foco é a iteração com o sistema já funcionando com o objetivo de verificar se ele está de acordo com as especificações. Este nível de teste exige grande atenção dos testadores e os mínimos detalhes devem ser observados ao executar os cenários que existem dentro da aplicação.

- **TESTE DE ACEITAÇÃO**

Nível de teste voltado para a visão usuário final, ou seja, aqui o objetivo é definir se produto final é realmente o que o cliente quer. Para atingir o objetivo é feita a verificação dos requisitos que foram levantados na fase de elaboração do projeto reproduzindo ações que o usuário final irá realizar ao interagir com o sistema.

3.3 TIPOS DE TESTE

Testes são atividades que além de garantir o funcionamento correto da aplicação desenvolvida visam cobrir itens como segurança, confiabilidade, usabilidade, desempenho, entre outros. Para atingir esse objetivo existem alguns tipos de teste que surgiram a partir dos conceitos dos níveis de teste e serão conceituados nesta seção.

- **TESTE UNITÁRIO**

Definição praticamente igual ao do nível teste de unidade. Os testes unitários são construídos por meio de codificação e testam pequenas partes ou menores unidades do código de

software, como métodos, funções e outros (Naik, 2011; IEEE, 2014; Pressman, 2000).

- **TESTE DE ACEITAÇÃO**

Esse tipo de teste que é realizado pelo próprio cliente e pode envolver ou não o apoio de profissionais envolvidos no projeto. O objetivo é validar os requisitos e confirmar que os critérios de aceitação do cliente foram atingidos (Naik, 2011; IEEE, 2014).

- **TESTE DE CONFIABILIDADE**

Tipo de teste que uso de dados numéricos para medir a disponibilidade da aplicação, ou seja, se ela atende à procura dos seus usuários finais quando ela é acessada (Naik, 2011; IEEE, 2014).

- **TESTE DE DESEMPENHO**

Tipo de teste que está dentro no nível de sistema. Aqui são mensurados itens como capacidade e tempo de resposta. A capacidade se refere à quantidade de acessos simultâneos feitos e o tempo de resposta para o quanto dura a execução das funcionalidades do sistema. Nesse tipo de teste fatores como velocidade da internet, hardware e configuração das máquinas são determinantes e influenciam nos resultados (IEEE, 2014; Pressman, 2000).

- **TESTE DE RECUPERAÇÃO**

A recuperação a que se refere esse tipo de teste é a capacidade do software, após ter sido encerrado por alguma falha, reiniciar e retomar o ponto em que as atividades que estavam sendo feitas usuário (IEEE, 2014; Pressman, 2000).

- **TESTE DE REGRESSÃO**

Teste realizado para manter a integridade do sistema, ou seja, se o sistema continua funcionando como esperado após as modificações que são feitas durante o desenvolvimento. Em termos técnicos, aqui é verificado se as camadas e componentes do sistema continuam se comunicando de forma correta. Esse tipo de teste consome grande quantidade de tempo, logo sua execução deve ser feita quando uma quantidade de funcionalidades considerável foi entregue. (IEEE, 2014; Pressman, 2000).

- **TESTE DE SEGURANÇA**

Teste voltado para a proteção do software em relação a fatores externos. Aqui os testadores verificam se o software consegue manter seu funcionamento e o sigilo dos dados em situações como, por exemplo, a de um ataque de hackers ou de usuários que façam mau uso da aplicação (IEEE, 2014; Pressman, 2000).

- **TESTE DE CONFIGURAÇÃO**

Tipo de testes que visam verificar a aplicação em diferentes configurações, ou seja, sistemas operacionais diferentes, recursos de hardware diferentes, navegadores diferentes e outros. Caso sejam encontradas limitações do software nesses testes, elas devem ser documentadas para possíveis manutenções futuras (IEEE, 2014; Pressman, 2000).

- **TESTE DE ESTRESSE**

Esse tipo de serve busca encontrar o limite do software impondo a ele situações anormais, situações que exijam alto processamento. Por exemplo, acessar uma mesma funcionalidade ou concluir uma ação com um volume grande de usuários simultâneos (Naik, 2011, IEEE, 2014; Pressman, 2000).

- **TESTE DE INTERFACE GRÁFICA**

Verifica se os componentes da interface gráfica estão respondendo às ações do usuário de maneira correta. Esse tipo de teste permite que sejam identificadas falhas também em outras camadas do sistema (IEEE, 2014).

- **TESTES FUNCIONAIS**

São testes que se preocupam com as entradas e saídas do sistema. Dependendo do sistema, o volume de entradas, saídas e fluxos possíveis pode ser muito grande a ponto de tornar-se crítica a verificação. Neste caso deve-se fazer uma priorização dessas entradas e saídas visando dar um enfoque maior nas funcionalidades mais importantes para o usuário final (Naik, 2011).

Alguns comentários sobre os testes funcionais são pertinentes neste momento. Dentre eles podem ser ressaltados:

  - Apesar dos testes de unidade em geral se focarem no código, eles também podem ser

funcionais, desde que os dados dos testes elaborados sejam baseados nos documentos que descrevem as funcionalidades do sistema e suas respectivas regras, ou seja, nos requisitos (Perry, 2006). Segundo Howden (1980), as funcionalidades de um sistema são como funções matemáticas, ou seja, podem possuir uma ou várias entradas assim como podem ter uma ou várias saídas. A diferença é que ao invés de o resultado de uma funcionalidade ser apenas um ou vários números, ele é um item cadastrado ou uma lista de itens, por exemplo.

- Myers (2012) afirma que nos testes funcionais os testadores devem simular as funcionalidades do sistema na visão do cliente com o objetivo de encontrar inconsistências entre a aplicação e sua documentação. Já que os testes funcionais são feitos na visão do cliente podemos concluir que eles contribuem não só na verificação do sistema como também para o sucesso da validação das funcionalidades da aplicação que está sendo entregue ao cliente.
- Perry (2006) diz que qualquer alteração que seja feita na aplicação exige uma nova rodada de execução de testes para garantir que outras partes da aplicação não foram impactadas. Um sistema só pode ser liberado para validação depois que todos os seus testes atingirem um nível de cobertura aceitável (Naik, 2011).
- Os testes funcionais dependem do sucesso dos testes estruturais. Segundo Myers (2012), para a realização dos testes funcionais deve haver confiabilidade na cobertura lógica dos testes unitários. Problemas durante a construção do software e dos testes unitários aumentam o tempo gasto com eles e reduzem o tempo de execução dos testes funcionais. Aqui encontramos um motivo para automatizar os testes funcionais, já que com a automação haverá uma maior rapidez na execução se comparada com a execução manual. Hogg (2013) apoia essa ideia afirmando que os testes devem ser automatizados de forma a evitar que sua execução seja humana.

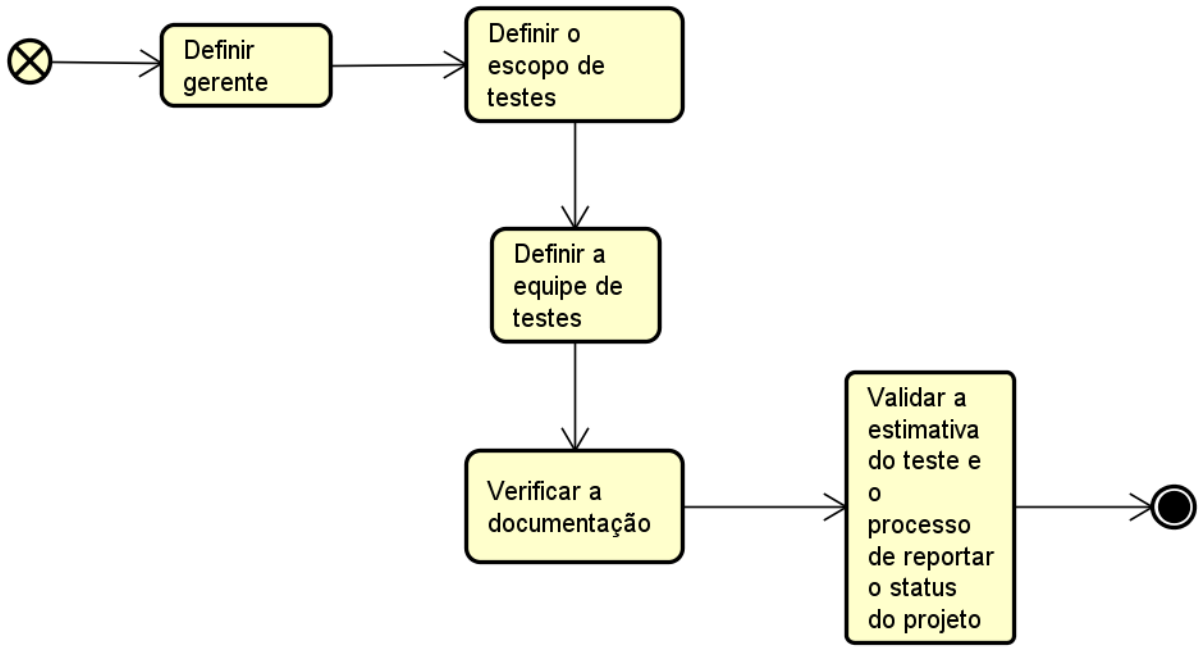
3.4 ORGANIZAÇÃO DOS TESTES

A tarefa de teste exige uma organização antes de iniciar o seu planejamento. Na organização são definidas questões relacionadas ao tempo e aos recursos necessários para que o processo de testes seja concluído com sucesso, sem perda de recursos e atrasos (Perry, 2006). Segundo este autor a organização do processo de teste deve passar pelos seguintes passos:

1. Definir o gerente de testes; O gerente de testes escolhido deve definir a equipe de teste que deseja trabalhar ao longo do processo de software, bem como o processo de software, além de escrever o plano de testes e avaliar os resultados finais; na empresa estudada esta atividade já está resolvida por meio da designação do gerente da equipe de teste pela organização.
2. Definir o escopo do teste, aqui é avaliado se existem objetivos específicos que devem ser atingidos pela equipe de testes durante a execução dos testes. Esse passo é importante já que, para se elaborar o plano de testes é necessário compreender o escopo dos testes. Esse passo é realizado pelo gerente de testes da organização estudada.
3. Definir a equipe de testes. Na empresa estudada esta atividade já está resolvida por meio da designação da equipe de testes conforme designado pela organização.
4. Verificar a documentação, aqui os analistas de teste devem procurar falhas na documentação elaborada para o sistema para que no momento do planejamento não haja problemas nas entradas e resultados esperados dos testes. Além disso esse passo auxilia que a equipe de testes tenha um maior conhecimento da aplicação que está sendo desenvolvida. Esta atividade é feita a cada projeto que será submetido à avaliação feita pela equipe de testes.
5. Validar a estimativa do teste e o processo de reportar o status do projeto. O objetivo deste passo é saber quais os recursos estarão disponíveis para a construção e execução dos testes. E além disso tomar conhecimento de como o status do projeto será reportado aos testadores. Os testes dependem do tempo gasto com o desenvolvimento do software e é necessário tomar conhecimento do tempo que se tem entre o fim do desenvolvimento e a entrega é adequado. Na empresa estudada esta atividade é da alçada do gerente da equipe de testes. Ele a executa com base nos cronogramas dos projetos de software em andamento na organização.

A figura abaixo apresenta o fluxo das atividades que descritas acima:

Figura 3: Fluxo de atividades da organização dos testes. Fonte autor.



3.5 PROCESSO DE TESTES

Após a organização dos testes, o processo de teste é iniciado e ele é basicamente definido em 3 partes: planejamento, execução e avaliação dos resultados.

3.5.1 PLANEJAMENTO

A entrada para a elaboração do planejamento de testes é o plano de projeto (Perry, 2006). O plano de projeto contém as tarefas relacionadas a requisitos, desenvolvimento, testes, entre outras que devem ser feitas para a conclusão do projeto. A saída ou entrega do planejamento é o plano de teste que apresenta informações sobre o software que será testado e os testes que serão realizados nele (Perry, 2006, Rätamann, 2003).

De forma geral, na fase de planejamento é definido o que será testado, como será testado, quem testará o quê, bem como a documentação que será usada para registrar as atividades que envolvem o teste. A mão de obra necessária para configurar e deixar o ambiente estável também são mensurados nessa fase (IEEE, 2014).

Segundo Perry (2006) essa fase também contempla a definição dos itens críticos e

características do sistema que devem ser observados de perto pelos testadores durante a execução dos testes. Dentre esses itens podemos citar confiabilidade, controle de acesso, facilidade de uso, ou seja, são itens que estão ligados diretamente com a percepção do testador.

As técnicas de teste devem ser selecionadas baseando-se na avaliação dos objetivos a serem alcançados com os testes. De acordo com os objetivos que devem ser alcançados, é definido como será feito o uso dos testes estruturais e funcionais (Perry, 2006).

Os testes estruturais e funcionais são os dois meios utilizados pelo processo de testes de software para assegurar que o software entregue está de acordo com o nível de qualidade esperado pelo cliente (Dyer, 1993). Segundo Dyer (1993) e Ahmed (2015) os testes estruturais, também chamados de teste de caixa-branca, focam no código, na parte interna do sistema que está sendo desenvolvido, ou seja, nas classes, métodos, estruturas de dados, padrões de projeto e outros. Já os testes funcionais ou testes de caixa-preta impõem que o testador esqueça a parte interna do sistema, ou seja, o código e se volte para a execução das funcionalidades do sistema.

As técnicas de teste escolhidas servem como pré-requisitos para a escolha do ambiente de testes. A estrutura do ambiente de teste deve favorecer toda construção e execução dos testes como também deve arquivar os resultados e erros encontrados (IEEE, 2014). O arquivamento de resultados e erros encontrados, serve de insumo para a geração de relatórios que podem ser usados em para analisar o desempenho dos testes e do projeto como um todo.

Tomando como referência o parágrafo anterior chegamos aos casos de teste que são os artefatos identificados e documentados nessa fase. Segundo Misra (2000), o planejamento de teste tem como um de seus principais objetivos identificar os casos de teste, o cumprimento desse objetivo é essencial para se iniciar o planejamento da execução. Sharma (2012) define que um caso de teste deve informar aos testadores os passos e condições (entradas) que devem ser seguidos para se chegar aos resultados (saídas) que devem ser verificados no momento da execução. Sendo assim, a geração dos casos de teste é uma das atividades mais importantes do planejamento, vale destacar que os resultados obtidos nela são usados para a criação do plano de teste.

A equipe de testes estudada nesse TCC elabora e salva os casos testes a serem executados utilizando uma ferramenta chamada TestLink, ela será melhor descrita e apresentada na seção 3.5.3 deste capítulo.

Ao fim dessa fase é realizada a criação do plano de teste que é o documento que apoiará a fase de execução e avaliação no processo de testes. Após o levantamento dos dados anteriormente,

um plano de testes deve possuir documentado algumas informações importantes, dentre elas estão os itens descritos abaixo (Myers, 2012):

- Objetivos. Os objetivos de cada fase bem definidos;
- Critérios. Estabelecer critérios que devem ser atingidos para uma fase possuir o status de finalizada;
- Cronograma. Definir prazo e momento para as atividades de identificar, descrever e executar casos de teste;
- Responsáveis. Designar quem irá realizar as atividades descritas no cronograma e quem irá resolver os erros possivelmente encontrados;
- Livraria de casos de teste e padrões. Aqui são definidas técnicas para padronizar a identificação, escrita e armazenamento de casos de teste;
- Ferramentas. Informar e descrever as ferramentas necessárias para a realização das atividades de teste, incluindo as atividades de automação;
- Procedimentos de depuração. Escolhe a forma como os erros serão relatados pelos testadores com objetivo de facilitar o entendimento do desenvolvedor no momento da correção;
- Teste de regressão. Devem ser realizados para identificar se as correções de erros não impactaram em outras partes ou funcionalidades do sistema.

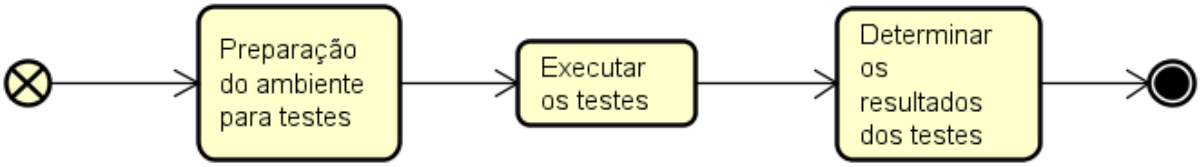
Com um plano de testes documentado, já possuímos insumo para partir para a fase de execução de testes.

3.5.2 EXECUÇÃO

Testes de sistema são considerados onerosos para o projeto de software e suas execuções são consideradas complicadas. Dessa forma nessa fase pode haver uma priorização dos casos de teste na fase de execução. Caso haja necessidade de priorização, os envolvidos podem definir que os casos de teste que possuem por exemplo prioridade para o cliente ou requisitos que mudam constantemente são executados primeiro (Srikanth, 2012).

Segundo Rätamann (2003) a execução dos testes pode ser dividida em 3 passos que estão representados na figura abaixo:

Figura 4: Fluxo de execução dos testes. Fonte autor



3.5.2.1 DESCRIÇÃO DOS PASSOS

No primeiro passo, o testador procura garantir que que o teste pode ser realizado, ou seja, não nenhum tipo de erro impeditivo ou falta de recurso para executá-lo. Voltando a atenção para os testes funcionais que envolve a relação humano/computador, um pré-requisito para sua execução é que a interface e o sistema como um todo esteja disponível pois a funcionalidade será testada na visão do cliente.

Em relação ao passo de execução, os testes funcionais podem ser feitos manualmente ou podem ser automatizados pode meio da interface gráfica do sistema. O autor destaca que caso a interface esteja sempre ao alcance do testador, os testes podem ser dirigidos através dela automatizando a ações do usuário usando mouse e teclado na tela do sistema.

O passo 3 referente a determinação dos resultados dos testes depende da ferramenta escolhida para rodar os testes ou registrar seus resultados, existem diversas ferramentas com diversas formas de definir o status dos resultados obtidos. No caso da equipe de testes envolvida o resultado dos testes executados são armazenados na ferramenta TestLink, ela será melhor descrita na próxima seção.

É importante destacar que o testador deve estar com um bom entendimento da documentação do sistema, já que ela permite encontrar erros que não podem ser tratados nos casos de teste, ex.: o posicionamento ou o funcionamento de componentes da tela só pode ser avaliado baseando-se na descrição de interface (Rätamann, 2003).

Durante o desenvolvimento de um software é normal que desenvolvedores e testadores tenham entendimentos diferentes do que representa uma falha ou que a documentação esteja um pouco confusa e gere dupla interpretação. Dessa forma durante a execução tudo deve ser registrado, principalmente os erros para que seja possível que a pessoa ou área do projeto que foi

direcionado o erro possa discordar ou concordar com o que foi escrito (IEEE, 2014). Os registros de execução servirão de insumo para o último passo do processo de testes, a avaliação dos resultados.

A equipe de testes estudada nesse TCC, faz uso da ferramenta Jira para reportar os bugs e erros encontrados. Essa ferramenta é paga e o acesso via web auxilia na gestão de projetos, assim como para o acompanhamento de erros e bugs contidos no projeto. Nesses reportes de erros e bugs são preenchidas informações como o projeto, o ciclo de teste que está sendo executado, a ordem de serviço, o caso de uso referente ao caso teste que se está executando, a descrição do erro, a gravidade, o desenvolvedor que deverá resolver o erro e colocá-lo em verificação e outros. A relação da fermenta TestLink com a ferramenta JIRA será explicada na seção a seguir.

3.5.3 AVALIAÇÃO DOS RESULTADOS

O resultado dos testes em geral é definido como passou ou falhou após o testador comparar, durante a execução, o resultado obtido com o resultado esperado que está registrado no caso de teste.

Em geral nessa fase é feito o uso de alguma ferramenta que realiza a análise de dados quantitativos como a quantidade de testes que passaram, a quantidade de testes que não passaram, a quantidade de testes que foram bloqueados devido a erros impeditivos e ao final definem uma porcentagem para o nível de cobertura que foi atingido com a execução realizada. Aqui também podem ser feitas análises de acordo com a gravidade dos erros. Um fator relevante dessa avaliação é que ela pode ter influência na escolha de quais casos de teste serão priorizados na próxima execução.

Como foi citado na seção anterior, a equipe de testes estudada faz uso do TestLink para elaborar os casos de teste. O TestLink é uma ferramenta web, gratuita e de código aberto voltada para o gerenciamento dos testes, ela tem como recurso o cadastro de planos e casos de teste. Essa mesma ferramenta é utilizada pelos testadores para definir o resultado dos casos de teste. Nesse momento entra a relação do TestLink com o Jira. Os erros reportados no Jira possuem um id único, esse id deve ser incluído no TestLink através do caso de teste que apresentou o erro. Nesse momento o testador basicamente copia o(s) id(s) do(s) erro(s) em uma caixa de texto que cada caso de teste do TestLink disponibiliza, falha o caso de teste e salva a execução. Com base nas quantidades de testes falhados e passados, o próprio TestLink calcula a cobertura de teste que foi

atingida.

3.6 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Este capítulo teve com o foco apresentar os fundamentos da área de testes, a fim de constituir um arcabouço de conhecimentos necessário para obter e consolidar conhecimentos mais específicos sobre automatizar os testes funcionais de interface com o usuário e quais suas vantagens na visão da equipe de testes da empresa estudada.

Aqui foram discutidos níveis de testes, tipos de testes, além dos passos presentes em um processo de testes (organização, planejamento, execução e avaliação dos resultados). Para uma melhor caracterização da organização estudada, durante os passos do processo de testes, foram feitas relações com a equipe de testes estudada, definindo quais papéis executaram as tarefas necessárias, além das ferramentas utilizadas para tal. Em relação ao gerenciamento de testes que envolve a criação de planos e casos de testes, for apresentada a ferramenta web TestLink. Para o gerenciamento de projetos e, reportes e acompanhamentos de erros foi apresentada a ferramenta web Jira. Além disso também foi apresentado como a equipe de testes apresenta no TestLink os reportes feitos no Jira durante a execução.

Dando continuidade ao estudo realizado neste trabalho de conclusão de curso, o próximo capítulo aprofunda sobre a importância de automatizar os testes e nas estratégias para automatizar os testes funcionais de interface com o usuário considerando as formas por meio de gravação do comportamento da funcionalidade e por meio da geração de código. Essas estratégias serão descritas destacando suas vantagens e desvantagens. Além disso, será explicado os passos para instalação e criação dos testes automatizados para cada estratégia baseando-se nas ferramentas utilizadas pela equipe de testes da organização.

4 AUTOMAÇÃO DOS TESTES FUNCIONAIS DE INTERFACE COM O USUÁRIO

No desenvolvimento de software existem dois pontos que sempre são acompanhados de perto pelos envolvidos, custo e qualidade. Segundo Li (2005), reduzir o custo da produção de software e melhorar a qualidade do software entregue são assuntos que sempre serão enfatizados nos projetos de software. Strugar (2014) adiciona que a maioria das empresas de software ainda verificam a aplicação desenvolvida de forma manual, deixando a automação como um segundo plano. Alégroth (2015) afirma que tarefas feitas de forma manual são em sua maioria de alto custo, tediosas e tendem à ocorrência de erros.

Sendo assim, Li (2005) afirma que as empresas e organizações envolvidas com a produção de software estão constantemente envolvidas em pesquisas que visam obter uma maior eficácia na execução dos testes de software antes de entregar a aplicação ao cliente. As pesquisas envolvem incluem identificar novas tecnologias com o objetivo de melhorar as formas de codificar scripts de teste e gerar casos de teste, além de também abranger o conceito de automação para testes.

Para Burns (2012) a automação de testes está em ascendência, ou seja, o seu uso está cada vez mais frequente devido a falta de tempo ou dinheiro para contratar profissionais de testes que verifiquem se o sistema está funcionando corretamente.

Automatizar testes é uma forma de exercitar e verificar o software desenvolvido sem que haja participação humana (Meszaros, 2007). Li (2005) afirma que o teste de software ao longo dos anos tem apontado para o uso cada vez mais frequente de ferramentas para automatizar os testes. O autor também destaca que a integração dessas ferramentas nos processos de testes contribui para que o ciclo de vida do projeto seja reduzido, assim como seu custo. Além disso, essas ferramentas apresentam uma maior eficácia, dessa forma elas encontram uma maior quantidade de falhas e erros no sistema em desenvolvimento. Com uma maior quantidade de erros encontrados e realizada a correção dos mesmos, a chance da aplicação chegar com falha no cliente é reduzida.

4.1 ESTRATÉGIAS DE AUTOMAÇÃO

As estratégias de automação visam simular o usuário interagindo com o sistema em desenvolvimento. Em geral, os profissionais de testes aprendem a usar duas estratégias para automatizar os testes funcionais de interface com o usuário: codificar scripts de testes ou fazer o uso de ferramentas de captura/reprodução para gravar as ações do usuário com a aplicação (Li, 2005).

O Selenium é um framework de testes muito utilizado por várias empresas, ele foi criado por Jason Huggins que identificou a necessidade de verificar o funcionamento dos softwares desenvolvidos em diversos sistemas operacionais e navegadores (Burns, 2012).

A escolha da organização por utilizar o Selenium para automatizar os testes funcionais de interface com o usuário se deve pelo fato dele ser gratuito e por possuir uma vasta documentação para estudo disponibilizada. Outro ponto importante é que o Selenium permite, para a codificação dos testes automatizados, o uso do Java, uma linguagem que já foi estudada por todos da equipe de testes durante a graduação e é conhecida em nível avançado pela maioria dos desenvolvedores da organização, dessa forma eles podem auxiliar na codificação dos testes automatizados, tirar dúvidas, etc.

O projeto do Selenium possibilita a criação de testes funcionais pelos testadores para guiar navegadores que suportem JavaScript. Baseando-se nas ferramentas para automatizar as interações do usuário com os navegadores do Selenium, descreveremos a seguir as estratégias de automação dos testes funcionais de interface com o usuário.

4.1.1 GRAVAÇÃO

Conforme já definido pela própria organização a ferramenta de testes usada é Selenium IDE para este tipo de estratégia. O Selenium IDE (Integrated Development Enviroment) é uma extensão do Firefox que permite aos profissionais de teste realizem a gravação os passos realizados ao executar as funcionalidades de uma aplicação web (Gundecha, 2013, Burns, 2012, Kovalenko, 2014, Prasad, 2015). Com os passos gravados é possível reproduzi-los novamente em possíveis testes futuros e verificar se as funcionalidades da aplicação está funcionando corretamente.

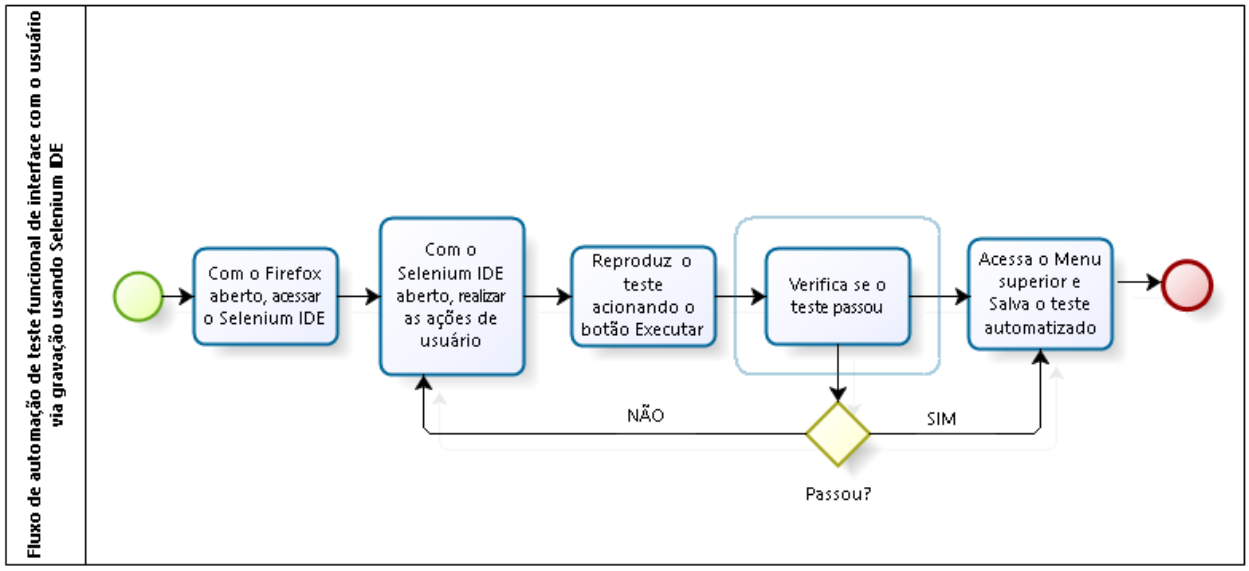
Dentre as vantagens da ferramenta podemos destacar a facilidade de uso por não exigir um nível alto de conhecimento técnico ou experiência (Li, 2005, Kovalenko, 2014). Além disso é uma ferramenta de uso rápido e intuitivo, basta acionar o botão de gravar e utilizar a aplicação que deseja testar como um usuário comum que a ferramenta identificará os elementos acessados na interface e gravará suas ações. Já as desvantagens ficam por conta inflexibilidade dos testes criados devido à limitação de poder usar apenas o Firefox como browser para gravar e reproduzir os testes. Existe também a dificuldade desses testes automatizados se adaptarem às mudanças, Alégroth (2015) confirma isso dizendo que a técnica de gravar e reproduzir as ações do usuário são afetadas quando ocorrem mudanças na interface ou no código do sistema. Essa dificuldade de se adaptar

às mudanças compromete o reuso desses testes automatizados gravados, segundo Li (2005), de forma geral, o nível de reuso dos testes funcionais de interface com o usuário automatizados via gravação são baixos.

4.1.1.1 CRIANDO UM TESTE AUTOMATIZADO VIA GRAVAÇÃO

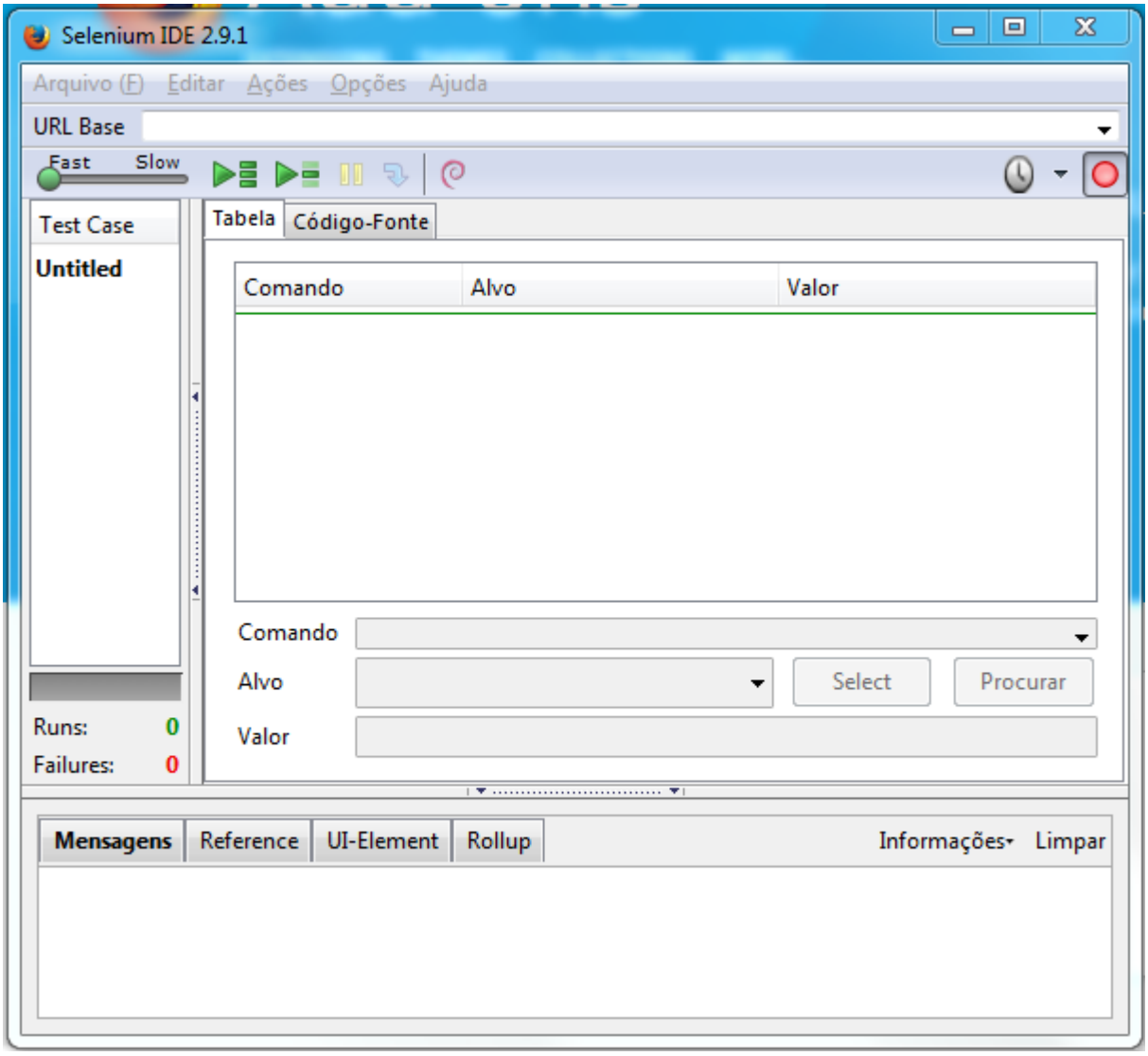
Após as devidas instalações e configurações do ambiente tecnológico no qual se insere, a ferramenta está pronta para ser usada. A Figura 3 apresenta abaixo, mostra o processo para a criação de um teste funcional de interface com o usuário automatizado utilizando a ferramenta Selenium IDE.

Figura 5: Fluxo de gravação utilizando Selenium IDE. Fonte Autor.



Após acessar o Selenium IDE clicando no ícone presente ao lado da barra de pesquisa do navegador, é apresenta a ferramenta conforme a figura abaixo.

Figura 6: Interface gráfica Selenium IDE. Fonte Autor.



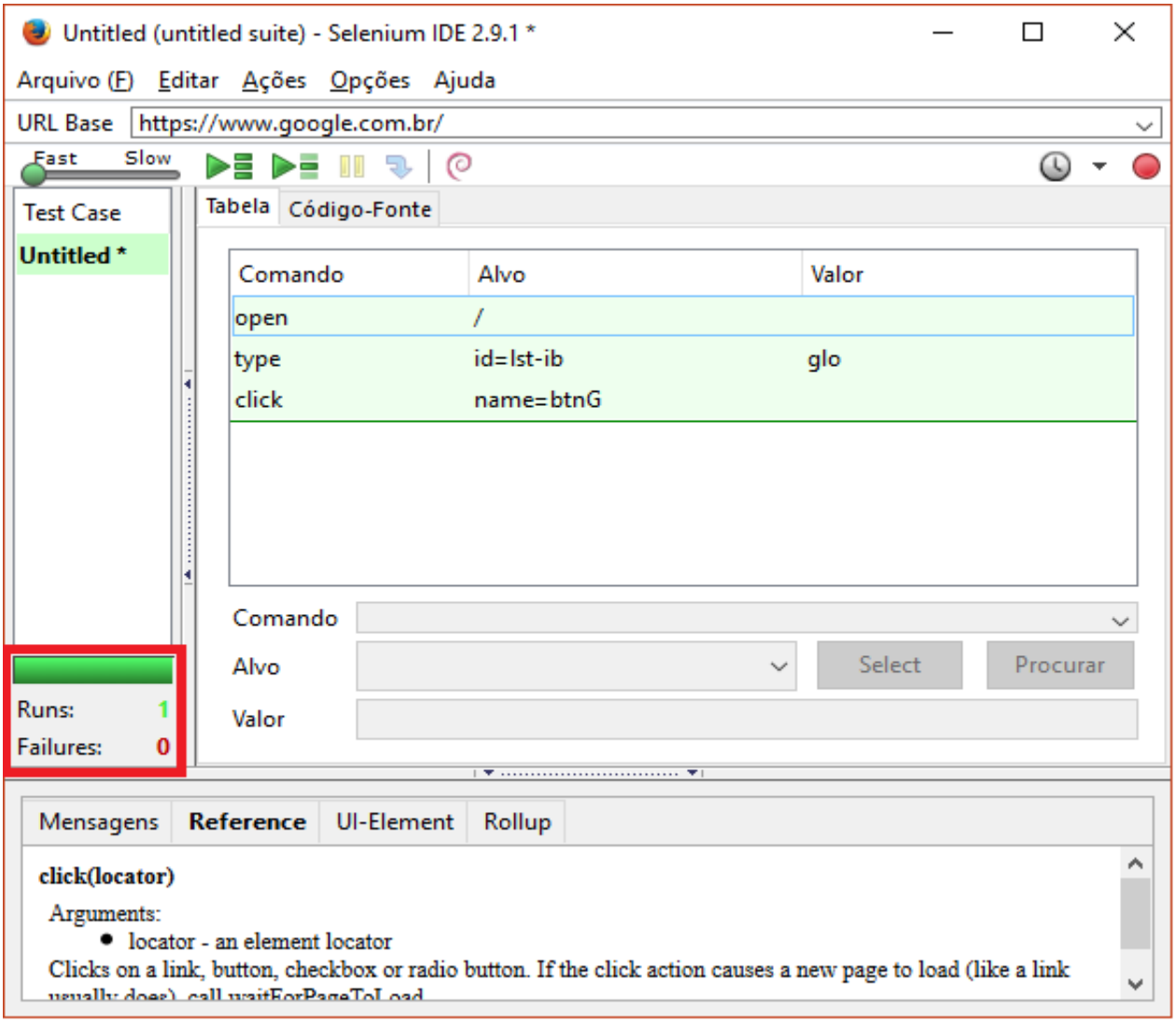
Com o Selenium IDE aberto basta seguir os passos descritos abaixo pare criar um teste funcional de interface com o usuário via gravação:

1. O Selenium IDE já é aberto com o botão Record ativo, dessa forma a automação já está pronta para iniciar, e a partir daí todas as ações do usuário com o navegador do Firefox serão gravadas. Para finalizar o teste basta acionar o botão Record novamente.
2. Depois de finalizado o teste, todas as ações capturadas durante a gravação serão apresentadas na Tabela localizada no centro da ferramenta. Além disso, a primeira url acessa será apresentada no campo URL Base localizado na parte superior da ferramenta.
3. Para executar o teste automatizado, basta selecioná-lo na caixa Test Case localizada à

esquerda da ferramenta e acionar o botão de Execução . Caso vários testes automatizados tenham sido criados e é necessário que todos sejam executados basta acionar o botão de Execução Test Suíte .

4. Após realizar a execução, a barra apresentada abaixo da caixa Test Case indica se o teste passou ou apresentou falha. Em caso de sucesso a barra é apresentada na cor verde, em caso de falha na cor vermelha.

Figure 7: Indicador de sucesso/falha Selenium IDE. Fonte Autor.



4.1.2 CODIFICAÇÃO

Para a técnica de codificação a equipe de teste utiliza o Selenium Webdriver para implementar os testes funcionais automatizados. O Selenium Webdriver é uma API que oferece

aos profissionais de teste diversos meios de codificar a interação dos usuários com a tela por meio da identificação dos componentes da interface gráfica do navegador (Gundecha, 2012, Zhan, 2015, Prasad, 2015, Burns, 2012).

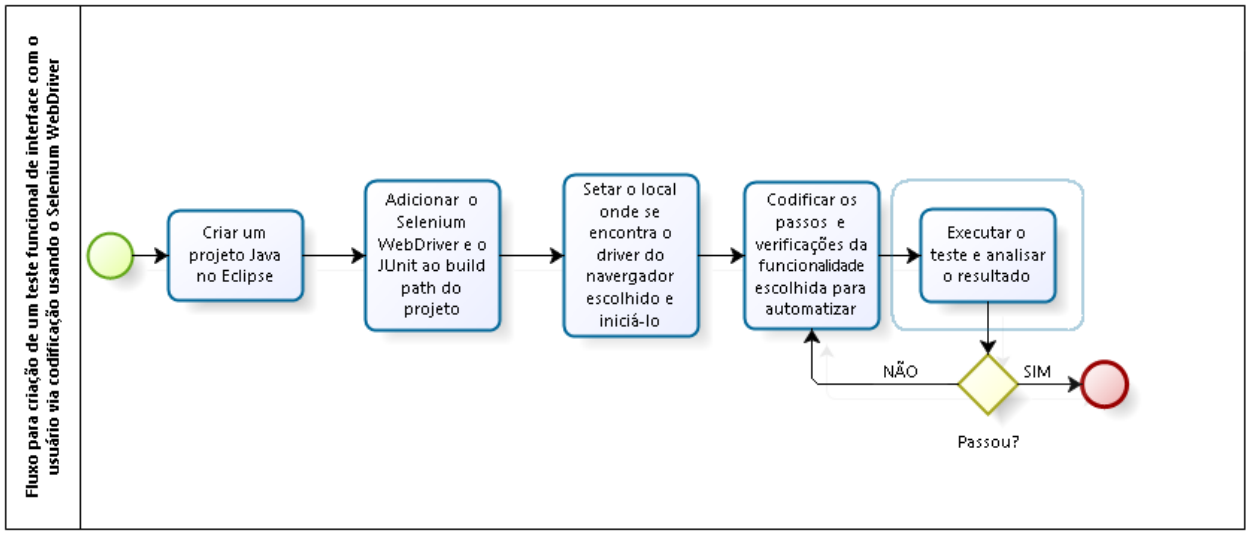
Os componentes da tela são identificados por atributos do HTML como o ID ou por CSS e XPATH (Gundecha, 2012, Zhan, 2015, Prasad, 2015). O CSS (*Cascading Style Sheets*) é uma linguagem voltada para a forma ou o estilo como os componentes de tela são apresentados em um arquivo escrito em linguagem de marcação, como o HTML. Já o XPATH é uma linguagem de caminho XML e está voltado para a hierarquia dos itens de marcação contidos no HTML, dessa forma ele monta o caminho(PATH) partindo do item de mais alto nível até chegar ao elemento que deseja identificar. A verificação é feita utilizando frameworks de teste como JUnit, NUnit ou TestNG, o framework a ser usado dependerá da linguagem utilizada para codificar os scripts de testes, já que o Selenium WebDriver suporta diversas linguagens de programação como Java, C#, Ruby, Python, VisualBasic.NET etc (Zhan, 2015).

Entre as vantagens de automatizar os testes funcionais de interface com o usuário via codificação está a sua melhor resposta as mudanças que afetem os testes, já que o código pode ser ajustado para que os testes automatizados voltem a funcionar corretamente, o que consequentemente aumenta seu poder de reuso (Li, 2005). Outra importante vantagem segundo Gundecha (2012) é a possibilidade da execução dos testes em diversos browsers. A desvantagem é que por ser feito via codificação, esse meio de automatizar testes funcionais de interface com o usuário exige um bom conhecimento em lógica de programação e da linguagem que será utilizada, além da necessidade de uma configuração antes de se iniciar a codificação e execução dos testes automatizados.

4.1.2.1 CRIANDO UM TESTE AUTOMATIZADO VIA CODIFICAÇÃO

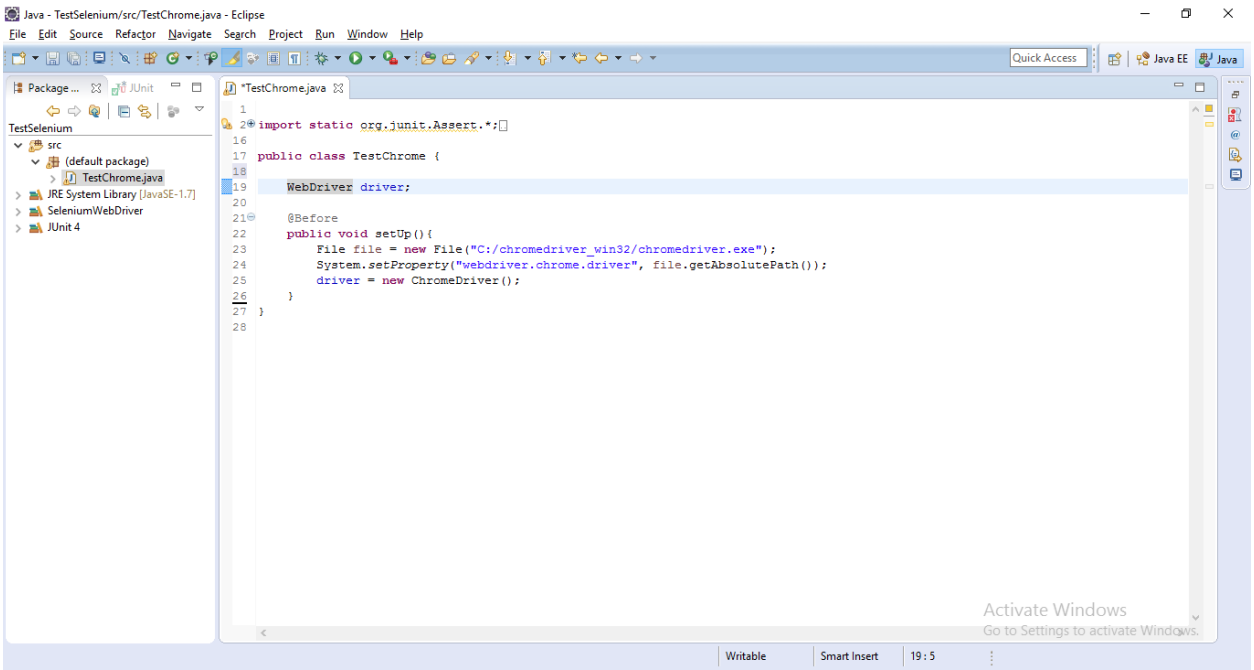
Baseando-se nos pontos citados no item anterior e nos próximos passos para a criação de um teste automatizado, definimos um processo para tal que está representado na figura apresentada abaixo.

Figure 8: Fluxo de codificação usando Selenium WebDriver. Fonte Autor.



Primeiramente iremos definir o caminho para o driver do nosso navegador escolhido para executar os testes no método setUp(), além disso nesse método também é o método abre a janela do navegador escolhido. A figura abaixo mostra como isso é feito para navegador Chrome.

Figure 9: Passo 1 codificação. Fonte Autor.



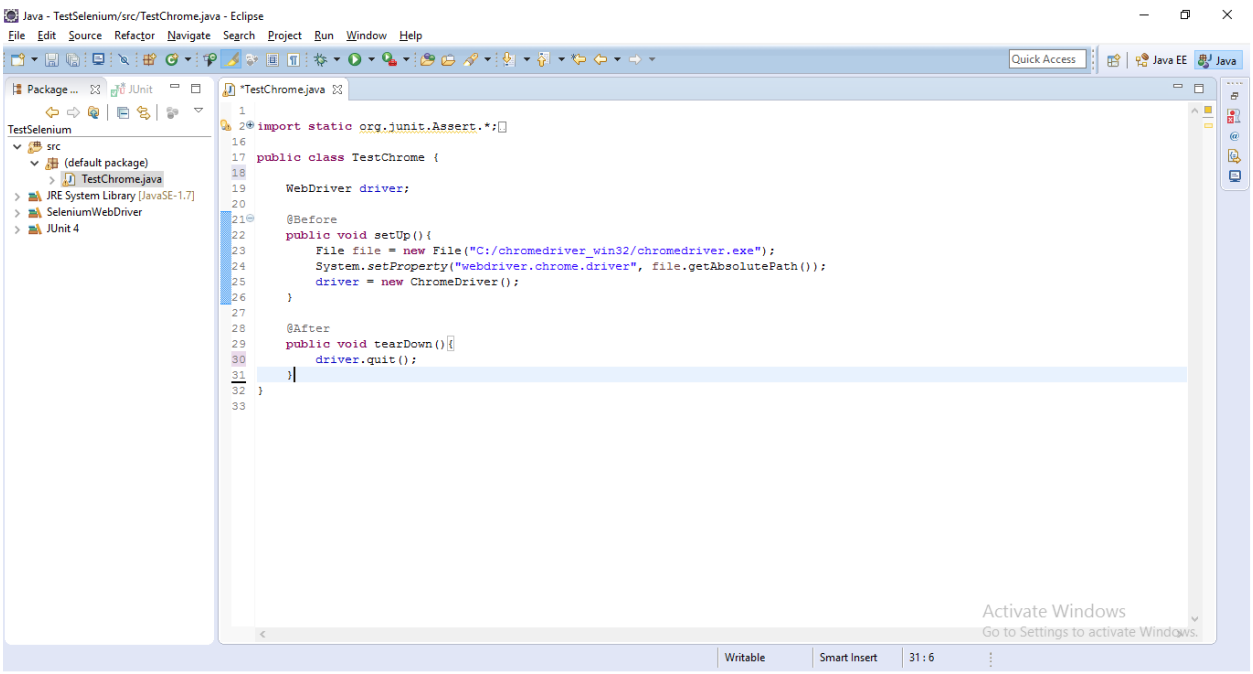
O @Before apresentado antes do método é uma marcação do JUnit que define o que será feito antes de executar o teste, ou seja, antes de iniciar o teste ele irá executar o método setUp() que irá abrir o navegador Chrome.

Em seguida, definiremos o que será feito após o teste ser encerrado. Para isso será utilizada

a marcação @After do JUnit que estará antes do método tearDown(), esse método basicamente fechará o navegador Chrome após a execução do testes.

As marcações @Before, @After e @Test, que serão apresentadas a seguir, são recursos que JUnit disponibiliza para uma melhor organização do código dos testes automatizados. No caso do TestNG são utilizadas as marcações @BeforeTest, @AfterTest e @Test.

Figure 10: Passo 2 codificação. Fonte Auto.

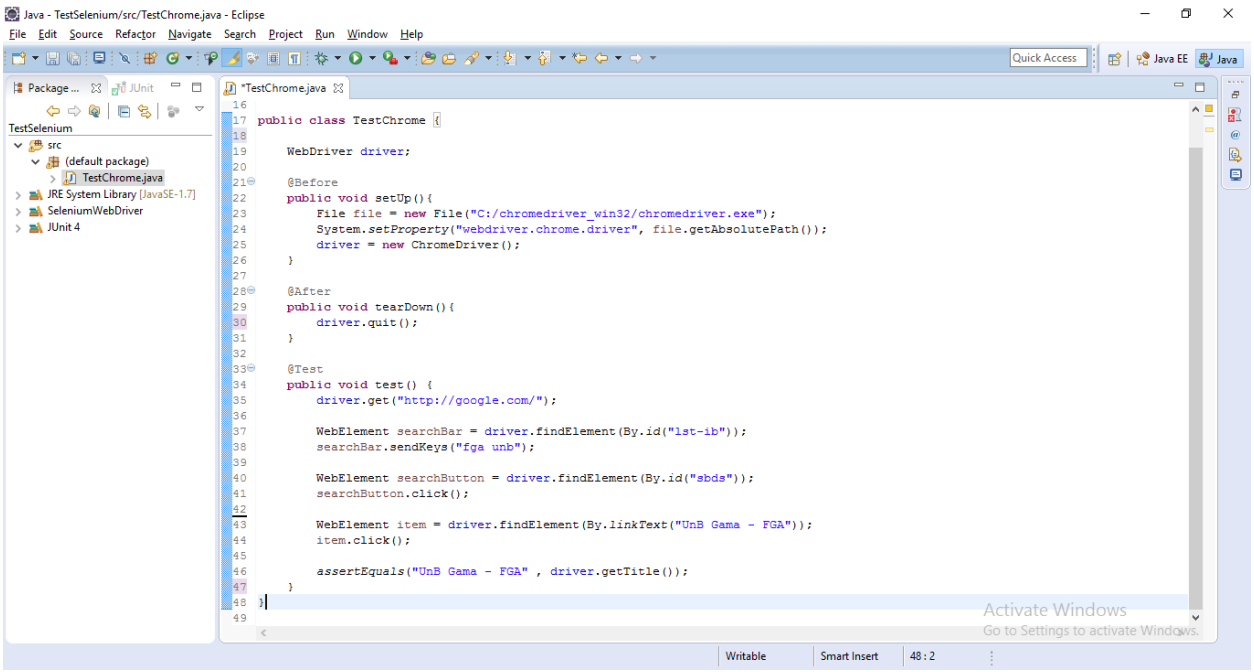


Após definir deve ser feito antes (@Before) e depois (@After) de rodar o teste automatizado, podemos seguir para a codificação do mesmo. O teste descrito neste exemplo utiliza a marcação @Test do Junit e executa os seguintes passos:

1. Acessa o site do Google;
2. Identifica a barra de pesquisa pelo id, e digita o texto “fga unb”;
3. Identifica o botão de pesquisa do site do Google e clica;
4. Identifica no resultado da pesquisa realizada o link com o texto “UnB Gama - FGA” e clica;
5. Em seguida é verificado se o título da página apresentada corresponde ao título esperado.

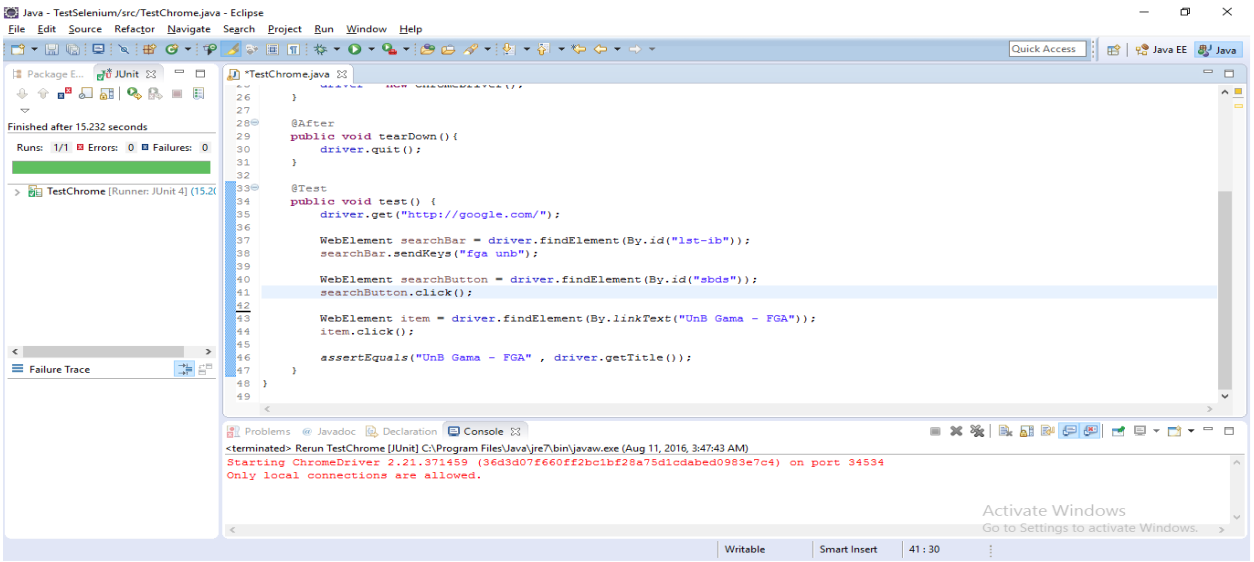
Os passos acima estão representados em forma de código na figura apresentada a seguir:

Figure 11: Passo 3 codificação. Fonte Auto.



Para realizar a execução do teste automatizado, basta salvar o teste, clicar com o botão direito em cima da classe Java criada, ir até “Run As” e clicar um JUnit Test, caso esteja utilizando o TestNG clicar em TestNG Test. O teste será executado e na aba do JUnit ou TestNG será apresentado o resultado, como é mostrado na figura abaixo:

Figure 12: Passo 4 codificação. Fonte Auto.



4.1.3 LINGUAGEM NATURAL

Ao tratar sobre a automação de testes funcionais utilizando linguagem natural é necessário

um melhor entendimento sobre o BDD (Behavior Driven Development). Primeiramente, o BDD é um método que favorece os testes funcionais por identificar se os requisitos funcionais estão sendo correspondidos no sistema em si (Carvalho, 2010). Segundo Häser(2016), a ideia do BDD é transformar em uma linguagem formal ou conhecida por todos os membros da equipe o conteúdo técnico dos testes e os seus retornos.

Num primeiro momento, ao utilizar a técnica do BDD deve ser feita a descrição dos requisitos que se deseja testar em linguagem formal que se deseja testar (Tavares, 2010). Dessa forma os requisitos devem ser transformados em cenários que simulam o usuário utilizando a aplicação, em seguida e baseando-se nessa história gera-se o cenário. Exemplo:

Cenário: Cadastrando um contrato

Como um gerente

Quero adicionar um contrato assinado pela empresa

Para que os usuários envolvidos tenham acesso ao conteúdo do mesmo

Baseando-se na descrição do requisito acima, Tavares(2010) afirma que podem ser identificadas as ações do usuário do sistema, assim como as respostas que ela devolve ao usuário. Com isso, podemos descrever essas ações e respostas em linguagem natural utilizando o Give-When-Then (Dado-Quando-Então), exemplo:

Cenário: O gerente deseja adicionar um contrato e disponibilizá-lo aos participantes

Dado que eu acesso a minha área de usuário

Quando seleciono a opção de menu adicionar contratos

Então O gerente cria o contrato e adiciona os participantes envolvidos

Além de trazer a ideia do uso da linguagem natural para descrever os requisitos, o BDD também tem como um de seus principais fundamentos o uso da automação (Tavares, 2010). Para a realização da automação utilizando linguagem natural é feito o uso de ferramentas que permitem que uma sentença escrita em linguagem natural chame e execute o pedaço do código que corresponda a ação descrita nesta sentença. Entre as ferramentas que podem ser utilizada para a automação de testes em linguagem natural está o Cucumber, pois ele é uma ferramenta que permite essa interação entre os passos descritos em linguagem natural e o código a ser executado. Além disso é possível integrá-lo com a IDE do Eclipse e com o Selenium.

Ao utilizar o Cucumber para automatizar testes é necessário o uso do Gherkins, que é a linguagem que o Cucumber entende. O Gherkins permite que seja descrito as ações do usuário na

aplicação com o uso de palavras chave Given-When-Then (Rose, 2015). Dessa forma são criadas features as quais descrevem os possíveis cenários da aplicação de acordo com as funcionalidades da mesma. Exemplo:

Feature: Testar a funcionalidade de criar contratos e disponibilizá-los

Cenário: O gerente deseja adicionar um contrato e disponibilizá-lo aos participantes

Given O gerente acessa sua área de usuário

When O gerente seleciona a opção de menu adicionar contratos

Then O gerente cria o contrato e adiciona os participantes envolvidos

Descrita as features, ao executar o Cucumber o mesmo lê as features e gera o código para teste, transformando cada sentença Given-When-Then em um método conforme o exemplo abaixo:

```
@Given("^O gerente acessa sua área de usuário")
public void O_gerente_acessa_sua_area_de_usuario(){
}
@When("^O gerente seleciona a opção de menu adicionar contratos")
public void O_gerente_seleciona_a_opcao_de_menu_adicionar_contratos(){
}
@Then("^O gerente cria o contrato e adiciona os participantes envolvidos")
public void O_gerente_cria_o_contrato_e_adiciona_os_participantes_envolvidos{
}
```

Após a geração dos métodos pelo Cucumber, são inseridos os códigos para chamar os comandos do selenium, dessa forma podemos afirmar que a automação por linguagem natural tem como base o uso da codificação para a implementação dos seus testes automatizados com o diferencial de reduzir a distância entre a parte de negócios e a parte de tecnológica, aproximando os cenários possíveis dos testes.

5 APLICAÇÃO, RESULTADOS E AVALIAÇÕES

5.1 SURVEY

O termo *survey* é traduzido como "levantamento de dados". Pasquali (1999) cita Fink & Kosecoff (1985) ao definir *survey* como uma técnica de obter opiniões, informações e pensamentos de um conjunto de pessoas. Segundo o autor, o *survey* se utiliza do questionário para atingir seu objetivo.

Segundo Pasquali (1999), antes de elaborar um questionário primeiramente definimos aonde queremos chegar como esse levantamento de dados. Com base no objetivo chegamos aos conceitos e a população-alvo, que devem ser trabalhados simultaneamente pois eles são determinantes no momento de definir as perguntas que estarão no questionário e a forma como este questionário será aplicado. O autor também acrescenta que o tamanho da população-alvo que será submetida ao questionário tem impacto direto em questões financeiras, no tempo gasto e também nos recursos necessários, pois será necessário guardar os dados obtidos e analisá-los. Além disso o ambiente físico usado para coletar os dados tem ligação direta com o acesso aos respondentes.

Pasquali (1999) cita que questões socioculturais também devem ser levadas em conta ao construir um questionário. Deve haver um limite ao tentar obter informações pessoais dos respondentes. Itens como o interesse dos respondentes pelo assunto e a instituição que o pesquisador está ligado podem afetar a aplicação do questionário. Segundo o autor, a opinião singular do respondente deve estar à frente da opinião pública, assim teremos respondentes que estão aptos a responder sobre os conceitos definidos.

Pasquali (1999) baseando-se nos estudos de Dillman (1978), estabelece 3 passos que o pesquisador poderia fazer que influenciariam no sucesso do *survey*, são elas: recompensar o respondente, reduzir o custo de responder e estabelecer confiança. Primeiramente deve-se estabelecer confiança buscando atrair o interesse do respondente, informando-o da relevância da pesquisa e também da própria opinião dele. Em relação à redução do custo de responder deve-se levar em conta que o interesse do respondente tem de ser constante. Além disso a tarefa de responder um survey deve ser rápida e fácil, ou seja, não deve exigir um grande gasto mental e físico do respondente. Por fim chegamos ao ponto da recompensa, aqui mais uma vez é reforçada a importância da contribuição do respondente, aqui vale ressaltar que passar os resultados para o

cliente após a análise dos dados obtidos também é uma forma de recompensá-lo.

Como dito no parágrafo anterior a atividade de responder um *survey* deve ser breve e fácil, e é nesse ponto que está a dificuldade. Segundo Pasquali (1999), uma técnica que ajuda a atingir esse objetivo é ir ao longo do *survey* afunilando o tema, ou seja, deve-se partir dos conceitos mais amplos aos mais específicos, de forma que os conceitos mais amplos sirvam de insumo para o respondente formar sua opinião sobre os itens específicos. Além disso o autor cita as 3 regras gerais de Sudman (1982), e dentre essas regras encontramos o raciocínio de que as perguntas elaboradas devem trazer a certeza que suas respostas são imprescindíveis para a pesquisa.

Ao final o autor também destaca que a formalidade da linguagem usada no *survey* se dá levando-se em consideração o entendimento da população estudada, de forma que não devem ser informais a ponto de serem usadas gírias, por exemplo. Além disso, sobre perguntas abertas e fechadas, o autor baseia-se em estudos de outros autores para afirmar que as perguntas abertas são geralmente usadas em pesquisas exploratórias, onde não se sabe o nível de discrepância que as respostas podem apresentar entre si, enquanto que as perguntas fechadas são usadas quando a pesquisa tem de ser feita rapidamente em uma população alvo considerada como grande.

5.1.1 CONSIDERAÇÕES TEMÁTICAS

Pasquali (1999) também ressalta alguns itens a serem observados na construção de questionários. Dentre eles:

5.1.1.1 GRAU DE AMEAÇA DE ITENS

Aqui o objetivo é realizar perguntas que digam ao pesquisador se o respondente está se sentindo à vontade com o assunto abordado. O objetivo é evitar que o respondente se sinta embaraçado por não saber algum item abordado no *survey*. Um respondente que não seja capaz de assumir que não sabe uma determinada pergunta pode chegar ao ponto respondente dar qualquer resposta, o que pode prejudicar o resultado do *survey* (Pasquali, 1999). O questionário apresentado procura identificar nos respondentes o grau de conhecimento que eles detêm da área através das respostas e justificativas apresentadas por eles. Como os respondentes não são leigos no assunto abordado pelo questionário por estarem inseridos em uma equipe dedicada à área de testes, não é esperado este tipo de desconforto por parte deles, logo não foram feitas perguntas relacionadas a essa seção.

5.1.1.2 ITENS PARA AVALIAR CONHECIMENTO

Mesmo assim, em um *survey* é de extrema importância realizar perguntas de forma a mensurar o nível de conhecimento do respondente, procurando saber seu nível de escolaridade. Essa informação auxilia na seleção das perguntas que virão em seguida. Sendo assim, logo no início do questionário é questionado quais papéis o respondente já exerceu na área de testes e por quanto tempo ele exerceu esse(s) papel(éis).

5.1.1.3 ITENS PARA AFERIR ATITUDES E OPNIÕES

Nesta abordagem são elaboradas perguntas de forma a extrair a opinião ou reação do respondente após incluí-lo em uma situação qualquer, como no caso deste trabalho as perguntas sobre as suas opiniões sobre a importância em se automatizar este tipo de teste e por qual estratégia seria melhor conduzir a automatização.

5.1.1.4 ITEM PARA OBTER INFORMAÇÕES FACTUAL

Perguntas para obter informações pessoais com o objetivo de caracterizar a população-alvo estudada. Essas perguntas são apresentas ao final do *survey* de forma bem sucinta e deve-se deixar claro ao respondente que seus dados serão mantidos em sigilo.

5.1.2 CONSIDERAÇÕES SOBRE USO DE ESCALAS

Aqui serão abordadas formas de considerar as respostas, para que possam ser mensuradas ou definidas numericamente. Para isso existem algumas escalas que podem ser usadas em questionários. Dentre elas:

5.1.2.1 ESCALA ORDINAL

Escala utilizada para identificar dados do respondente usando números, como por exemplo:

Ex: Qual o seu nível de escolaridade?

Fundamental incompleto.....1
Fundamental completo.....2
Médio incompleto.....3
Médio completo.....4
Superior incompleto.....5
Superior completo.....6

Aqui também podem ser feitas perguntas de forma a hierarquizar as respostas dos respondentes. Por exemplo, em uma pergunta sobre canais de TV, pede-se que o usuário enumere os canais de TV contidos nas respostas por ordem de preferência.

5.1.2.2 ESCALA INTERVALAR

Determina-se um intervalo numérico para o respondente definir a sua resposta. Por exemplo, pode-se pedir para o usuário definir um número dentro de um intervalo que diga o quanto ele concorda com um projeto de lei.

5.1.2.3 ESCALA DE RAZÃO

Aqui o uso dos números é feito de maneira direta. Por exemplo, a quantidade de filhos de uma pessoa, a idade, números de quartos da casa, entre outros.

5.1.2.4 ESCALA LIKERT

Escala que busca levantar opinião de forma a evitar o uso de apenas duas opções contrárias como sim e não. Aqui são inseridos termos entre as opções contrárias que dão mais opções ao respondente o que pode tornar sua opinião mais fidedigna.

Nessa pesquisa aplicamos essa escala nas perguntas de 1 a 5 do questionário para permitir uma flexibilidade maior no julgamento feito pelos entrevistados das afirmações feitas e também nas suas justificativas. Ao invés de usarmos as respostas concordo e discordo, utilizamos discordo completamente, discordo parcialmente, não tenho opinião sobre o assunto, concordo parcialmente e concordo completamente.

5.1.3 OBJETIVO

Neste TCC será aplicado um questionário e entrevistas junto aos profissionais de teste e qualidade da empresa envolvida. O questionário visa obter opiniões sobre a importância dos testes em geral e dos testes funcionais, além de abordar a forma como os testes funcionais são executados (manual e automatizada). Para o tema de automação procura-se obter a preferência desses profissionais entre as estratégias de automatizar via código ou via gravação.

5.2 EQUIPE DE TESTE

A equipe de teste da organização estudada neste TCC é formada por 1 gerente de testes, 5

analistas de testes e 4 testadores.

5.2.1 ATIVIDADES DE TESTE

As atividades de execução são feitas pelos testadores, podendo os analistas participarem das execuções caso os testadores estejam sobrecarregados e a finalização da execução seja uma prioridade. Os testadores antes de iniciarem a execução de uma (OS)-Ordem de Serviço que já foi testada, devem verificar os itens ou erros que foram reportados na execução anterior.

Os analistas de teste validam as documentações elaboradas pela equipe de requisitos e elaboram os casos de teste que serão utilizados pelos testadores no momento da execução. Dependendo do nível de experiência do testador, ele também pode realizar a validação das documentações.

O gerente de testes organiza e distribui a equipe de acordo com as demandas que chegam para teste, ele que define o que será executado, ou seja, qual OS será priorizada. Além disso avalia os resultados obtidos nas execuções e o rendimento dos testadores e analistas de teste. Aqui vale ressaltar que o gerente de testes está no mesmo patamar do gerente de projetos, dessa forma a opinião dos dois tem o mesmo peso em momentos de decisões conjuntas.

5.3 QUESTIONÁRIO

O questionário elaborado durante esse TCC está apresentado no Apêndice A. Primeiramente ele apresenta uma introdução que informa aos respondentes os objetivos do TCC, ressaltando que os dados coletados pelo questionário aplicado no TCC1 terão grande relevância para o TCC2. Em seguida é informado que não será necessário que o respondente se identifique. Ao final busca-se caracterizar o entrevistado, sendo assim é questionado qual(is) o(s) perfil(is) da área de testes que ele já trabalhou, além de informar o tempo de experiência em cada perfil selecionado.

Em seguida já são as questões conceituais em si, elas serão descritas separadamente a seguir assim como seus objetivos. Vale lembrar que todas as questões exigem a justificativa dos entrevistados.

5.3.1 QUESTÕES UTILIZADAS - OBJETIVOS

- Na questão 1 busca-se identificar o nível de concordância ou discordância dos

entrevistados do fato das atividades de teste serem consideradas críticas, ou seja, elas exigem um alto nível de atenção e dedicação dos envolvidos.

- A questão 2 o objetivo é identificar se os entrevistados consideram os testes funcionais uma atividade importante para verificar a consistência dos comportamentos do sistema, ou seja, se eles estão respondendo da forma que é esperada.
- A questão 3 trata de obter a nível de credibilidade dos entrevistados em relação ao ganho de tempo e mão de obra ao automatizar os testes funcionais e executá-los, ao invés de realizar uma execução manual.
- A questão 4 procura mensurar o quanto os profissionais de testes acreditam que a execução manual dos testes funcionais pode prejudicar na verificação do sistema devido às falhas humanas ocorridas, por exemplo, no momento de seguir os passos definidos ou de analisar o resultado obtido em relação ao resultado esperado.
- A questão 5 foca nas execuções de funcionalidades que devem ser repetidas diversas vezes pelo testador. Ela procura obter o nível de credibilidade da informação de que a repetição de uma mesma funcionalidade seguidamente incrementa a probabilidade de acontecer uma falha humana durante a execução dos testes funcionais.
- A questão 6 é a única questão que não obedece a escala Likert, pois nela só há 2 duas opções de resposta. Após uma breve introdução sobre as estratégias de automação dos testes funcionais (gravação e codificação) os entrevistados devem selecionar sua preferência entre elas e em seguida justificar sua escolha.

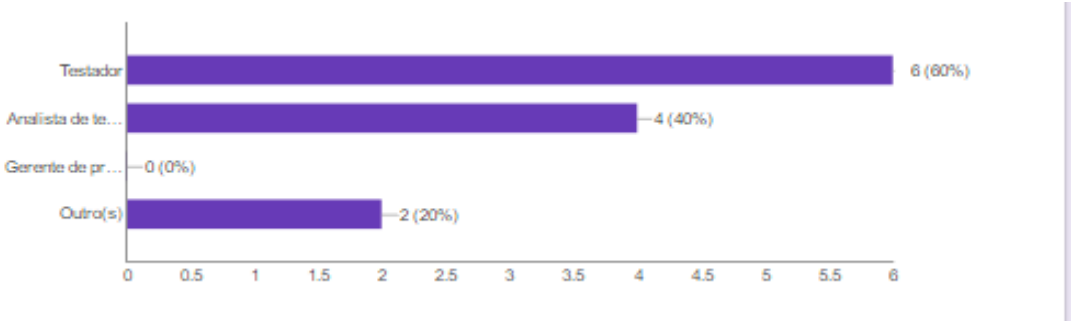
5.3.2 RESULTADOS OBTIDOS

O questionário descrito foi respondido por 10 pessoas, os perfis dos entrevistados e as respostas obtidas serão apresentados nos subitens a seguir ressaltando que antes das questões propriamente ditas foi feita um mapeamento dos perfis e tempo de experiência dos respondentes ao questionário. A seguir são apresentadas as perguntas utilizadas, seus resultados e, quando existente, um detalhamento ou discussão sobre itens acrescentados pelos respondentes sobre as suas respostas.

5.3.2.1 QUANTO AOS PERFIS E TEMPO DE EXPERIÊNCIA

Informe o tempo que você trabalhou no perfil selecionado. Caso você tenha trabalhado em mais de um perfil, diga o tempo que atuou em cada um deles. (Ex.: Testador: 2 anos e 3 meses)

Figura 13: Resultado dos perfis dos entrevistados. Fonte autor.



A tabela abaixo sumariza as respostas obtidas neste levantamento.

Tabela 1: Análise tabular dos perfis e do tempo de experiência. Fonte autor.

	Quantidade	%
Testador	4	40
Analista de testes	3	30
Testador + Analista de Requisitos	1	10
Testador + Analista de Testes	1	10
Analista de Qualidade	1	10

Experiência média	
Testador:	1 ano e 3 meses
Analista de Testes:	3 anos e 7 meses
Analista de Requisitos:	11 meses
Analista de Qualidade:	1 ano e 6 meses

5.3.2.2 QUANTO À QUESTÃO 1

No desenvolvimento de software os testes constituem uma atividade crítica.

Figura 14: Resultado da questão 1. Fonte autor.

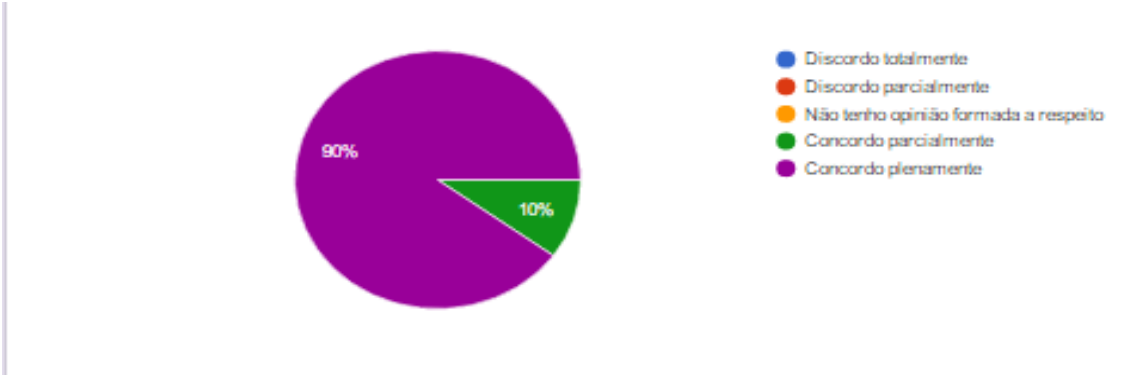


Tabela 2: Análise tabular da questão 1. Fonte autor.

	Quantidade	%
Concordo plenamente	9	90
Concordo parcialmente	1	10

Todos destacaram a fase de testes importante e crucial para o projeto de software. O contribuinte que marcou que concorda parcialmente é testador a 9 meses e não explicou direito o motivo de ter marcado parcialmente, sua justificativa daria para ter marcado "Concordo plenamente", porém o mesmo destacou que não é possível obter um software 100% livre de bugs.

Justificativa: "É extremamente necessário que haja uma(s) fase(s) de teste na maioria dos projetos desenvolvidos hoje, para que o sistema entregue esteja com o mínimo de defeitos possíveis (apesar de nunca algo ser 100% livre de bugs)."

5.3.2.3 QUANTO À QUESTÃO 2

Os testes funcionais têm alta importância para avaliar se as funcionalidades do software estão funcionando corretamente.

Figura 15: Resultado da questão 2. Fonte autor.

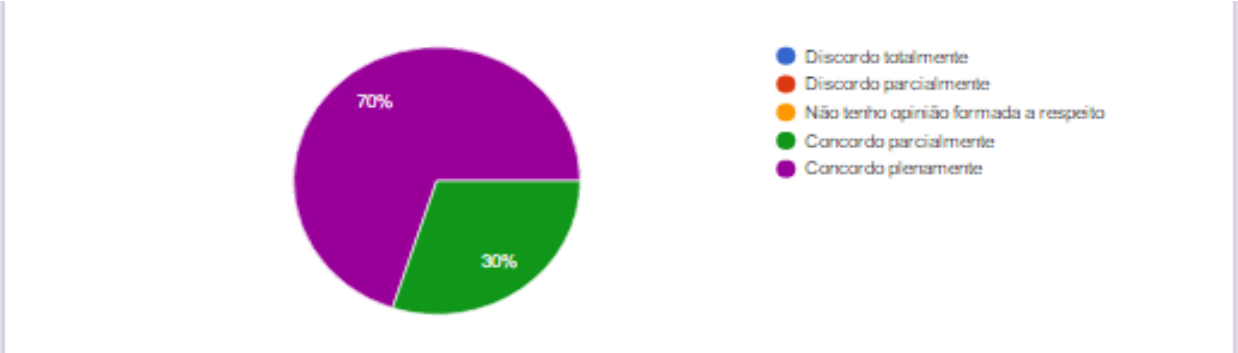


Tabela 3: Análise tabular da questão 2. Fonte autor.

	Quantidade	%
Concordo plenamente	7	70
Concordo parcialmente	3	30

Todos concordaram que os testes funcionais têm grande importância para avaliar se as funcionalidades do software estão funcionando corretamente, porém 3 concordaram parcialmente.

O primeiro que concordou parcialmente é analista de testes e afirmou que sem os testes de regressão, aceitação e regressão (quando necessário) não é possível visualizar se as funcionalidades estão funcionando corretamente

Justificativa: "É importante, mas sem o teste de sistema, aceitação, integração e quando necessário,

o teste de regressão, o teste funcional não permite visualizar a garantia de sucesso das operações do sistema. ”

O segundo é testador a 1 ano e 4 meses deu uma justificativa que daria para marcar como “Concordo plenamente”, porém ele destacou que os testes funcionais não servem só para verificar o funcionamento correto do sistema, mas também avaliar se está de acordo com as necessidades do cliente.

Justificativa: “Não só se estão funcionando corretamente mas também se aquela funcionalidade está de acordo com as necessidades do cliente. ”

O terceiro é testador a 1 ano e também deu uma justificativa que complementa a afirmação feita logo daria para ter marcado como “Concordo plenamente”.

Justificativa: “Eles que comparam os requisitos levantados com o que foi desenvolvido. ”

5.3.2.4 QUANTO À QUESTÃO 3

Os testes funcionais de interface do usuário feitos manualmente exigem uma quantidade de tempo e mão de obra maior do que quando eles são feitos de forma automatizada.

Figura 16: Resultado da questão 3. Fonte autor.

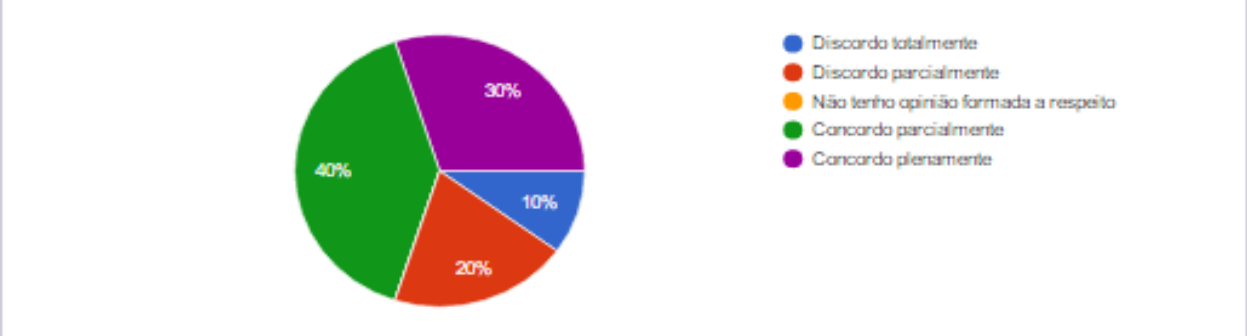


Tabela 4: Análise tabular da questão 3. Fonte autor.

	Quantidade	%
Concordo plenamente	4	40
Concordo parcialmente	3	30
Discordo Parcialmente	2	20
Discordo Totalmente	1	10

Três marcaram a resposta como “Concordo Parcialmente” e destacaram que:

- Se os testes tiverem que ser executados poucas vezes é melhor realizar a execução manualmente, já que neste caso a automação pode ter alto custo. (Analista de testes: 9 anos)

Justificativa: “Nem sempre. Depende da estratégia utilizada, assim como da

quantidade de vezes que tais testes tiverem que ser executados. Caso sejam poucas, o custo de programar a automação mais a execução irão ser mais altos do que executar tais testes manualmente. ”

- Se a interface sofrer muitas alterações a manutenção do código de automação demandará evoluções que também custarão tempo. (Testador: 9 meses, Analista de Requisitos: 11 meses)

  Justificativa: “Concordo plenamente pois o ser humano é falho. Uma máquina programada para realizar atividades sistêmicas executa de forma mais eficiente que o ser humano que pode sofrer as mais diversas influências externas, que não ocorreriam com uma máquina. ”

- Este destaca que tanto os testes automatizados quanto os testes manuais demandam tempo e que deve ser avaliado quando aplicar um outro. (Analista de testes: 7 anos e 4 meses)

  Justificativa: “Os testes manuais e automatizados de interface oneram uma quantidade de tempo significativa, no entanto o tempo gasto no teste manual é pouco variável em caso de mudanças, já o teste automatizado para a sua elaboração o tempo gasto é superior ao manual e sua manutenção pode ser inviável a depender das alterações sofridas em interface pelo tempo consumido em suas alterações, portanto deve-se avaliar muito bem quais funcionalidade e em qual momento a automatização deve ser aplicada.”

Dois marcaram como “Discordo parcialmente” e destacaram que:

- Dependendo do teste é mais prático/rápido realiza-lo à mão do que automatizá-lo. (Testador: 9 meses

  Justificativa: “Dependendo do teste pode ser mais prático/rápido realizar a mão do que criar um sistema automatizado para o mesmo. ”

- Os testes manuais têm custo e tempo maior que os testes automatizados a partir do momento que é necessário reexecutar e ajustar os testes. (Analista de Teste 3 anos e 6 meses, Testador 2 anos e 6 meses)

  Justificativa: “Os testes manuais têm o custo e tempo menor em relação aos testes automatizados, no entanto essa relação tende a inverter a medida que novas necessidades de se reexecutar e ajustar testes são realizadas”

Um marcou que “Discorda Totalmente” a automação não é vantajosa para todas as aplicações.

Justificativa: “Nem toda aplicação se torna vantajoso o uso da automação, logo discordo totalmente. ”

5.3.2.5 QUANTO À QUESTÃO 4

Execuções manuais dos testes funcionais acrescentam possibilidades de ocorrência de erros humanos.

Figura 17: Resultado da questão 4. Fonte autor.

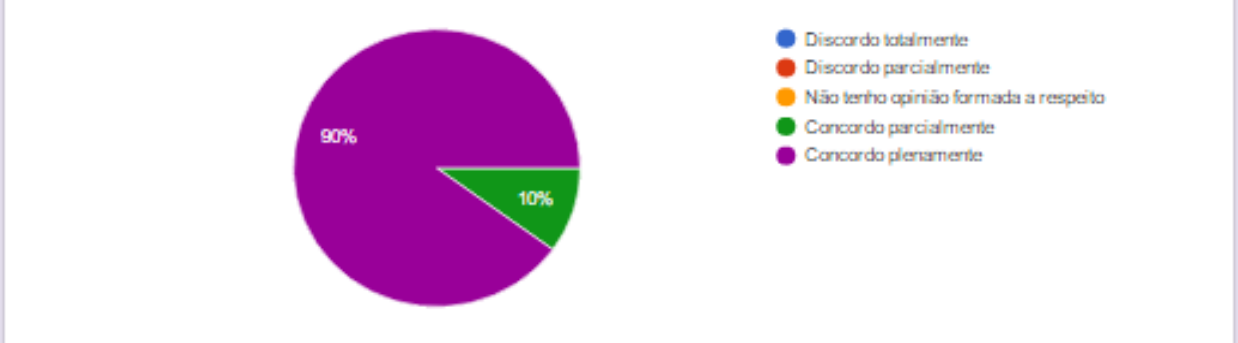


Tabela 5: Análise tabular da questão 4. Fonte autor.

	Quantidade	%
Concordo plenamente	9	90
Concordo parcialmente	1	10

A maioria concordou plenamente com a afirmação (9 questionados ou 90%), apenas 1 marcou como “Concordo parcialmente”. Ele destacou que os testes automatizados também estão sujeitos a erros humanos.

Justificativa: “Tanto testes manuais com testes automatizados incidem a intervenção humana com isso todos estão sujeitos a erros humanos. ” (Analista de Teste 3 anos e 6 meses, Testador 2 anos e 6 meses)

5.3.2.6 QUANTO À QUESTÃO 5

Quando uma mesma funcionalidade deve ser repetida diversas vezes pelo testador, a chance de erro humano aumenta ainda mais.

Figura 18: Resultado da questão 5. Fonte autor.

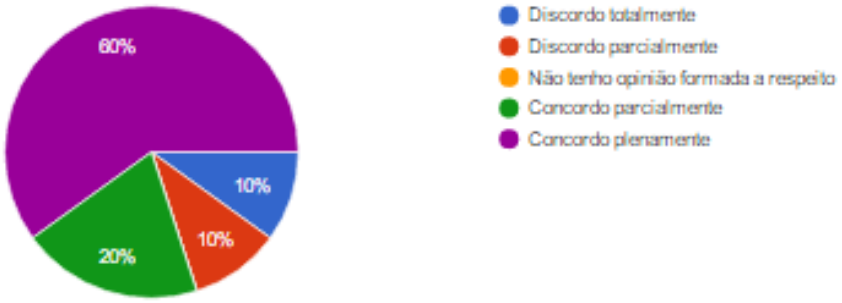


Tabela 6: Análise tabular da questão 5. Fonte autor.

	Quantidade	%
Concordo plenamente	6	60
Concordo parcialmente	2	20
Discordo Parcialmente	1	10
Discordo Totalmente	1	10

Dois concordaram parcialmente:

- O primeiro deu uma justificativa que tem conteúdo para marcar como "Concordo plenamente".

  Justificativa: "Quando se executa diversas vezes o teste pelo mesmo testador a tendência é que o mesmo fique com uma visão enviesada. " (Testador: 9 meses, Analista de Requisitos: 11 meses)

- O segundo também deu uma justificativa que tem conteúdo para marcar como "Concordo plenamente".

  Justificativa: "Quanto mais repetições o testador acaba deixando erros passarem, por achar que como ele já testou aquilo o sistema não tem mais erro. " (Testador: 1 ano)

Um discordou parcialmente justificando que com a análise dos resultados pode-se encontrar falhas nos processos de teste e assim a chance de erro diminui.

Justificativa: "Discordo após realização de teste deve ocorrer analise dos resultados com isso a retroalimentação nesse sentido a busca por falhas nos processos podem ser encontradas e assim a chance de erro pode ser reduzida. " (Analista de Teste 3 anos e 6 meses, Testador 2 anos e 6 meses)

Por fim um discordou totalmente justificando que com o tempo vem o aprimoramento do conhecimento e do que se espera como resultado esperado.

Justificativa: “Isso pra mim é uma falácia. Acho que com o tempo pode vim o aprimoramento do conhecimento e do que se espera como resultado esperado. ” (Analista de testes, 3 anos e 7 meses)

5.3.2.7 QUANTO Á QUESTÃO 6

Você prefere automatizar os testes funcionais via código ou via gravação? Além de opinar por apenas uma das opções, cite as vantagens da opção que você escolheu.

Tabela 7: Análise tabular da questão 6. Fonte autor.

	Quantidade	%
Via código	7	70
Via gravação	3	30

3 preferiram por automatizar pela gravação destacando a facilidade, a praticidade, versatilidade e a não necessidade de configuração prévia complexa. Dentre eles um destacou que tanto a automação via código quanto via gravação são importantes no projeto de software e que o uso de uma não exime o uso da outra.

7 escolheram via código e destacaram:

- O menor esforço na manutenabilidade dos testes em casos de mudança (menor retrabalho);
- A adequação a diversas situações;
- A identificação mais precisa dos componentes da interface;
- Maior número de maneiras para manipular como seus testes serão realizados;
- Oferece ferramentas mais poderosas, com mais recursos;
- Maior poder de reutilização das automações feitas via código.

5.4 AVALIAÇÃO DO RESULTADO OBTIDO NA QUESTÃO 6

Tomando como base as qualidades de cada estratégia de automação nas justificativas apresentadas pelos profissionais de testes, levantamos os seguintes pontos importantes que devem ser levados em conta para encorajar a automação dos testes funcionais guiados pela interface. Abaixo os pontos levantados:

- Esforço para configuração do teste;
- Reuso dos testes;
- Confiabilidade dos resultados dos testes;
- Custo do teste;
- Facilidade;
- Praticidade;

Em seguida, como um esclarecimento adicional, solicitamos aos respondentes a eles para numerar esses itens de 1 a 6 em ordem de importância, conforme a técnica de escala ordinal do *survey*, sendo o 1 o mais importante e o 6 o menos importante. Com base na frequência que cada item aparece nas posições de 1 a 6, obtivemos o seguinte resultado:

Tabela 8: Análise tabular dos pontos importantes para a escolha de codificar testes funcionais. Autor fonte.

	Frequência		Frequência
Primeiro lugar		Quarto lugar	
Reuso do teste	3	Facilidade	2
Custo do teste	2	Praticidade	2
Confiabilidade dos resultados do teste	1	Esforço para configuração	1
		Confiabilidade dos resultados do teste	1
Segundo lugar		Quinto lugar	
Custo do teste	3	Esforço para configuração	2
Reuso do teste	1	Praticidade	2
Praticidade	1	Confiabilidade dos resultados do teste	1
Confiabilidade dos resultados do teste	1	Facilidade	1
Terceiro lugar		Sexto lugar	
Esforço para configuração	2	Facilidade	3
Confiabilidade dos resultados do teste	2	Reuso do teste	1
Reuso do teste	1	Esforço para configuração	1
Custo do teste	1	Praticidade	1

Com base na tabela acima chegamos a seguinte ordem de importância:

1. Reuso do teste;
2. Custo do teste;
3. Esforço para configuração e Confiabilidade dos resultados do teste empatados;
4. Facilidade e Praticidade empatados;
5. Esforço para configuração e Praticidade empatados;
6. Facilidade.

5.5 ENTREVISTA

Como visto no item anteriormente, as afirmações 3 e 5 mostraram concordâncias e discordâncias por parte dos profissionais de testes e devido a isso foi feita uma entrevista para esclarecer os pontos conflitantes apresentados. Essa entrevista foi respondida por um gerente de teste, um analista de teste e um testador.

A afirmação 3 possui o seguinte conteúdo:

Os testes funcionais de interface com o usuário feitos manualmente exigem uma quantidade de tempo e mão de obra maior do que quando eles são feitos de forma automatizada.

Um dos profissionais de testes discordou e justificou que dependendo do teste, pode ser mais prático/rápido realizar a mão do que automatizá-lo. Com base nessa afirmação foi feita a seguinte pergunta em uma entrevista posterior:

Quais as características de um teste funcional de interface que fazem você considerá-lo não apto para automação?

O gerente de testes destacou que caso o projeto desenvolvido possua requisitos muito instáveis, o custo para a automação e manutenção dos testes funcionais de interface com o usuário ficam muito altos, o que inviabiliza a automação. Além disso é necessário avaliar o sistema como um todo e o framework utilizado para gerar os HTML's do projeto.

O analista de testes disse que caso o processo de desenvolvimento não organize e prepare o código desenvolvido para automação, a aplicação envolvida pode se tornar não apta para automação. Além disso outro item que pode inviabilizar a automação é uma interface gráfica que passe por alterações frequentemente.

O testador afirmou que a complexidade da estrutura do HTML das interfaces pode tornar as funcionalidades de uma aplicação não aptas para automação.

Também na afirmação 3, um profissional de testes discordando dela justificando que os testes manuais têm menor custo e tempo em relação aos testes automatizados. Porém essa relação tende a inverter a medida que é necessário reexecutar os testes. Sendo assim foi realizado a seguinte pergunta aos profissionais de testes:

Em média quantos ciclos de execução de teste têm sido feitos por ordem de serviço (OS) dos projetos da organização?

a. *Nunca são repetidos.*
b. *Menos que 3 repetições.*

c. *Mais que 3 repetições.*

O gerente de testes, o analista de testes e o testador afirmaram que em geral são feitos mais de 3 ciclos de execução de teste por OS. O analista de teste destacou que é aconselhado pela própria equipe de testes um mínimo de 3 ciclos de execução de teste por OS.

Em seguira foi perguntado aos profissionais de teste:

Qual foi o maior número de ciclos que você já presenciou na organização?

Resultado:

- Gerente de teste: 7 ciclos;
- Analista de teste: 6 ciclos;
- Testador: 7 ciclos.

Ainda na afirmação 3, um dos profissionais de teste discordou dela e justificando que nem para toda aplicação se torna vantajosa a automação, baseando-se nisso foi feita a pergunta:

Na sua opinião, quais a características de uma aplicação que justificariam a adoção de técnicas de automação dos testes funcionais de interface com o usuário?

O gerente de testes afirmou que é importante que os requisitos sejam estáveis e que o HTML esteja bem estruturado, esse item dependerá do framework utilizado para a criação das interfaces gráficas.

O analista de testes destacou que a automação dos testes funcionais de interface com o usuário evita que erros não sejam identificados a cada entrega de software funcionando, principalmente em projetos muitos extensos que desenvolvem aplicações de forma incremental. Além disso, ele justifica que esse tipo de automação deve ser praticado caso uma aplicação possua uma funcionalidade que seja crítica para o negócio.

O testador definiu que a automatizar os testes funcionais de interface com o usuário se justifica para uma aplicação que possua um código suscetível a erros, ou que as mudanças feitas ao longo do projeto causem erros.

A afirmação 5 é descrita da seguinte forma:

Quando uma mesma funcionalidade deve ser repetida diversas vezes pelo testador a chance de erro humano aumenta ainda mais.

Um dos profissionais que responderam o questionário discordaram da afirmação dizendo que após a realização dos testes devem ser analisados os resultados. Com isso, a busca por falhas nos processos podem ser encontradas e assim a chance de erro pode ser reduzida. Com base nessa

justificativa fizemos a seguinte pergunta aos entrevistados:

Na sua opinião, quais as análises devem ser feitas nos resultados esperados dos testes funcionais de interface com o usuário para evitar erros do testador durante a execução?

O gerente de teste afirmou que o testador deve ser mais investigativo quanto a documentação do projeto que está sendo testado. Dessa forma pode-se encontrar erros que não estão descritos nos casos de teste, e também é possível encontrar erros nos próprios casos de teste elaborados pelos analistas de teste.

O analista de testes destacou que é importante que os casos de teste sejam o principal guia do testador. Porém a documentação relacionada também deve ser verificada, principalmente quando houver divergência entre o que é apresentado pelo sistema e resultado esperado no caso de teste.

O testador informou que é necessário comparar o resultado esperado no caso de teste com o objetivo dos fluxos do caso de uso que está sendo executado.

Ainda na afirmação 5, um dos respondentes discordou justificando que com o tempo vem o aprimoramento do conhecimento e do que se espera como resultado. Com base nessa justificativa foi perguntado aos entrevistados:

Na sua opinião, a repetição da execução dos testes funcionais de interface com o usuário aumenta a possibilidade de erros humanos ou aumenta o domínio do testador sobre o sistema? Por quê?

Para o gerente de testes, as duas situações podem ocorrer. Os erros humanos ocorrem devido o vício dos testadores executarem diversas vezes a mesma funcionalidade diversas vezes, o que leva a diminuir sua atenção a cada execução, resultando em erros de execução.

O analista também considerou que ambas as situações podem ocorrer. Ele justificou dizendo que a repetição da execução de uma mesma tarefa pelo mesmo profissional pode aumentar seu domínio sobre a funcionalidade, porém sua atenção pode diminuir, permitindo erros de execução que deixem falhas no sistema passarem despercebidas.

O testador considerou que aumenta apenas a possibilidade de erros humanos, e justificou dizendo que o testador ao ficar vendo a mesma tela diversas vezes pode acabar deixando passar algum erro.

5.6 CONSIDERAÇÕES FINAIS

Podemos acreditar que os dados obtidos têm um bom nível de confiabilidade para o ambiente compreendido pelo fato das questões exigirem um bom nível de conhecimento dos entrevistados, ou seja, não é possível respondê-las sem ter um bom conhecimento sobre o assunto que advém da experiência dos respondentes em atuar na equipe de testes da organização. Além disso, de maneira geral, as justificativas obtidas são plausíveis e possuem embasamento teórico e prático dos profissionais envolvidos na pesquisa.

Com os resultados obtidos na primeira e segunda afirmação do questionário observou-se uma concordância dos especialistas em testes da organização estudada com relação à literatura pesquisada sobre a criticidade e importância dos testes funcionais de interface com o usuário para avaliar a qualidade do produto entregue ao usuário/cliente pois, eles, além de avaliar se o comportamento das funcionalidades está ocorrendo corretamente (Verificação), assim como se as funcionalidades entregues estão aderentes às necessidades declaradas dos usuários (Validação).

A terceira afirmação envolve questões de custo e mão de obra para a execução dos testes funcionais de interface com o usuário. As justificativas dos respondentes que concordaram com a afirmação nos levam a considerar que os testes funcionais feitos manualmente têm a tendência de serem mais onerosos que os automatizados. Ainda baseado nas justificativas, podemos dizer que uma vez que os testes estão automatizados eles podem ser reutilizados e executados diversas vezes, além disso outro respondente afirmou que a velocidade de execução do teste automatizado é bastante superior à execução humana.

Entretanto a terceira afirmação apresentou envolveu discordâncias por parte dos respondentes e como base nas suas justificativas foram levantadas dúvidas em relação as funcionalidades que não estão aptas para automação, as aplicações que apresentam características para automação e a quantidade de ciclos de teste que são executados na organização estudada. Com base nas respostas obtidas nas entrevistas realizadas, podemos afirmar que para se automatizar as funcionalidades de uma aplicação deve-se ser feita uma análise da estrutura e complexidade do HTML da interface gráfica, além de verificar a estabilidade dos requisitos do projeto e também interface gráfica da aplicação. Outro ponto importante que se deve analisar para iniciar a prática de automação é o número de ciclos que serão executados os testes envolvidos.

A quarta e quinta afirmação abordam falhas humanas que podem acontecer na execução dos testes funcionais, sejam eles realizados várias vezes seguidas ou não. As justificativas obtidas

nos levam a crer que tudo que é feito por seres humanos está suscetível a erros, porém em se tratando de erros de execução de testes funcionais de interface, eles podem ser minimizados por meio da prática da automação.

Dentre as afirmações citadas no parágrafo anterior, a quinta afirmação apresentou discordâncias. As justificativas obtidas foram usadas para levantar questões nas entrevistas em relação as análises que devem ser feitas durante a execução para se evitar falhas dos testadores e, em relação a repetição das execuções foi interrogado se ela aumenta o domínio do testador sobre o sistema ou aumenta a sua chance de cometer erros de execução. As respostas obtidas nos levam a crer que as análises feitas durante a execução pelo testador não devem se basear apenas nos resultados dos casos de teste, é necessário também ter conhecimento da documentação disponível para o projeto e usá-la como apoio no momento da execução. Já quanto ao tema de repetição das execuções, podemos afirmar que ela pode aumentar a chance de erros humanos pelo fato de que a execução de uma mesma funcionalidade diversas vezes pode ir reduzindo a atenção do testador. Porém a afirmação anterior não elimina uma possibilidade de aumento no domínio do testador quanto a aplicação que está sendo testada.

Reunindo as justificativas dos entrevistados, há indícios de que no ambiente da organização estudada, a automação dos testes funcionais pode ser vantajosa devido às possibilidades de reuso dos testes automatizados e, consequentemente, ganhos de velocidade nas suas execuções. Esses ganhos de velocidade poderiam melhorar os prazos, os custos e diminuir as ocorrências de falhas humanas nas próprias execuções dos testes previstos. Porém, para que estes resultados sejam realmente obtidos é necessário um estudo da aplicação de software para avaliar algumas condições próprias como: a estrutura e complexidade dos HTML's da aplicação assim como do framework utilizado para criá-los, a estabilidade dos requisitos e da interface gráfica do sistema, a duração que terá o projeto de software e se ele é incremental ou não, e também a existência de funcionalidades críticas para o negócio.

Sendo assim, a automatização dos testes funcionais de interface com o usuário pode ser justificada caso o projeto possua um HTML bem estruturado e sem uma alta complexidade, além disso o framework utilizado para gerar os HTML's devem favorecer essa tarefa. Os requisitos e a interface gráfica quanto mais estáveis mais impulsionam a tarefa de automatizar esse tipo de teste.

6 ESTUDO DE CASO

6.1 OBJETIVOS

O principal objetivo deste estudo de caso é verificar a veracidade dos dados levantados no survey e nas entrevistas realizadas no capítulo anterior. Primeiramente verificaremos se a automação dos testes de usuário guiados pela interface é vantajosa frente a execução manual desses testes. Em seguida, analisaremos qual seria a melhor estratégia para automatizar estes testes de acordo com as estratégias discutidas no capítulo 4. E por último, discutiremos a confiabilidade e o reuso dessas automações.

6.2 HIPÓTESES

Projeto da organização estudada, utilizando as estratégias de gravação e codificação:

1- O tempo de execução dos testes automatizados são menores que os testes manuais;
2- Ambas estratégias de automação utilizadas atingem um índice de reuso satisfatório que justifique seu uso nos testes de software;
3- Ambas estratégias de automação utilizadas apresentam confiabilidade nos resultados dos seus testes;

Projeto CodeSchool, utilizando as estratégias de codificação e linguagem natural:

1- O tempo de execução dos testes automatizados são menores que os testes manuais;
2- Ambas estratégias de automação atingem um índice de reuso satisfatório que justifique seu uso nos testes de software;
3- Ambas estratégias de automação utilizadas apresentam confiabilidade nos resultados dos seus testes;

6.3 INDICADORES

Conforme a discussão realizada na seção 5.4 do capítulo 5, foram levantados os itens mais importantes nas justificativas dos profissionais de teste para encorajar a automação dos testes funcionais guiados pela interface e em seguida os mesmos profissionais enumeraram estes fatores por ordem de importância. Nesse resultado obtido, foram selecionados os 3 primeiros considerados mais importantes para o levantamento de indicadores deste estudo de caso: Reuso, Custo e

Confiabilidade.

Os indicadores serão refletidos em métricas que foram levantadas utilizando o GQM (Goal-Question-Metric). O GQM é um método que foca no objetivo que se deseja atingir para definir as métricas que serão utilizadas para levantar dados do problema ao qual você deseja resolver (Solingen, 1999).

Para mensurar a confiabilidade da execução dos testes automatizadas foram definidas as seguintes métricas:

Verificações

Goal(Objetivo)	Identificar a quantidade de verificações realizadas pelos testes automatizados
Question(Pergunta)	Quantas verificações os testes automatizados fazendo durante a execução do suite de testes?
Metric(Métrica)	Número de verificações

Testes falhos

Goal(Objetivo)	Identificar a quantidade de testes que serão falhados de forma inconsistentes caso algum teste automatizado falhe no meio da execução
Question(Pergunta)	Quantas testes automatizados serão falhados caso um deles falhe durante a execução do suite de testes?
Metric(Métrica)	Número de testes falhos

Em relação aos custos de execução e implementação dos testes:

Tempo de execução

Goal(Objetivo)	Identificar o tempo de execução dos testes manuais e automatizados
Question(Pergunta)	Quanto de tempo foi gasto para execução dos testes?
Metric(Métrica)	Quantidade de tempo em minutos e segundos

Tempo de implementação

Goal(Objetivo)	Identificar o tempo gasto na implementação dos testes automatizados de acordo com as estratégias utilizadas
Question(Pergunta)	Quantas de tempo foi gasto na automatização dos testes em cada estratégia?

Metric(Métrica)	Quantidade de tempo em horas

Em relação ao reuso dos testes automatizados:

Ciclos

Goal(Objetivo)	Identificar a quantidade de ciclos que os testes podem ser reutilizados
Question(Pergunta)	Quantos ciclos poderiam fazer uso dos testes automatizados?
Metric(Métrica)	Número de ciclos

Casos de uso

Goal(Objetivo)	Identificar a possibilidade do reuso dos testes automatizados em casos de uso de outras OS's?
Question(Pergunta)	Quantos casos de uso de outras OS's se beneficiariam da execução dos testes automatizados?
Metric(Métrica)	Número de casos de uso

Ajustes

Goal(Objetivo)	Identificar a necessidade de manutenção dos testes automatizados no momento de reuso para a geração de dados diferentes de acordo com cada estratégia de automação
Question(Pergunta)	Quantos ajustes são necessários nos testes automatizados para que sempre seja gerado dados novos?
Metric(Métrica)	Número de ajustes

6.4 PROCEDIMENTOS E COLETA DE DADOS

6.4.1 APLICAÇÃO DA ESTRATÉGIA DE CODIFICAÇÃO NOS PROJETOS

Os procedimentos e a coleta de dados foram feitos em 2 projetos, um projeto pertence a organização estudada a qual foram extraídas as opiniões dos profissionais de teste e outro projeto é o Codeschool desenvolvido pelo LAPPIS da FGA.

O CodeSchool foi desenvolvido com o objetivo de auxiliar no aprendizado de diversas linguagens de programação como Python, C e C++. Nele o usuário primeiramente faz o seu cadastro e após realizar o login ele em acesso a exercícios de programação que poderão estar sendo

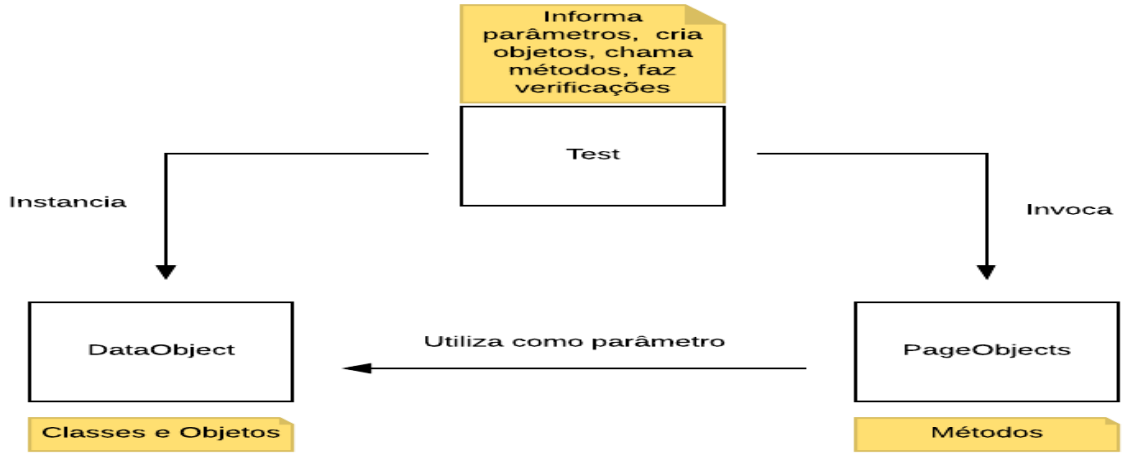
resolvidos por ele na linguagem que ele desejar. Após implementar o código de cada exercício o usuário envia, o sistema analisa o código e retorna para o usuário os erros encontrados ou uma mensagem de sucesso informando que o código enviado está correto.

Primeiramente vamos descrever os procedimentos realizados para a estratégia de codificação em ambos projetos citados. A arquitetura utilizada para codificação possui os seguintes pacotes:

- DataObjects: Este pacote contém as classes que serão utilizadas para instanciar os objetos de cada caso de uso, de forma que ela traz como atributos os campos que serão preenchidos na funcionalidade;
- PageObjects: Possui as classes que utilizam os objetos dos DataObjects como parâmetro para implementar os métodos que serão utilizados para a execução dos fluxos dos casos de uso;
- Test: Contém as classes de teste que instanciam as classes contidas nos DataObjects, chamam os métodos implementados no PageObjects e realiza as verificações necessárias.

A figura abaixo representa a arquitetura utilizada para a codificação dos testes automatizados guiados pela interface:

Figura 19: Arquitetura para uso da estratégia de codificação. Fonte autor.



Os passos para automatizar um fluxo do caso de uso começam primeiramente criando classe DataObject, em seguida são implementados os métodos nas classes PageObject e por último criando a classe de teste no pacote Test.

Para o projeto da organização estudada foi selecionada uma Ordem de Serviço que contém

8 casos de uso para serem testados. Desses casos de uso foram automatizados os fluxos de cadastro e de pesquisa, de forma que eles simulem o cadastro de um item da funcionalidade e em seguida é feita a pesquisa para verificar se item foi salvo corretamente pelo sistema.

Abaixo estão apresentados os casos de uso os quais foram codificados os fluxos de cadastro e pesquisa e também a ordem de execução dos mesmos:

1.	Login
2.	Manter tipo de contrato
3.	Manter tipificação da receita
4.	Manter código da receita
5.	Manter previsão de receita com contrato
6.	Manter previsão de receita sem contrato
7.	Validar previsão de receita
8.	Associar registro de arrecadação com contrato
9.	Associar registro de arrecadação sem contrato

Para a execução dos testes implementados foi utilizado o xml do framework do TestNG, de forma que nele estão descritos o browser que se deseja utilizar para execução e a ordem que a suíte de testes deve ser executada. Para este estudo de caso foi utilizado o Firefox como browser e a ordem foi a mesma apresentada acima. Abaixo a imagem do testng.xml:

Figura 20: Xml TestNG para execução do suíte de testes das automações do projeto da organização. Fonte autor.

```
    <parameter name="url_execucao"
        value="https://10.0.0.160:8443/sso/login?service=http%3A%2F%2F10.0.0.160%3A9090%2Freceita%2Fview%2Fhome.xhtml" />
        <!-- value="https://10.0.0.160:8443/sso/login" /> -->
    <!-- Configurar para a maquina que for fazer os testes -->
    <parameter name="firefox_driver_path"
        value="C:\\Users\\joao.araruna\\AppData\\Local\\Mozilla Firefox\\firefox.exe" />
    <parameter name="firefox_geko_driver_path"
        value="E:\\Bibliotecas\\geckodriver-v0.11.1-win64\\geckodriver.exe" />
    <parameter name="ie_driver_path"
        value="E:\\Bibliotecas\\IEDriverServer_x64_2.46.0\\IEDriverServer.exe" />
    <parameter name="chrome_driver_path"
        value="E:\\Bibliotecas\\chromedriver_win32\\chromedriver.exe" />

    <parameter name="timeout_default" value="15" />

    <parameter name="iterador_id_entidades" value="06" />

    <test name="Test">

        <parameter name="browser_execucao" value="firefox" /> <!--Opcoes disponiveis: firefox, firefoxGeko, ie, chrome -->

        <groups>
            <dependencies>
                <group name="PrevisaoReceitaComContrato" depends-on="TipoContrato"/>
                <group name="PrevisaoReceitaComContrato" depends-on="TipificacaoReceita"/>
                <group name="PrevisaoReceitaComContrato" depends-on="CodigoReceita"/>
                <group name="PrevisaoReceitaSemmContrato" depends-on="TipoContrato"/>
                <group name="PrevisaoReceitaSemContrato" depends-on="TipificacaoReceita"/>
                <group name="PrevisaoReceitaSemContrato" depends-on="CodigoReceita"/>
                <group name="ValidarPrevisaoReceita" depends-on="PrevisaoReceitaComContrato"/>
                <group name="ValidarPrevisaoReceita" depends-on="PrevisaoReceitaSemContrato"/>
                <group name="AssociarRAComContratoTest" depends-on="ValidarPrevisaoReceita"/>
                <group name="AssociarRASemContratoTest" depends-on="ValidarPrevisaoReceita"/>
            </dependencies>
        </groups>

        <classes>
            <class name="br.net.mirante.test.ct.LoginTest" />
            <class name="br.net.mirante.test.TipoContratoTest" />
            <class name="br.net.mirante.test.TipificacaoReceitaTest" />
            <class name="br.net.mirante.test.CodigoReceitaTest" />
            <class name="br.net.mirante.test.PrevisaoReceitaComContratoTest" />
            <class name="br.net.mirante.test.PrevisaoReceitaSemContratoTest" />
            <class name="br.net.mirante.test.PrevisaoReceitaValidarTest" />
            <class name="br.net.mirante.test.AssociarRAComContratoTest" />
            <class name="br.net.mirante.test.AssociarRASemContratoTest" />
```

Como podemos ver na imagem acima o xml do TestNG utiliza a marcação “url_execução” para definir a url por onde se iniciará a execução, “browser_execução” para definir o browser a ser utilizado e em seguida são apresentados os caminhos das classes de teste dentro do projeto na ordem de execução desejada.

Após a execução dos testes foi colhido o tempo de execução apresentado pelo próprio TestNG e também o tempo gasto com a implementação dos testes automatizados via código.

Para o projeto do Codeschool foi utilizada a mesma arquitetura apresentada anteriormente. Foram automatizados os fluxos de cadastro, login e de envio de 7 exercícios disponíveis os quais os códigos foram feitos na linguagem C. A ordem de execução pode ser vista no xml do TesteNG apresentada abaixo:

Figura 21: Xml TestNG para execução do suíte de testes das automações do projeto CodeSchool. Fonte autor.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite">
    <!-- Configurar a execucao de teste -->

    <parameter name="url_execucao"
        value="https://codeschool.lappis.rocks/auth/login/" />
    <!-- Configurar para a maquina que for fazer os testes -->
    <parameter name="firefox_driver_path"
        value="\\home\\jgararuna\\Documents\\geckodriver\\geckodriver.exe" />

  <test name="Test">
    <parameter name="browser_execucao" value="firefox" />
    <classes>
      <class name="test.LoginTest"/>
      <class name="test.ContaBotecoTest"/>
      <class name="test.ReceitaPaoTest"/>
      <class name="test.CadastroExercicioTest"/>
      <class name="test.SomaSimplesTest"/>
      <class name="test.RecursividadeTest"/>
      <class name="test.ProgramaPoliticoTest"/>
       <class name="test.ConjecturaCollatzTest"/>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

Após a execução dos testes automatizados foram coletados o tempo de execução apresentado pelo TestNG e também o tempo gasto com a codificação dos testes automatizados via código.

O fluxo de cadastro de usuário não foi incluído na execução devido sua menor importância frente aos outros, mas sua automação poderá ser utilizada caso sejam feitas alterações no sistema que tenha impacto na funcionalidade de cadastro e seja necessária sua verificação novamente. No caso do xml acima bastaria incluir o caminho da classe de teste de cadastro de usuário antes da

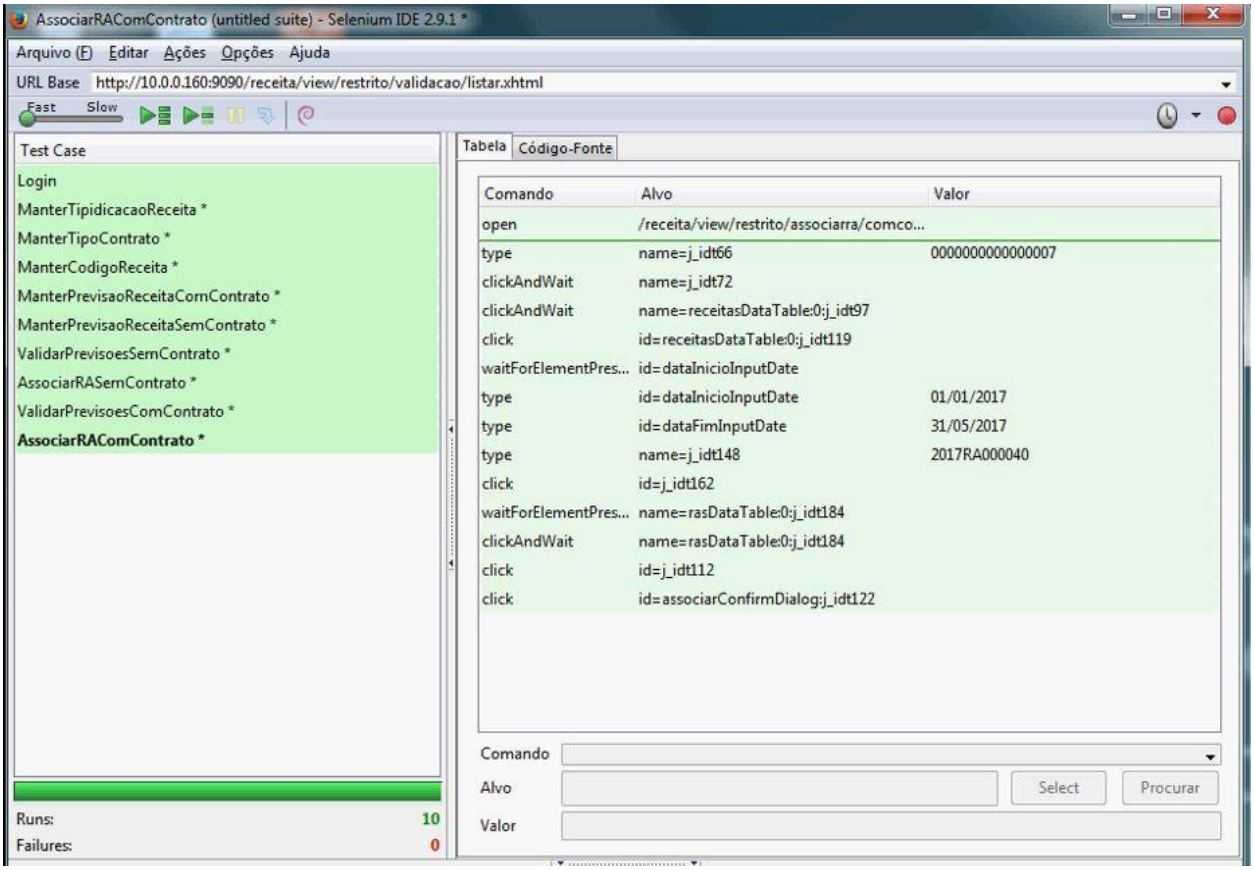
classe de teste de login e executar o teste normalmente.

6.4.2 APLICAÇÃO DA ESTRATÉGIA DE GRAVAÇÃO

A estratégia de gravação foi aplicada somente no projeto da organização estudada, pois ela é a outra estratégia de automação utilizada pela equipe de testes para a automação de testes funcionais guiados pela interface. Foi utilizada o uso da ferramenta Selenium IDE, que foi apresentada no capítulo 4 sobre estratégias de automação.

Os fluxos automatizados são os mesmos que foram feitos via codificação para a ordem de serviço da organização estudada. Após a automatização dos testes via gravação obtermos o suíte de testes que pode ser apresentado na própria ferramenta na parte e Test Case conforme a figura abaixo:

Figura 22: Suíte de testes no Selenium IDE das automações via gravação. Fonte autor.



Aqui vale destacar que a ordem de execução dos testes é mesma utilizada na execução dos testes automatizados via codificação. Após automatização via gravação o suíte de testes e executado, e ao final foi coletado o tempo de execução e também o tempo gasto para automatizá-los.

6.4.3 APLICAÇÃO DA ESTRATÉGIA DE LINGUAGEM NATURAL

A estratégia de de automação via linguagem natural foi aplicada apenas no projeto do CodeSchool com o objetivo de compará-la com a estratégia de codificação que também foi aplicada no projeto CodeSchool.

Os fluxos automatizados aqui são os mesmos que foram automatizados via codificação para o projeto CodeSchool. Para que implementar a automação via linguagem natural, primeiramente são criadas as features com o uso da linguagem Gherkins e em seguida o Cucumber é executado para que sejam gerados os métodos a serem implementados. Após a implementação dos métodos eles são executados utilizando a classe TestRunner, nela estará contido suíte de testes automatizados via linguagem natural como podemos ver na figura abaixo:

Figura 23: Suíte de testes na classe TestRunner das automações via linguagem natural. Fonte autor.

```
1  package runner;
2
3  import org.junit.runner.RunWith;
7
8  @RunWith(Cucumber.class)
9  @Cucumber.Options(features="features"
10                   ,glue={"stepDefinition"}
11                   ,tags={"@Login, @ContaBoteco, @ReceitaPao, @Cadastro, @SomaSimples, @Recursividade, @ProgramaPolitico, @ConjecturaCollatz"}
12                   //,tags ={"@Login, @ContaBoteco"}
13                   )
14 public class TestRunner {
15
16         public static WebDriver driver;
17
18 }
```

Na figura acima na parte opções do Cucumber, o termo “features” define o nome da pasta que contém as features com os cenários descritos em linguagem natural na linguagem Gherkins. O termo “glue” recebe o nome do pacote que contém as classes de testes com os métodos implementados. E por fim o termo “tags” recebe as marcações que estão contidas nas features e representam os testes automatizados que serão executados na ordem apresentada, ou seja, nele está contido o suíte de testes.

Para a execução do suíte de testes automatizados em linguagem natural, foi utilizado o framework JUnit e após a execução foi coletado o tempo de execução apresentado pelo JUnit bem como o tempo gasto com a automatização dessa estratégia.

6.4.4 EXECUÇAO MANUAL DOS TESTES

Além de automatizar os testes nas diversas estratégias de automação estudadas, também foram executados os testes funcionais relacionados aos projetos da organização e do Codeschool de forma manual. Ao final da execução manual foi coletado o tempo de execução manual, ou seja, o tempo que um testador normal levaria para executar os mesmos fluxos que foram automatizados na mesma ordem em que os suítes de testes foram executados.

6.5 RESULTADOS

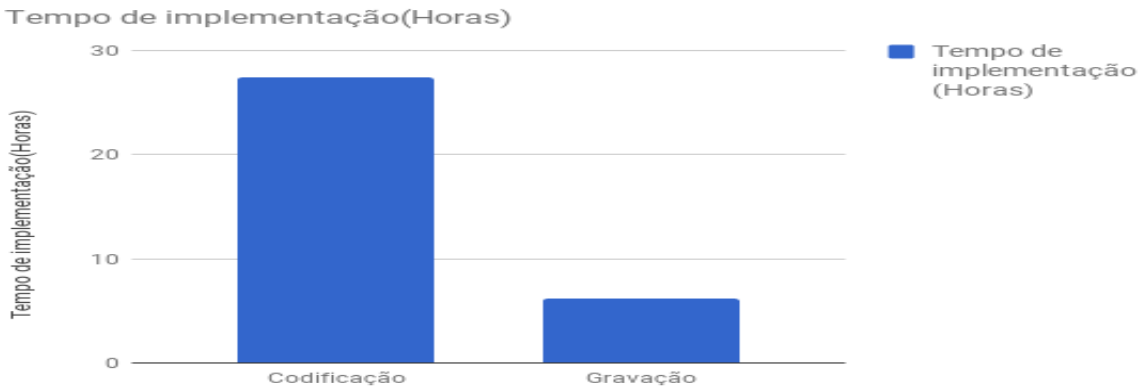
Com base nas métricas levantadas na seção 6.4 deste capítulo faremos a análise dos dados obtidos durante os procedimentos e também dos dados que não foram obtidos durante o procedimento citados no capítulo anterior, mas que complementam o estudo sobre reuso e confiabilidade dos testes.

6.5.1 CUSTO

Primeiramente analisaremos as métricas relacionadas a custo, e dentre elas temos os tempos de execução dos testes e o tempo gasto com as automações de acordo com cada estratégia.

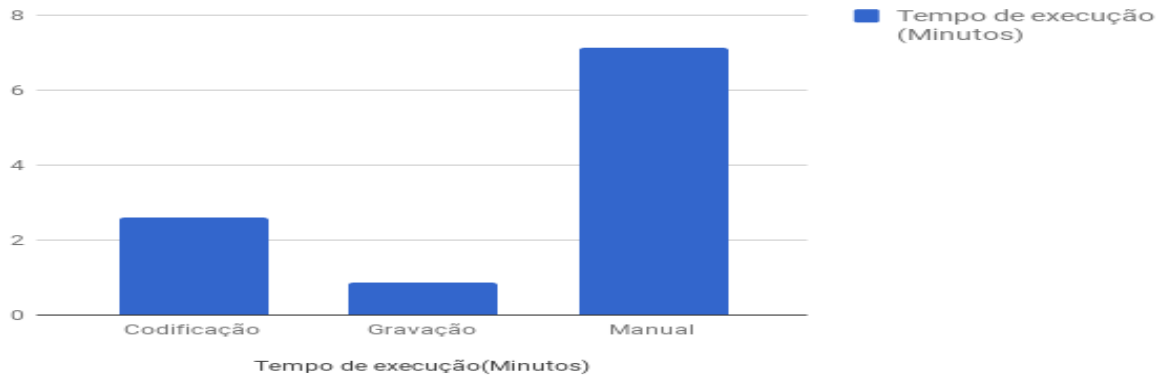
Para o projeto a organização o resultado em relação ao tempo gasto pode ser observado no gráfico abaixo:

Figura 24: Gráfico comparativo do tempo de implementação dos testes de acordo com cada estratégia. Fonte autor.



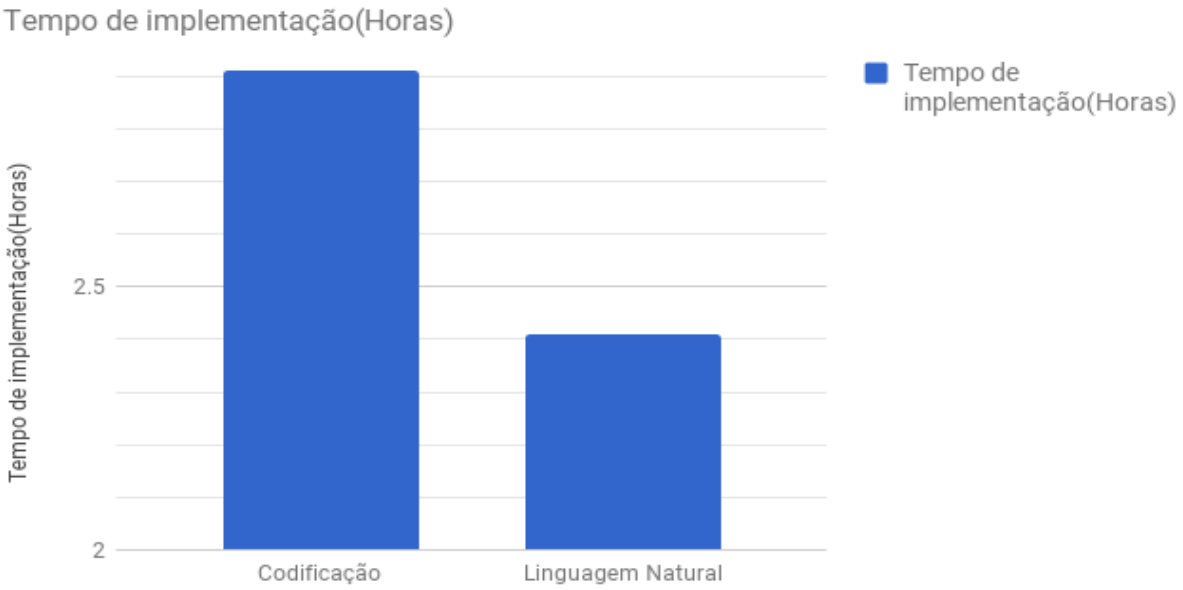
Em relação ao tempo execução dos testes automatizados de acordo com cada estratégia utilizada e também da execução manual chegamos ao resultado do gráfico abaixo:

Figura 25: Gráfico comparativo do tempo de execução dos testes de acordo com cada estratégia. Fonte autor.



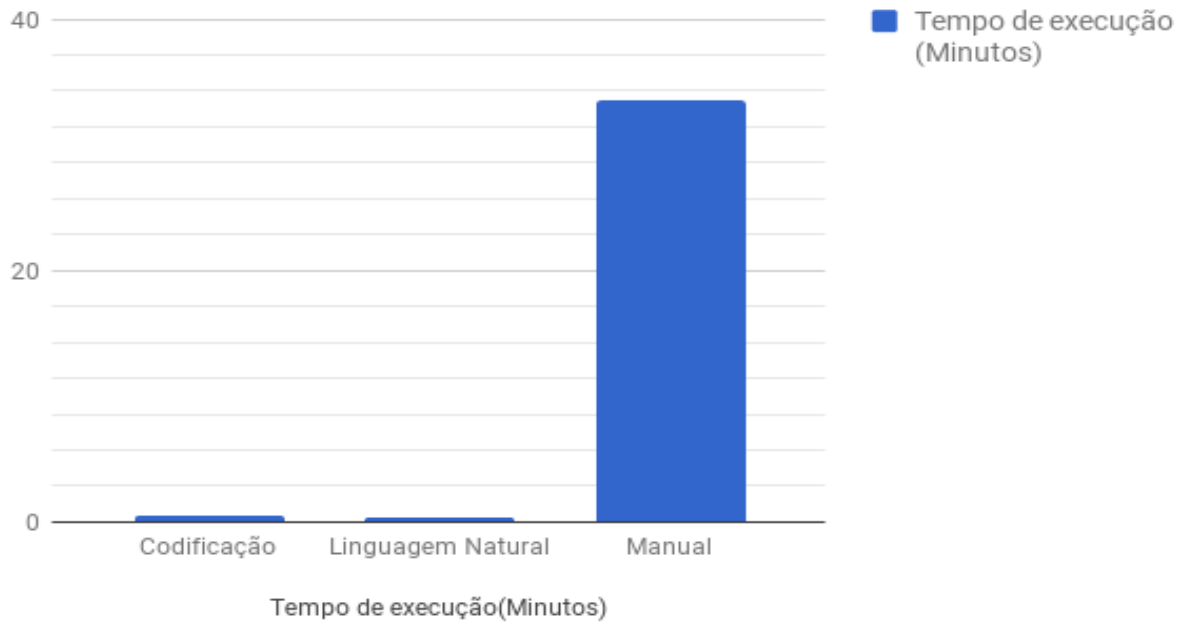
Em relação ao projeto do Codeschool chegamos ao seguinte resultado em relação ao tempo de implementação das automatizações conforme as estratégias de codificação e linguagem natural:

Figura 26: Gráfico comparativo do tempo de implementação dos testes de acordo com cada estratégia. Fonte autor.



O tempo de execução dos testes automatizados e executados de forma manual estão apresentados no gráfico abaixo:

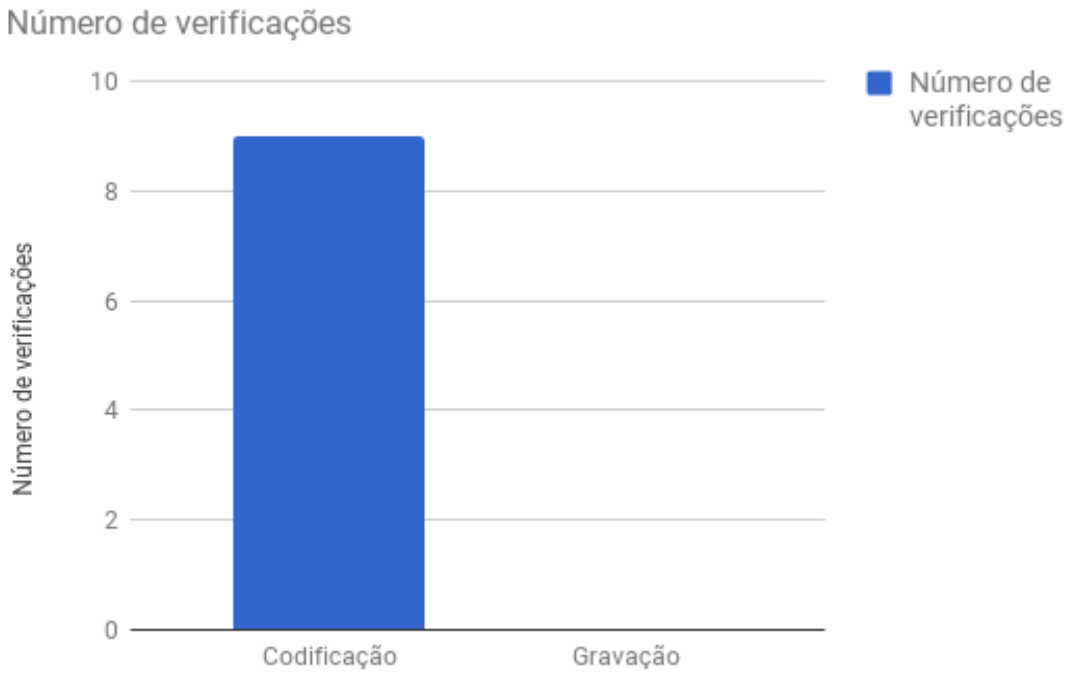
Figura 27: Gráfico comparativo do tempo de execução dos testes de acordo com cada estratégia. Fonte autor.



6.5.2 CONFIABILIDADE

Mudando para os indicadores relacionados à confiabilidade dos testes, temos métricas relacionadas a quantidade de verificações que são feitas durante a execução dos testes automatizados de acordo com cada estratégia. O gráfico abaixo representa o número de verificações que são feitas durante a execução dos testes automatizados para cada estratégia utilizada no projeto da organização estudada:

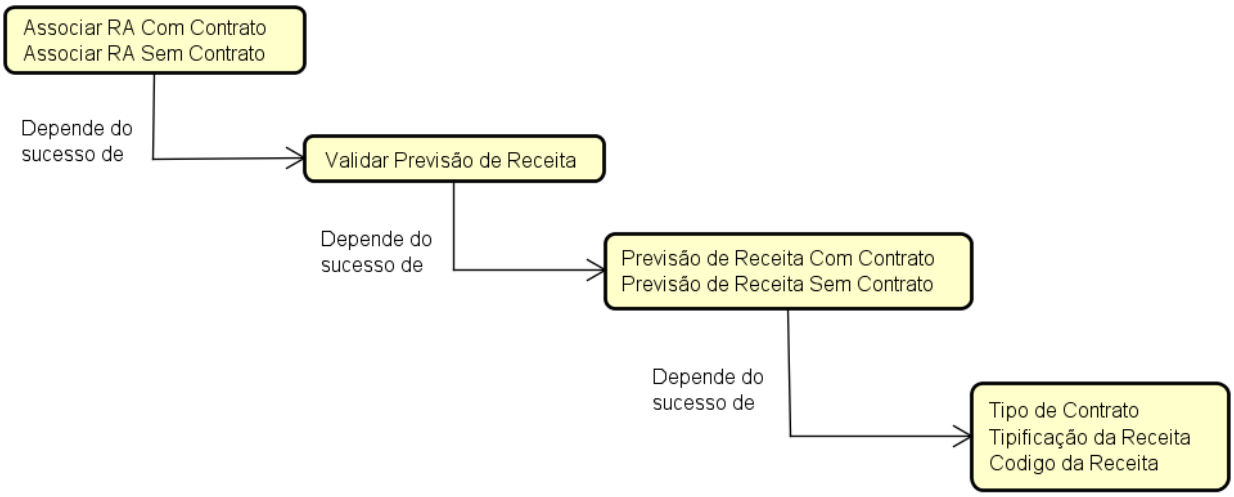
Figura 28: Gráfico comparativo de verificações feitas pelos testes automatizados de acordo com cada estratégia. Fonte autor.



Como podemos ver, a estratégia de codificação com o suporte do TestNG faz as verificações necessárias para garantir que os testes realmente cadastraram ou validaram os dados informados pelas funcionalidades testadas. Já a estratégia de gravação automatiza as ações do usuário, porém não realiza nenhuma verificação após a execução de cada teste para garantir o funcionamento correto da aplicação. Sendo assim os testes automatizados via gravação não verificam se realmente foram cadastrados ou validados os dados informados pelas funcionalidades testadas, sendo necessária uma verificação após a execução dos mesmos quanto a isso.

Outro fator importante para a questão de confiabilidade dos testes são as condições de parada de cada estratégia de automação. No caso do projeto da organização estudada isso faz uma grande diferença devido a dependência que existe entre as funcionalidades, de forma que uma não possa ser executada caso a funcionalidade anterior não tenha sido finalizada com sucesso, ou seja, sem erros. A dependência entre as funcionalidades automatizadas pode ser melhor representada na figura abaixo:

Figura 29: Imagem que representa a dependência entre as funcionalidades do projeto da organização. Fonte autor.



Devido a essa dependência entre as funcionalidades, é necessário que as funcionalidades dependentes de outra não sejam executadas caso a funcionalidade que ela dependa tenha falhado durante a execução do suíte de testes.

Na estratégia de codificação, o TestNG permite que sejam incluídos no seu xml marcações que expressam a dependência entre entre os testes que serão executados. Na Figura 14 da seção anterior é possível observar a seguinte expressão na parte de dependências "<group name="PrevisaoReceitaComContrato" depends-on="TipoContrato"/>", que significa dizer que para que o teste relacionado a funcionalidade de Previsão de receita com contrato seja executado, é preciso que o teste referente Tipo de Contrato tenha passado, o que pode ser confirmado na Figura 22 acima. Caso contrário, o suíte de testes encerra sua execução.

Porém para a estratégia de gravação não é possível estabelecer uma condição de parada do suíte de teste, ou seja, caso um teste falhe durante a execução o suíte de testes continuará a executar os próximos testes. Tomando como base a Figura 22, caso o teste da funcionalidade de Tipo de Contrato falhe a execução do suíte seguirá adiante e acabará por falhar os demais testes que dependem direta ou indiretamente do sucesso dos testes da funcionalidade de Tipo de Contrato. Essa falha dos testes executados após a falha do teste relacionado ao Tipo de Contrato é inconsistente, pois uma de suas dependências não está sendo disponibilizada, sendo assim, não podemos afirmar que os mesmos são falhos por si só, o que afeta a confiabilidade dos resultados obtidos com a estratégia de gravação. A tabela abaixo apresenta a quantidade de testes que são definidos como falhos de forma inconsistente caso determinado teste falhe durante a execução do suíte:

Tabela 9: Análise tabular dos testes falhados de forma inconsistente pela automação via gravação. Fonte autor.

	Número de testes definidos como falhos de forma inconsistente	
Caso os testes abaixo falhem	Gravação	Codificação
Código da Receita	5	0
Tipo de Contrato	5	0
Tipificação da Receita	5	0
Previsão da Receita Com Contrato	3	0
Previsão da Receita Sem Contrato	3	0
Validação	2	0

Quanto ao projeto do Codeschool a confiabilidade dos testes é mesurada por meio da verificação do resultado apresentado ao usuário após o mesmo enviar o código do exercício. Tanto a estratégia de codificação quanto a de linguagem natural realizam estas verificações fazendo uso de frameworks de testes, no caso da codificação foi feito uso do TestNG e para linguagem natural foi utilizado o JUnit. Dessa forma, em vias de comparação, ambas as estratégias realizam as verificações necessárias para garantir o funcionamento correto das funcionalidades que foram automatizadas.

6.5.3 REUSO

Primeiramente analisando o projeto da organização estudada, levantamos dados quanto a possiblidade de execuções dos testes automatizados em relação a quantidades de ciclos de testes que as automações poderiam ser reaproveitadas e o tempo que seria economizado em relação a execução manual dos testes. No caso da ordem de serviço dos fluxos que foram automatizados neste estudo de caso, foram executados 4 ciclos de testes.

Os gráficos a seguir demonstram a quantidade de tempo gasto na execução manual e automatizadas dos testes (de acordo com cada estratégia) e também a economia de tempo que cada estratégia proporcionaria ao fazer reuso dos testes automatizados em cada ciclo.

Figura 30: Gráfico que representa a economia de tempo com o reuso dos testes automatizados via codificação. Fonte autor.

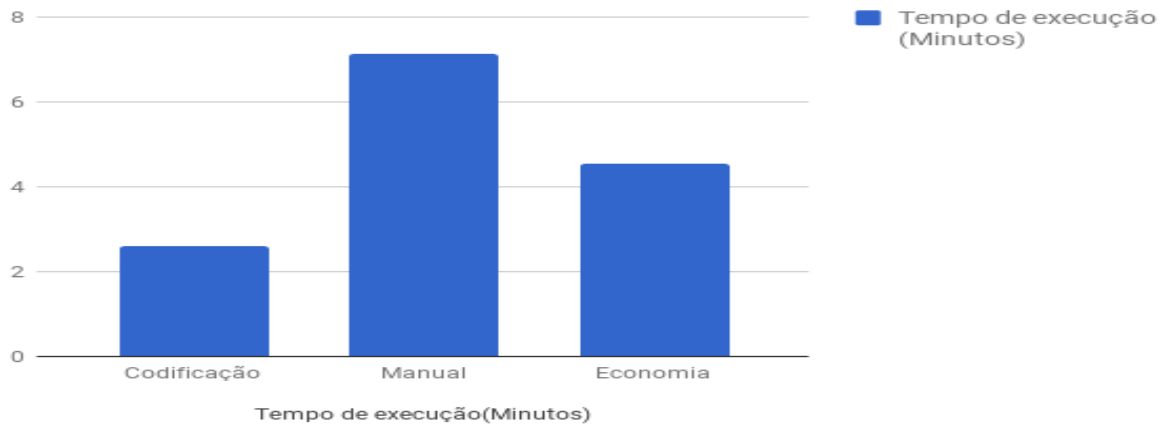
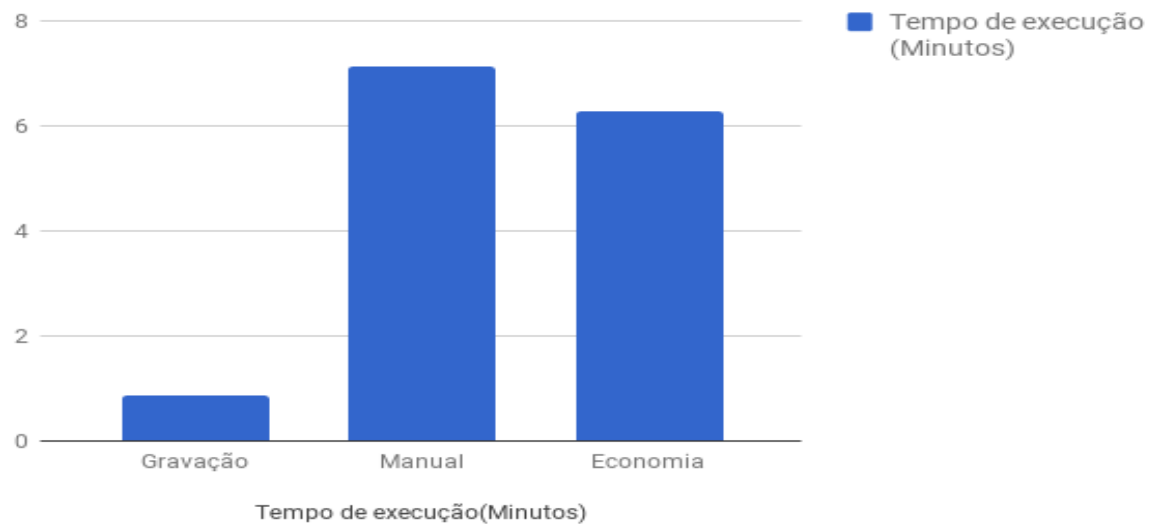


Figura 31: Gráfico que representa a economia de tempo com o reuso dos testes automatizados via gravação. Fonte autor.



Aqui vale ressaltar que os fluxos que foram automatizados são repetidos diversas vezes dentro dos ciclos de testes executados, de forma que essas repetições são de acordo com a necessidade do testador e não podem ser mensuradas. Dessa forma, a economia de tempo a ser mensurada dentro de cada ciclo não se dá apenas pela economia de tempo que esta demonstrada nos gráficos acima e sim pela economia de tempo multiplicada pela quantidade de vezes que os fluxos foram executados dentro de cada ciclo.

Outra importante colaboração que os testes automatizados proporcionam e a geração de massa para relatórios, devido a pratica da organização em limpar o banco de dados da aplicação entre os ciclos de testes.

Uma outra ordem de serviço da organização estudada está relacionada a geração de relatórios que dependem diretamente das massas geradas pela execução dos fluxos que foram automatizados neste estudo de caso. Em se tratando de números, para a ordem de serviço citada foram executados 3 ciclos de testes e envolveu a geração de 6 relatórios que de uma forma geral dependem das seguintes pré-condições citadas abaixo:

1 Deve haver Previsão de Receita com Contrato Cadastradas;

2 Deve haver Previsão de Receita sem Contrato Cadastradas;

3 Deve haver Previsão de Receita com Contrato Cadastradas associadas a um Registro de Arrecadação (RA);

4 Deve haver Previsão de Receita sem Contrato Cadastradas associadas a um Registro de

Arrecadação (RA);

As pré-condições acima podem ser atingidas de forma mais rápida executando os testes automatizados neste estudo de caso em ambas estratégias utilizadas no projeto da organização (gravação e codificação). Dessa forma os testadores podem analisar esses relatórios sem a necessidade de perder grande quantidade de tempo criando dados para esses relatórios.

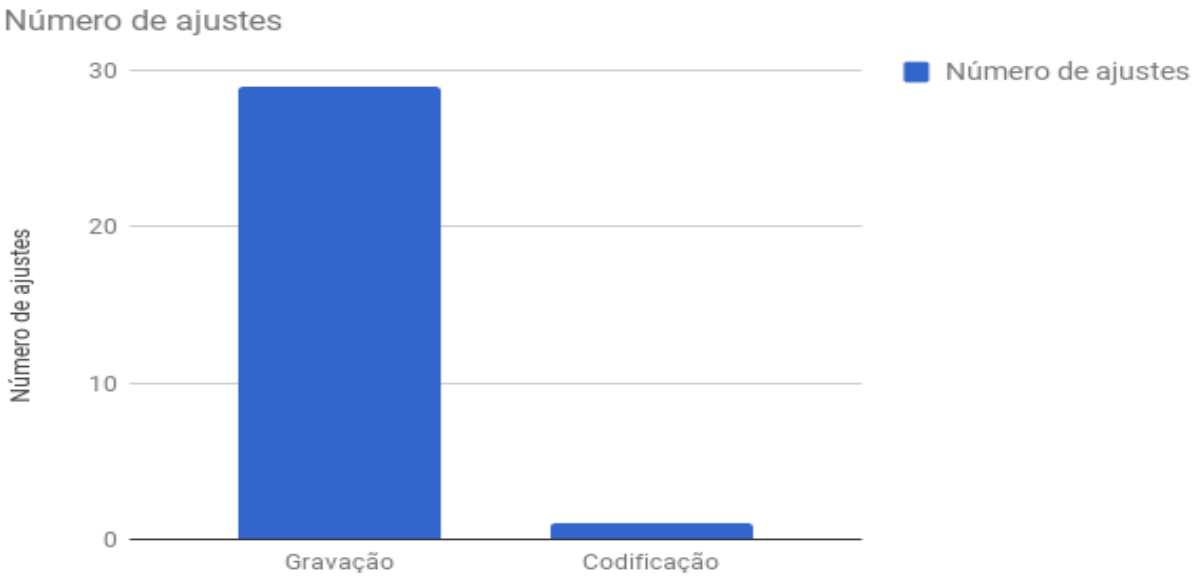
Ainda tomando como base a execução relacionada a geração dos relatórios, e preciso que sejam geradas massas com dados diferentes para que os mesmos atinjam uma quantidade de dados satisfatória para eles tenham o corpo de um relatório e para que testes como ordenação e paginação do relatório possam ser feitos. Dessa forma, para que o reuso dos testes automatizados sejam utilizados diversas vezes sempre criando dados novos são necessários ajustes nas automações.

Os testes automatizados via codificação como já dito anteriormente, fazem uso do xml do TestNG para serem executados. Dentro desse xml é possível utilizar um índice que será replicado em todos os testes automatizados de forma que cada vez que esse índice recebe um valor e o suíte de testes é executado, novas massas são geradas.

Na figura 14, que apresenta o xml para execução do suíte de testes do projeto da organização, podemos ver o parâmetro “iterator_id_entidades”, nele é colocado o valor do índice e quando o suíte de testes é executado esse índice será replicado em todos os campos envolvidos nos cadastros de massa. Por exemplo, vamos supor que o “iterator_id_entidades” esteja com valor 7, e eu esteja cadastrando um Tipo de Contrato. Ao criar o objeto Tipo de Contrato, eu preencho os seus atributos sempre chamando o índice do xml de forma que ele fique único, sendo assim sua descrição seria definida como “TipoContrato7”. E assim para gerar novos dados bastaria alterar apenas o índice do xml para outro número que ainda não tenha sido utilizado.

Para a estratégia de gravação, como não é feito o uso de um xml para a execução dos testes não é possível fazer uso do índice. Dessa forma, para que sejam gerados novos dados para compor os relatórios, é necessário que sejam feitos ajustes de forma manual em cada teste automatizado, o que leva a um esforço maior para fazer o reuso desses testes automatizados caso o objetivo seja gerar dados novos. O gráfico abaixo mostra a quantidade de ajustes para cada estratégia que são necessários para fazer o reuso dos testes automatizados gerando sempre dados novos.

Figura 32: Gráfico que representa a quantidade de ajustes por estratégia para gerar dados novos. Fonte autor.



Quanto ao projeto do Codeschool podemos considerar que o reuso dos testes automatizados seria feito executando os testes cada vez que uma nova versão do software fosse disponibilizada para testes, e isso representaria o ciclo de teste. Sendo assim chegamos aos seguintes gráficos quanto a economia de tempo que o reuso dos testes automatizados proporcionariam a cada ciclo de testes de acordo com cada estratégia:

Figura 33: Gráfico que representa a economia de tempo ao fazer reuso dos testes automatizados via codificação. Fonte autor.

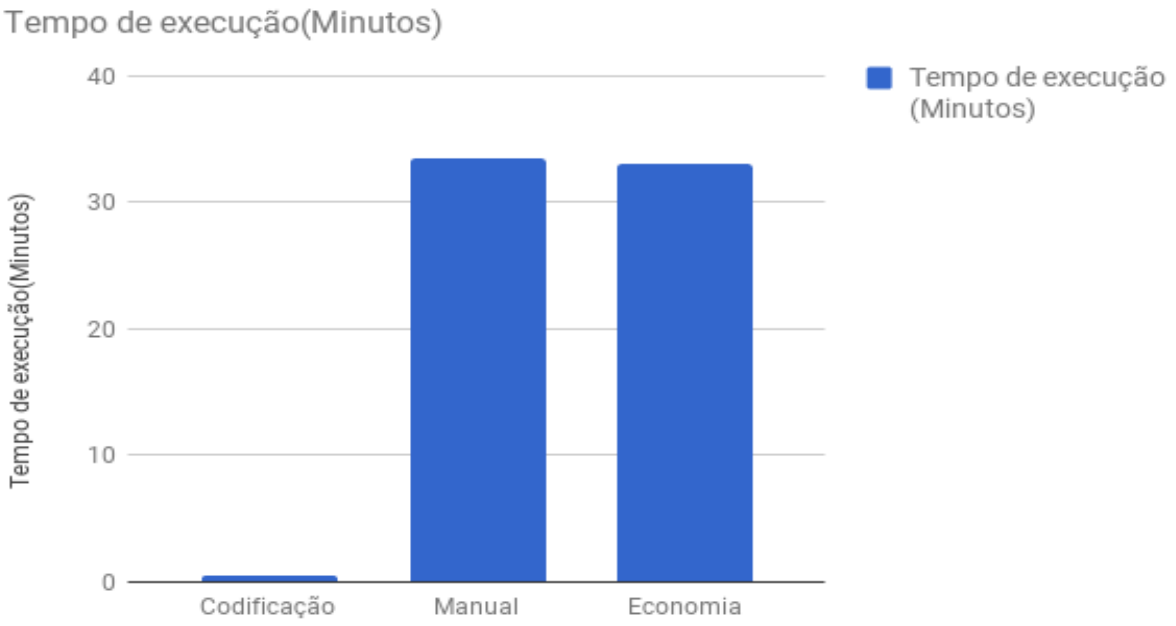
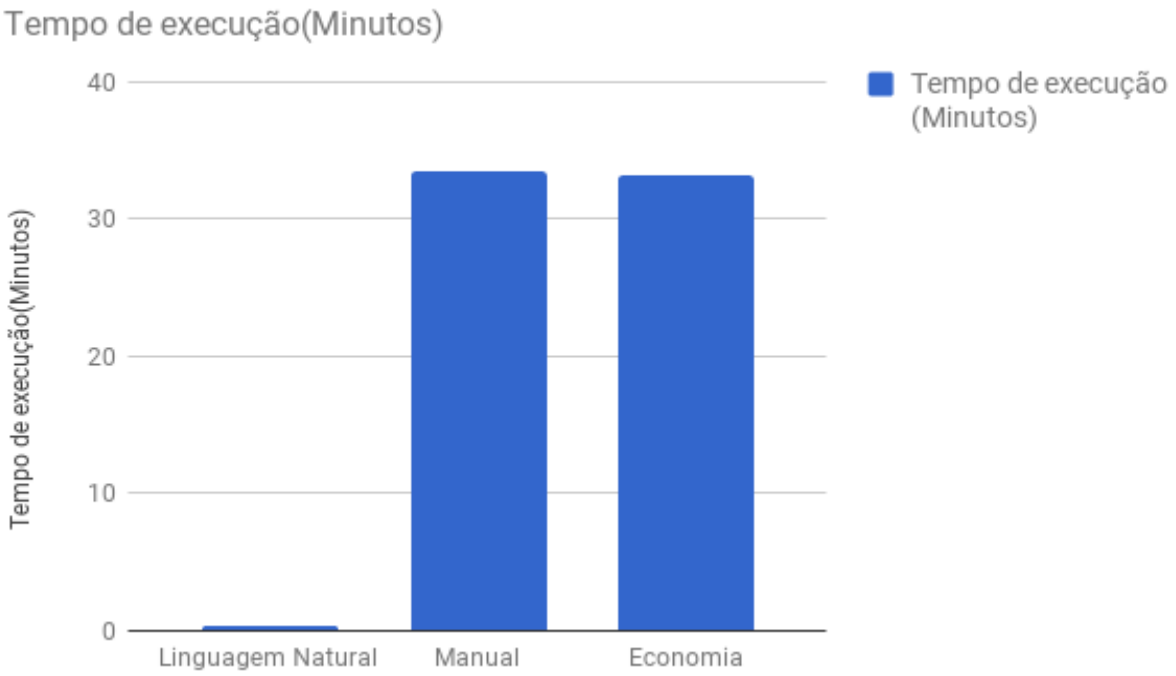


Figura 34: Gráfico que representa a economia de tempo ao fazer reuso dos testes automatizados via linguagem natural. Fonte autor.



6.6 ANÁLISE DOS RESULTADOS

Analisando primeiramente o projeto da organização observamos que quanto ao custo, ambas a estratégias apresentadas se mostraram vantajosas frente a execução manual quanto ao tempo de execução do suíte de testes que contém as funcionalidades que foram automatizadas, tendo a estratégia de gravação gastado apenas 12% do tempo da execução manual e a codificação 36%. Porém, nesse quesito se formos comparar apenas as estratégias observamos que a estratégia de gravação leva vantagem sobre a codificação pois ela leva aproximadamente 3 vezes menos tempo para executar o suíte de testes automatizados.

Quanto ao tempo gasto com a automação em si, a estratégia de gravação também leva vantagem sobre a codificação tendo levado aproximadamente 4,4 menos tempo para automatizar as funcionalidades.

No quesito confiabilidade a estratégia de codificação leva vantagem sobre a de gravação por fazer o uso de frameworks de testes (TestNG) para verificar se as funcionalidades cumpriram seus objetivos esperado. Além disso, devido a dependência entre as funcionalidades conforme já

destacado na seção anterior, os frameworks de teste utilizados na codificação fazem com que o suíte de testes pare caso falhe algum teste que influencie diretamente no funcionamento do próximo teste. Como na gravação caso algum teste falhe a execução do suíte continua, ele pode chegar a falhar até entre 2 e 5 testes de forma inconsistente.

No caso do reuso também podemos afirmar com base nos resultados que ambas as estratégias de automação são vantajosas, de forma que a cada ciclo de testes a estratégia de codificação permite uma economia de aproximadamente 67% em relação a execução manual e a estratégia de gravação aproximadamente 88%.

Ainda no reuso, ao pensar nesse quesito como utilizar as automações para a geração de dados para os relatórios envolvidos em outra ordem de serviço da organização, analisamos a quantidade de ajustes que devem ser feitos nos testes automatizados para atingir tal objetivo. Com base nos resultados podemos afirmar que a estratégia de codificação leva grande vantagem nesse quesito por exigir uma quantidade muito menor de ajustes ou manutenção para a geração de dados novos em relação a estratégia de gravação.

Agora mudando o foco para o projeto do CodeSchool, analisamos primeiramente a questão do custo e temos que ambas as estratégias (Codificação e Linguagem Natural) utilizadas apresentaram ser vantajosas frente a execução manual, de forma que o suíte de testes com as funcionalidades automatizadas na estratégia de codificação usa apenas 1.3% da execução manual para ser executado, e a estratégia de linguagem natural aproximadamente 1%. Ao compararmos o tempo de execução apenas das estratégias observamos que a estratégia de linguagem natural leva vantagem sobre a codificação, de forma que o suíte de testes da linguagem natural gasta 27% menos tempo que a codificação para ser executado.

Quanto ao tempo gasto na automação dos testes em si, a estratégia de linguagem natural é mais vantajosa devido ao fato de gastar menos 18% do tempo de implementação utilizado na codificação.

Na questão de confiabilidade houve empate pois ambas as estratégias fazem uso de frameworks de testes (TestNG e JUnit) que verificam se as funcionalidades automatizadas estão funcionando conforme o esperado.

No reuso, como é levado em conta o tempo que é economizado a cada execução do suíte

de testes, temos que a cada ciclo de testes realizado após cada nova versão disponibilizada a estratégia de linguagem natural leva ligeira vantagem por permitir uma economia de tempo aproximadamente 0,4% maior que a estratégia de codificação frente a execução manual.

7 CONCLUSÃO

O foco do estudo deste trabalho de conclusão de curso é a equipe de testes de uma organização de mercado privado que possui projetos tanto privados quanto públicos. A equipe de testes envolvidas é formada por 1 gerente de testes, 5 analistas de testes e 4 testadores. O processo de teste segue um processo definido pela empresa e aplicável aos seus projetos de software.

O problema abordado nesse TCC está presente no capítulo 1 na seção 1.2, ele foi definido como identificar as vantagens de automatizar testes funcionais de interface com o usuário na visão da equipe de testes estudada, além de identificar qual a melhor estratégia de automação para realizar essa tarefa. Com base no problema escolhido foi definido um objetivo geral na seção 1.3 do capítulo, ele se resume em avaliar as percepções dos testadores sobre as estratégias de automatização dos testes funcionais de interface com o usuário.

Ainda na seção 1.3 do capítulo 1, são definidos os objetivos específicos desse trabalho de conclusão de curso. O primeiro objetivo específico foi atingido no terceiro capítulo, o qual destaca-se a importância dos testes no desenvolvimento de software. Além disso, é enfatizada a importância dos testes funcionais e são abordados pontos que levam a escolha por automatizá-los.

Em seguida, o segundo objetivo específico foi cumprido no capítulo quatro que descreve a automação de testes em si e destaca suas vantagens para um projeto de software. Além disso, com base nas ferramentas utilizadas pela equipe de testes estudada, são explicadas as estratégias de automatizar testes funcionais de interface com o usuário, destacando suas vantagens e desvantagens e também como instalar, configurar e utilizar essas ferramentas.

O quinto capítulo cumpre os objetivos específicos três, quatro e cinco iniciando pela definição das técnicas usadas para levantar os dados da equipe de testes estudada (*survey* com questionário e entrevista estruturada), e após aplicá-las foi apresentado os resultados obtidos e as avaliações feitas a partir deles. A partir do *survey* foi levanta opiniões e justificativas sobre afirmações relacionadas aos testes automatizados; também foi definida a preferência da automatização dos testes funcionais e a justificativa para tal, a partir dessas justificativas foram retirados itens que justifiquem a escolha de determinada estratégia de automatização dos testes funcionais de interface com o usuário. Em seguida, esses pontos foram numerados por ordem de importância pelos profissionais de teste baseando-se na estratégia que foi escolhida pela maioria. Após a realização do questionário, foi elaborada uma entrevista estruturada para investigar as justificativas encontradas nas discordâncias (parciais e completas) obtidas em algumas questões

do questionário.

Os resultados obtidos pela aplicação dos questionários e entrevistas adicionais, constituem indícios que motivam o uso de estratégias de automação dos testes funcionais de interface com o usuário como a melhoria dos prazos, custos e a diminuição de problemas relacionados a erros humanos ao executar os próprios testes.

Neste sentido o estudo de caso realizado avaliou a veracidade das opiniões dos técnicos envolvidos sobre os aspectos importantes para a automatização dos testes funcionais de interface com o usuário. O estudo de caso envolveu um projeto da própria organização estudada e também o projeto CodeSchool desenvolvido pelo LAPPIS, e nele foi observado qual seria melhor estratégia baseando-se em métricas levantadas a partir do custo, confiabilidade de reuso.

Quanto ao projeto da organização estudada temos que quanto ao custo, tanto de execução quanto de implementação, temos como melhor opção a estratégia de gravação por exigir uma menor quantidade de tempo em relação a codificação. No reuso temos também como base o tempo gasto na execução do suíte de testes nos ciclos posteriores o que mais uma vez nos leva a escolher a estratégia de gravação. Entretanto, caso a organização deseje a geração de dados novos para testar relatórios a melhor escolha seria o uso da codificação pois ela exigiria uma quantidade de ajustes muito inferior à da estratégia de gravação.

A questão de confiabilidade aponta diretamente aponta diretamente para a estratégia de codificação por esta ter suporte de frameworks de testes que realizam as verificações necessárias e permite a inclusão de condições de parada para que testes não sejam falhados de forma errada, recursos que não são compreendidos na estratégia de gravação.

No projeto Codeschool nos quesitos de reuso e custo temos que a estratégia automação seria melhor escolha, apesar da diferença dos dados obtidos quando comparados com a codificação ter sido tão expressiva. Quanto as questões de confiabilidade ambas as estratégias podem ser selecionadas por utilizarem os frameworks de teste para as verificações necessárias.

Por fim, podemos destacar que as automações, independente da estratégia utilizada, em ambos os projetos se mostraram vantajosas em comparação as execuções manuais quanto ao quesito de tempo de execução de forma que as economias de tempo obtidas comprovam que as automações dos testes funcionais guiados pela interface são vantajosos e trazem ganhos significativos para a área de testes de software.

8 REFERÊNCIAS

Ahmed, Bestoun S., Taib Sh. Abdulsamad, e Moayad Y. Potrus. (2015). "Achievement of Minimized Combinatorial Test Suite for Configuration-Aware Software Functional Testing Using the Cuckoo Search Algorithm". *Information and Software Technology* 66 (2015): 13–29.

Ammann, Paul, e Jeff Offutt. *Introduction to software testing*. Cambridge University Press, 2008. *Google Scholar*.

Burns, David (2012). Selenium 2 Testing Tools Beginner's Guide. Packt Publishing.

Dillman, D. A. (1978). *Mail and telephone surveys. The total design method.* New York: Wiley.

Dingsøyr, Torgeir et al. "A Decade of Agile Methodologies: Towards Explaining Agile Software Development". *Journal of Systems and Software* 85.6 (2012): 1213–1221. *CrossRef*. Web.

Dyer, Michael (1993). Distribution-based statistical sampling: An approach to software functional test. The Journal of Systems & Software, 1993, Vol.20(2), pp.107-114.

Fink, A., & Kosecoff, J. (1985). *How to conduct surveys: A step-by-step guide.* Bervely Hills: Sage.

GALIN, D. (2003). "Software Quality Assurance, From Theory to Implementation", 1st edition. Pearson Addison Wesley.

Gil, Antonio Carlos. *Como elaborar projetos de pesquisa*. São Paulo: Atlas, 2008. Print.

GIL, Antonio Carlos. Entrevista. In: Métodos e Técnicas de Pesquisa Social. 6. ed. São Paulo: Atlas, 2008.

Gopal, Anandasivam, e Balaji R. Koka. "The role of contracts on quality and returns to quality in offshore software development outsourcing". *Decision Sciences* 41.3 (2010): 491–516.

Gundecha, Unmesh. *Instant Selenium Testing Tools Starter a Short, Fast, and Focused Guide to Selenium Testing Tools That Delivers Immediate Results*. Birmingham, U.K.: Packt Publishing, 2013. *Open WorldCat*.

Gundecha, Unmesh. *Selenium Testing Tools Cookbook over 90 Recipes to Build, Maintain, and Improve Test Automation with Selenium WebDriver*. Birminghan, UK: Packt Pub., 2012. *Open WorldCat*.

Günther, H. & Lopes, Jr., J. (1990). Perguntas abertas vs. perguntas fechadas: Uma comparação empírica. *Psicologia: Teoria e Pesquisa, 6*, 203-213.

Häser, Florian (2016). Is business domain language support beneficial for creating test case specifications: A controlled experiment. Information and software technology [0950-5849] vol:79 pág:52 -62.

Hogg, Mike (2013). European Medical Device Technology. UBM Canon LLC.

Howden, William E. "Functional program testing". *IEEE Transactions on Software Engineering* 2 (1980): 162–169.

Hushalini, S (2014); Software Test Automation in Practice: Empirical Study from Sri Lanka. COMPUSOFT : International Journal of Advanced Computer Technology' [2320-0790] vol:3 fasc:11 pág:1232 -1237.

IEEE Computer Society (2014). Guide to the Software Engineering Body of Knowledge (Swebok(r)): Version 3.0. IEEE Computer Society Press.

IEEE Std 610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology.

ISO/IEC-25000 (2005) "Software engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE", ISO/IEC.

Kovalenko, Dima. Selenium Design Patterns and Best Practices. Packt Publishing (2014).

Larkman, Deane Robert (2012). A decision support framework for software test managers. University of Canberra : Faculty of Information Sciences & Engineering.

Li, Kanglin, e Mengqi Wu. *Effective GUI Test Automation: Developing an Automated GUI Testing Tool*. San Francisco, Calif.; London: SYBEX, 2005. *Open WorldCat*.

Maldonado, José C.; Barbosa, Ellen F.; Vincenzi, Auri, M. R.; Delamaro, Márcio E.; Souza, Simone R. S; Jino, Mario; (2007). Introdução ao Teste de Software. CAMPUS - RJ, 2007.

Mao, Chengying (2014). Generating Test Data for Software Structural Testing Based on Particle Swarm Optimization. Arabian Journal for Science and Engineering [1319-8025] vol:39 fasc:6 pág:4593 -4607.

Meszaros, Gerard. *xUnit test patterns: refactoring test code*. Upper Saddle River, NJ: Addison-Wesley, 2007. Print. The Addison-Wesley signature series.

Misra, Sudip (2000). A software test plan generation approach for pedagogical puposes. National Library of Canada.

MORESI, E. Metodologia da Pesquisa. 2003.

Myers, Glenford J., Corey Sandler, e Tom Badgett. *The art of software testing*. 3rd ed. Hoboken, N.J: John Wiley & Sons, 2012. Print.

Naik, K.; Tripathy, P.; (2011). Software Testing and Quality Assurance: Theory and Practice. John Wiley & Sons.

PASQUALI, L. Instrumentos Psicológicos: manual prático de elaboração. Brasília: LABPAM /

IBAPP, 1999.

Perry, William E. *Effective methods for software testing*. 3rd ed. Indianapolis, IN: Wiley, 2006.

Prasad, Raghavendra MG. Learning Selenium Testing Tools, Third Edition. Packt Publishing (2015).

Pressman, R. S. (2000) Software Engineering – A Practitioner's Approach, European adaptation by D. Ince, 5th edn, McGraw-Hill International, London.

Rätamann, Manfred; De Young, Clinton (2003); Galileo Computing Software Testing and Internationalization. Lemoine International, Incorporated.

Carvalho, Rogério; Manhães, Rodrigo; Silva, Fernando(2010). Filling the Gap between Business Process Modeling and Behavior Driven Development. *CoRR, abs/1005.4975*, .

Carvalho, Rogério; Manhães, Rodrigo; Silva, Fernando (2010). Mapping Business Process Modeling constructs to Behavior Driven Development. *CoRR, abs/1006.4892*, .

ROSE, S.; WYNNE, M.; HELLESØY, A. **The cucumber for Java book: behaviour-driven development for testers and developers**. Dallas, Texas: The Pragmatic Bookshelf, 2015.

Sharma, Abhiraja (2012). Defect Prevention Technique in Test Case of Software Process for Quality Improvement. International journal of computer technology and applications, vol:3, iss:1, pg:56 - 56.

SOLINGEN, RINI VAN; BERGHOUT, EGON. **The Goal/Question/Metric Method:a practical guide for quality improvement of software development.** London: McGraw-Hill Publishing Company, 1999.

Srikanth, Hema, e Sean Banerjee. "Improving Test Efficiency through System Test Prioritization". *Journal of Systems and Software* 85.5 (2012): 1176–1187.

Stephens, Rod. *Beginning software engineering*. Indianapolis, IN: John Wiley and Sons, 2015.

Strugar, Ivan, Jovana Zoroja, e Božidar Jaković. "Development Practices of Embedded Systems: SMEs in SEE countries". *Business Systems Research Journal* 5.1 (2014): n. pag.

Sudman, S., & Bradburn, N. M. (1982). *Asking questions.* San Francisco, CA: Jossey-Bass.

Tavares, Hugo Lopes; Rezende, Gustavo Guimarães; Santos, Vanderson Mota; Manhães, Rodrigo Soares; Carvalho, Rogério Atem (2010). A tool stack for implementing Behaviour-Driven Development in Python. *CoRR, abs/1007.1722*, .

Zhan, Zhimin (2015).Selenium WebDriver Recipes in C#, Second Edition. Apress Media.

Link do projeto Codeschool no GitHub: https://github.com/cslms/cs-server.

APÊNDICE A – QUESTIONÁRIO

1 Introdução

Questionário sobre automação de testes de interface

O questionário a seguir tem como objetivo levantar dados para responder a seguinte questão:

Quais as vantagens para um projeto de desenvolvimento de software em automatizar os testes funcionais?

A pergunta acima será respondida de forma separada em trabalhos de conclusão do curso de Engenharia de Software da UnB – Universidade de Brasília/FGA – Faculdade Gama (TCC1 e 2). O TCC1 é de caráter exploratório e tem como objetivo identificar os aspectos importantes (segundo os envolvidos na área de teste de software) que justificam a automatização de testes de interface do usuário por meio de elaboração de código. Os resultados obtidos no TCC1 serão insumos importantes para o TCC2, onde é prevista a realização de experimentos para confirmar se é vantagem fazer a automação do referido teste por meio da elaboração de código.

Responda as perguntas que se seguem de acordo com as suas descrições. Não é necessária a identificação do respondente no questionário.

Questionário:

Dê sua opinião sobre as afirmações abaixo e justifique.

* Required

Qual o seu perfil na área de testes de software *

- [ ] Testador
- [ ] Analista de testes
- [ ] Gerente de projeto
- [ ] Outro(s)

Informe o tempo que você trabalhou no perfil selecionado. Caso você tenha trabalhado em mais de um perfil, diga o tempo que atuou em cada um deles. (Ex: Testador: 2 anos e 3 meses)

Your answer

2 Questão 1

1) No desenvolvimento de software os testes constituem uma atividade crítica. *

- ◯ Discordo totalmente
- ◯ Discordo parcialmente
- ◯ Não tenho opinião formada a respeito
- ◯ Concordo parcialmente
- ◯ Concordo plenamente

Justifique. *

Your answer

3 Questão 2

2) Os testes funcionais têm alta importância para avaliar se as funcionalidades do software estão funcionando corretamente. *

- ◯ Discordo totalmente
- ◯ Discordo parcialmente
- ◯ Não tenho opinião formada a respeito
- ◯ Concordo parcialmente
- ◯ Concordo plenamente

Justifique. *

Your answer

4 Questão 3

3) Os testes funcionais de interface do usuário feitos manualmente exigem uma quantidade de tempo e mão de obra maior do que quando eles são feitos de forma automatizada. *

- ◯ Discordo totalmente
- ◯ Discordo parcialmente
- ◯ Não tenho opinião formada a respeito
- ◯ Concordo parcialmente
- ◯ Concordo plenamente

Justifique. *

Your answer

5 Questão 4

4) Execuções manuais dos testes funcionais acrescentam possibilidades de ocorrência de erros humanos. *

- ◯ Discordo totalmente
- ◯ Discordo parcialmente
- ◯ Não tenho opinião formada a respeito
- ◯ Concordo parcialmente
- ◯ Concordo plenamente

Justifique. *

Your answer

6 Questão 5

5) Quando uma mesma funcionalidade deve ser repetida diversas vezes pelo testador, a chance de erro humano aumenta ainda mais. *

- ◯ Discordo totalmente
- ◯ Discordo parcialmente
- ◯ Não tenho opinião formada a respeito
- ◯ Concordo parcialmente
- ◯ Concordo plenamente

Justifique. *

Your answer

7 Questão 6

A automação por meio de gravação é realizada por uma ferramenta que grava as ações manuais do testador e depois realiza a reprodução delas. Já na automação via código os passos do usuário são codificados usando a API de uma ferramenta de automação.

Em relação às formas de automação descritas acima responda as questões 6 e 7.

6) Você prefere automatizar os testes funcionais via código ou via gravação? Além de opinar por apenas uma das opções, cite as vantagens da opção que você escolheu. *

Your answer