UNIVERSIDADE FEDERAL DE SANTA CATARINA

Automação de testes para plataforma Flex

Augusto Boehme Tepedino Martins

Florianópolis - SC

Novembro 2015

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE ESTATÍSTICA E INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

Augusto Boehme Tepedino Martins

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação

Florianópolis - SC

Novembro 2015

Augusto Boehme Tepedino Martins

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação

Banca Examinadora

_____

Profª. Drª. Patricia Vilain

Universidade Federal de Santa Catarina - UFSC

_____

Profº. Dr. Raul Sidnei Wazlawick
Universidade Federal de Santa Catarina - UFSC

Orientador:

_____

Jean Carlo Rossa Hauck, Dr.

Universidade Federal de Santa Catarina - UFSC

AGRADECIMENTOS

Primeiramente agradeço a meus pais e irmãos, por todo carinho, ensinamento, e apoio que me deram não apenas na faculdade, mas em toda minha vida.

Aos meus amigos mais próximos que, mesmo com os altos e baixos, estiveram ao meu lado para me ajudar quando eu precisava, na vida acadêmica e pessoal.

Aos meus amigos do exterior Kristen e Felix. Nunca os conheci pessoalmente, mas me fizeram sentir como se estivessem sempre ao meu lado.

Ao meu orientador Jean Hauck, primeiramente pela entrevista de estágio que cheguei atrasado, mas teve a paciência de me esperar e ensinar a dar os primeiros passos ao meu futuro, e continuou me guiando para fazer este trabalho.

Ao colegas de trabalho do Grupo Specto, pelo conhecimento adquirido no ano que estive trabalhando, e que me conferiram a permissão de utilizar seu software neste trabalho.

Por fim, aos membros da banca, por terem aceitado o convite, agregando seus conhecimentos.

"O Tempo passa, a pessoas mudam... Como o fluxo do rio, isso nunca acaba... O pensamento infantil se torna uma nobre ambição... Uma jovem paixão se torna uma profunda afeição..." - (Sheik)

RESUMO

A área de testes demonstra cada vez mais sua importância nas organizações que desenvolvem software. Ferramentas empregadas servem para trazer uma melhor confiabilidade e agilidade ao que está sendo produzido. Realizar testes é caro e demorado, automatizar testes é uma alternativa interessante e tecnicamente viável, entretanto, para sistemas legados, automatizar os testes pode ser um desafio devido às tecnologias utilizadas. A automação de testes para plataforma Flex possui alguns desafios. Nesse sentido, é desenvolvido neste trabalho, utilizando uma aplicação já existente, casos de teste documentados seguindo as melhores práticas, e a partir desses, foram desenvolvidos scripts de testes que pudesseM realizar os testes na plataforma na qual a aplicação foi desenvolvida. Requisitos foram desenvolvidos para a escolha de uma ferramenta de automação de testes, a fim de escolher a que melhor atendesse as necessidades das aplicações, e que evitasse retrabalho. Um estudo de caso é então planejado e realizado objetivando a automação de testes de um sistema desenvolvido em plataforma Flex. A automação de testes é desenvolvida utilizando um software de automação de ações. No decorrer do trabalho, obstáculos técnicos para automação foram encontrados e superados. Por fim, a automação de testes na plataforma foi implementada para todos os casos de teste solucionados. Dados de esforço e tempo de elaboração e execução dos casos de teste são coletados, bem como um questionário foi desenvolvido com o objetivo de levantar a opinião dos envolvidos na área de teste de software da organização. Por fim, os resultados observados no estudo de caso levantam indícios de que a automação trouxe o benefício de agilizar o processo de testes juntamente com uma maior confiabilidade dos testes ao remover grande parte do erro humano, trazendo menos custo ao projeto.

Palavras-chave: Testes. Integração. Automação. Flex.

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

Com a tecnologia cada vez mais avançada, desenvolvendo e produzindo mais produtos, mais dispositivos e softwares são criados para a comodidade das pessoas, seja no trabalho ou no lazer. Muitas vezes as pessoas não notam o quanto os softwares estão envolvidos em suas vidas, possivelmente porque suas ações são tão automatizadas que se tornam algo trivial no seu dia-a-dia.

Como dito por Sommerville (2007), praticamente todos os países são dependentes de sistemas baseados em computadores, sendo a infraestrutura e serviços nacionais geralmente conectados a computadores. Muitos dispositivos eletrônicos também são conectados a um computador e a um software de controle, levando a manufatura e a distribuição industriais a serem quase completamente automatizadas.

Dada essa dependência atual de softwares, nota-se que a criação de um software deve ser estudada e avaliada, pois mesmo que venha a ser produzido um software gratuito ao público, não será bem aceito se não for de fácil manuseio ou se possuir defeitos nas suas funcionalidades.

Pressman (2011) mostra alguns atributos encontrados na maioria dos Aplicativos Web, como disponibilidade de acesso, desempenho, evolução contínua e segurança. Se um WebApp demorar muito tempo para responder o usuário, por exemplo este pode acabar procurando um outro produto que lhe satisfaça.

Mas então, como garantir a qualidade de software? Tentar garantir que o software está isento de erros é dizer que o programa está perfeito e atente aos requisitos que o cliente precisa sem a necessidade de maiores alterações. Nesse sentido, Rätzmann e De Young (2002), definem qualidade como: "... O agregado de todas as características e propriedades do produto ou atividade que se relaciona com sua sustentabilidade ao encontrar os requisitos específicos".

Mas, o alcance dessa qualidade esperada não tem sido sempre possível: em muitos casos softwares, ainda com erros e falhas, têm sido entregues aos clientes, mesmo que cobrindo os requerimentos mínimos especificados. Uma das maneiras de garantir que a qualidade do produto está sendo atendida, e que as funcionalidades que o cliente necessita estejam presentes e funcionando, pode ser alcançada com testes de software.

Sommerville (2007) também aborda sobre gastos de software na ideia que a integração do sistema e testes são suas rotinas mais caras, com a probabilidade de custarem 40 a 50% do orçamento total do projeto.

Ainda considerando testes, Rätzmann e De Young (2002) passa sua ideia que testes de software são como um jogo, e os vencedores são: o programa sendo testado, este sempre será vitorioso pois suas falhas estão sendo encontradas e está sempre melhorando, e o outro vencedor é a equipe que criou o programa, a empresa.

Entretanto, alguns testes são feitos por pessoas e, portanto, não são perfeitos, são suscetíveis a falhas. Muitas vezes as pessoas acabam deixando alguns pequenos problemas passarem despercebidos ou, também, terem visões diferentes de como fazer os testes.

Algumas das formas de se garantir testes mais homogêneos são a padronização dos testes e a automatização de testes. A padronização dos testes, como por exemplo a documentação sobre casos de testes, com o passo-a-passo de como o teste deve ser feito, diminui a chance dos testes serem feitos de diferentes formas. A IEEE std 829-2008[1] é um documento que define o contexto e formato de documentações que envolvem todo o processo de testes de software, identificando o que deve ser testado, o responsável pelo teste e os riscos associados com o plano estabelecido aos testes.

Rätzmann e De Young (2002) comentam que a estratégia de testes por documentação prove a possibilidade de fazer testes por utilização do sistema. Esta documentação deve ser entregue a pessoas que vão experimentar o programa, para fazer os testes, como um usuário. Isso aumenta as chances destes documentos responderem os interesses do usuário final.

Complementando, Molinari (2003) afirma que o custo total e efetivo do gerenciamento de qualidade é definido pela soma de quatro fatores: prevenção, inspeção, falha interna, e falha externa. Estes são os fatores que define como mais importante. O caso de falha humana se encaixaria na parte de falha externa.

Além do risco de falhas humanas no teste manual de software, Sommerville (2007) ainda adiciona que a fase de testes é uma parte bastante trabalhosa do processo de software, e que ferramentas para testes de software deveriam ser as primeiras a serem desenvolvidas, oferecendo vários recursos e podendo reduzir custos de forma significante.

A automatização de testes tende a diminuir a falha humana nos processos de testes. Assim, um grupo menor de pessoas poderia automatizar alguns tipos de testes que seriam repetidos toda vez que fosse necessário. Então, se um grupo de testadores usar uma mesma automatização, o teste será feito da mesma forma pois foi feito diretamente pela máquina a pedido dos testadores.

[1]<http://segoldmine.ppi-int. Com/content/standard-ieee-std-829-%EF%BB%BFieee-standard-software-and-system-test-documentation>

Nesse sentido, Bartié (2002), explica que a automatização de testes requisita um esforço inicial, mas que fornece rapidez, confiabilidade, e eficiência que não será atingida com facilidade, ou até mesmo não ser atingida, com testes manuais.

Ao se produzir um novo software, é necessária uma documentação dos testes para se ter um controle do que foi feito e do que será feito em relação ao software. Existem algumas ferramentas específicas para isso. Mas, infelizmente, nem todas as ferramentas de automatização de testes possuem uma integração com tais ferramentas. Assim, existe uma chance que esta documentação dos testes automatizados seja perdida, ou que saia de controle.

Assim, a escolha de qual ferramenta de automatização de testes utilizar deve ser feita de forma minuciosa. Precisa-se de uma ferramenta de automatização de testes que possa satisfazer as funcionalidades do sistema para garantir sua qualidade. Nesse sentido, este trabalho de conclusão de curso procura abordar a implementação da automatização de testes de um software desenvolvido na plataforma Flex[2], e avaliar seus resultados.

1.1 Problemática

Testadores de software tem, algumas vezes, a necessidade de testar funcionalidades do software similares, ou então uma nova versão de um mesmo software que precisa ser testada novamente. Isso gera uma repetição de trabalho já realizado, que poderia ser agilizado com a ajuda de automatização de testes.

Além disso, um ponto a se levar em consideração são softwares desenvolvidos em Flex. Flex é um framework Open-source que permite que aplicações webs sejam utilizadas em diversos browsers, desktops, e aparelhos (Flex,2015). Mas, Flex não é como HTML, o que acarreta em dificuldades para automatizar os testes em páginas de sistemas Web que tenham sido desenvolvidos utilizando essa tecnologia. O Apache Flex [3] é uma ferramenta visada a desenvolvedores, pelo que informa a wiki[4] dos dispositivos Adobe e seria necessário encontrar uma ferramenta de automatização de testes que funcione com essa tecnologia.

Tomando por base esses fatores, a pergunta de pesquisa a ser abordada neste trabalho é: seria possível automatizar testes para um software Flex de forma a reduzir o esforço, tempo e

[2]<http://www. Adobe. Com/products/Flex. Html>
[3]<http://Flex. Apache.org/>
[4]<https://wikidocs. Adobe. Com/wiki/display/Flex/Get+oriented+to+Flex>

retrabalho, tomando por base uma ferramenta já existente para modelagem e automatização de testes?

1.2 Objetivos

Aqui são descritos os objetivos, geral e específicos, deste trabalho.

1.2.1 Objetivo Geral

Desenvolver e avaliar a automatização de testes em uma aplicação Web desenvolvida utilizando plataforma Flex.

1.2.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- Identificar as especificidades e limitações de um software Flex no contexto da automação de testes;
- Identificar diferentes abordagem e formas de se testar softwares;
- Analisar ferramentas de automação de testes de softwares desenvolvidas com Flex;
- Desenvolver scripts automatizados de testes para a ferramenta de automação melhor alinhada às características de aplicações desenvolvidas com Flex;
- Avaliar os resultados obtidos por meio de um estudo de caso.

1.3 Justificativa

O crescimento dos produtos de software pode ser notado pela comodidade que confere a pessoas no cotidiano de suas vidas. Isso faz a demanda de softwares de qualidade crescer, levando as empresas que produzem softwares de baixa qualidade a perder seus clientes para outros fornecedores.

Pressman (2011), afirma que, a cerca de cinquenta anos, ninguém poderia prever que softwares seriam incorporados em sistemas de todas as áreas, como transporte, medicina, militar, e entretenimento por exemplo. Entretanto, a qualidade do software as vezes não é garantida e, em alguns casos, são encontrados erros quando o software está nas mãos do cliente.

Em um estudo realizado pela NIST (National Institute of Standards and Technology), em 2002, revelou que cerca de 59,5 bilhões de dólares são gastos anualmente devido a bugs e erros de software. Destes gastos, dizem os estudos, cerca de um terço do valor, ou aproximadamente 22 bilhões, poderiam ser economizados com uma melhoria de testes e infraestrutura que permite uma identificação e remoção de erros mais cedo(NIST,2002).

Em um pensamento comum a ideia de testar um software é algo simples, sendo apenas necessário ter-se o software para que os testes possam ser feitos por qualquer pessoa. Rätzmann e De Young (2002), escrevem que o papel de testador é um visto como um que qualquer pessoa pode preencher quando é chamado para fazê-lo, e que as vezes até alguns dos desenvolvedores acabam se tornando testadores do software que criaram.

Entretanto, essa visão está mudando. Sabe-se que em alguns tipos de testes, como repetitivos ou de carga, erros podem ser encontrados em qualquer momento, desde a etapa de desenvolvimento, até a etapa de testes, e produção, quando o produto estará nas mãos do cliente. Estes testes são dirigidos por pessoas, então são suscetíveis a erros de falha humana. Sana-se estes problemas com, por exemplo, a automatização de testes em um escopo definido anteriormente, com casos de testes providos de ferramentas de modelagem.

Este trabalho se propõe a unir o conhecimento do tema de automação de testes de software ao interesse pessoal do autor como para organizações, melhorando os testes feitos em softwares desenvolvidos por empresas e, assim, tendo um controle dos testes e melhor garantia da qualidade dos produtos desenvolvidos.

1.4 Abordagem metodológica

Esta seção aborda a metodologia utilizada no trabalho, apresentando qual tipo de pesquisa é seguido, suas fases metodológicas, amostra da solução abordada, e delimitações da pesquisa.

1.4.1. Caracterização do Tipo de Pesquisa

Menezes e Silva (2005) descrevem pesquisa como "um conjunto de ações, propostas para encontrar a solução para um problema, que têm por base procedimentos racionais e sistemáticos. A pesquisa é realizada quando se tem um problema e não se têm informações para solucioná-lo".

Lakatos e Marconi (2009) trazem muitos conceitos de métodos, e definem que método é o conjunto das atividades sistemáticas e racionais que, com maior segurança e economia, permite

alcançar o objetivo - conhecimentos válidos e verdadeiros-, traçando o caminho a ser seguido, detectando erros e auxiliando as decisões do cientista.

1.4.2. Pesquisa Qualitativa

Uma pesquisa qualitativa, segundo Lakatos e Marconi (2009), emprega instrumentos estatísticos, preocupando-se em analisar e interpretar aspectos mais profundos, descrevendo a complexidade do comportamento humano, e fornecer uma análise mais detalhada da pesquisa.

Nas investigações conduzidas, será utilizada, dentre outras, uma técnica chamada observação. Lakatos e Marconi (2009), descrevem a técnica como "coleta de dados para conseguir informações utilizadas na obtenção de determinados aspectos na realidade", e citam algumas delas:

- Assistemática
- Sistemática
- Não Participante
- Participante
- Individual e de Equipe

Observações Sistemáticas são observações estruturadas e planejadas, mas não padronizadas pois seus objetivos variam, enquanto que observações Assistemáticas não possuem uma forma de planejamento específico. Observações Participantes são aquelas em que o investigador e o investigado trabalham em conjunto, confiando um no outro para melhor compreensão do produto sem esconder o objetivo da investigação, diferente das observações Não Participantes. Observações Individuais e de Equipe é quando o observador, ou grupo de observadores, projetam sua personalidade sobre o observado, podendo intensificar a objetividade das informações adquiridas nos eventos reais.

O trabalho entra nos aspectos de observações do tipo Sistemática, Participante, e Individual e de equipe, já que a elaboração dos testes é feito de forma planejada, mas não padronizada, os testadores e os desenvolvedores trabalham em conjunto para garantir a qualidade do produto, e o testador, ou grupo de testadores, podem trabalhar em equipe para melhor compreensão dos testes sendo realizados.

De acordo com Sabino (1996), análises quantitativas se efetuam "com toda informação numérica resultante da investigação", que se "Apresentará como um conjunto de quadros, tabelas e medidas".

Essa ideia mostra o foco de pesquisas quantitativas com os resultados obtidos, tornando-os itens mensuráveis. Lakatos e Marconi (2009) dão ideias de conteúdos que podem ser afetados, tais como frequência de aparição de palavras em textos, temas e expressões. Também informam que a pesquisa deve ser sistemática e objetiva:

- Sistemática: deve ser ordenada, metódica;
- Objetiva: deve proceder de forma rigorosa e reaplicável, descrevendo, compreendendo e explicando os fatos analisados.

Esta abordagem de pesquisa possui algumas vantagens e desvantagens, demonstradas no Quadro 1:

Quadro 1 - Vantagens e desvantagens da Abordagem Quantitativa

Vantagens	Desvantagens
Precisão e controle	Excessiva confiança nos dados
Integração dos métodos de quantificação e qualificação	Falta de detalhes do processo e de observação sobre diferentes aspectos e enfoques
Explicitação dos passos da pesquisa	Certeza nos dados colhidos
Prevenção da inferência e da subjetividade do pesquisador	Desenvolvimento com a situação da pesquisa

Fonte: criado pelo autor, baseado em Lakatos e Marconi (2009)

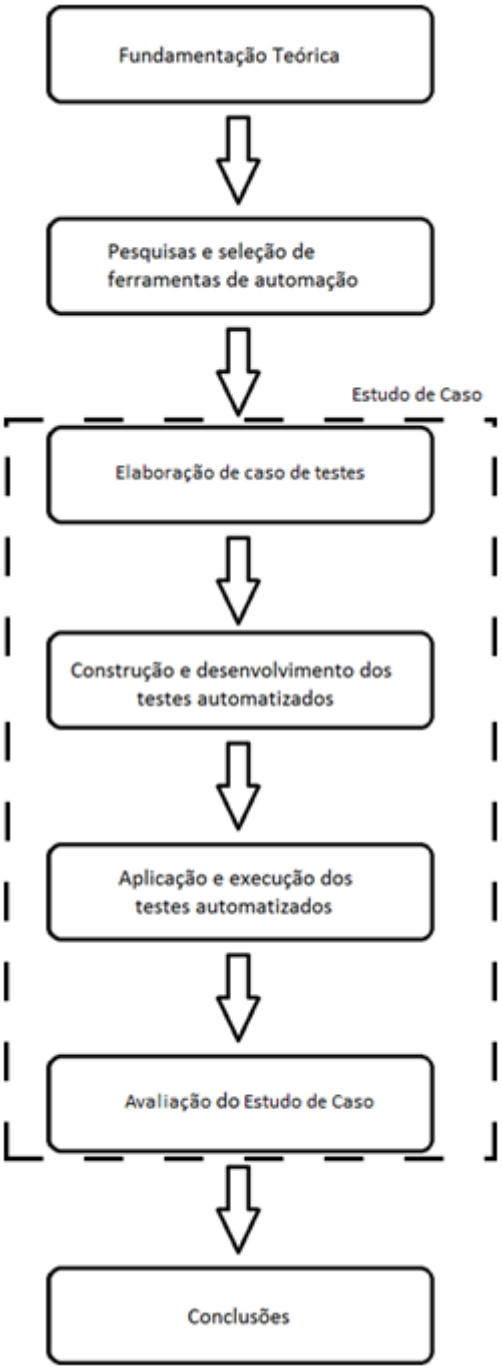
Na comparação das vantagens e desvantagens, nota-se que a pesquisa deve ser feita com cautela, pois qualquer informação errônea dos dados adquiridos afetará os resultados da pesquisa. Assim, a presente pesquisa apresenta características principais de uma pesquisa qualitativa.

1.4.3.Etapas Metodológicas

Nesta seção, serão desenvolvidas as etapas que representam a abordagem metodológica que irá servir de apoio para todo o trabalho.

A Figura 1 - Etapas metodológicas apresenta as etapas metodológicas deste trabalho:

Figura 1 - Etapas metodológicas



Fonte: elaborado pelo autor

A primeira etapa refere-se à fundamentação teórica, sendo especificados os principais conceitos teóricos necessários ao desenvolvimento de um trabalho.

Na segunda etapa, é realizada a pesquisa e seleção das ferramentas de automação de testes, verificando-se qual a melhor ferramenta para uma determinada aplicação. Para isso, são levados em conta seus prós e contras, sendo que a ferramenta que cobrir mais requisitos será a escolhida.

As etapas Elaboração de caso de testes, Construção e desenvolvimento dos testes automatizados, Aplicação e execução dos testes automatizados, e Avaliação do estudo de caso, são realizadas no contexto de um Estudo de Caso, cujos passos metodológicos são melhor detalhados no capítulo 6.

Em sequência, serão elaborados os casos de testes, que serão a base utilizada para a criação dos scripts dos testes automatizados. Na quarta etapa, serão desenvolvidos os testes automatizados, que serão executados na ferramenta que foi definida após a segunda etapa.

Já na quinta etapa, são aplicados os scripts de testes automatizados, utilizando uma ferramenta de modelagem como local de armazenamento e uma ferramenta que execute estes scripts.

Na última etapa, será realizada a avaliação do estudo de caso e, por fim, desenvolvida a conclusão do trabalho.

1.4.4. Delimitações

Este trabalho apresenta delimitações, descritas a seguir, como:

- Não será desenvolvida uma aplicação web. O presente trabalho utilizará uma já existente no estudo de caso;
- O teste automatizado será exclusivo da aplicação que será utilizada como base;
- Toda e qualquer alteração no sistema poderá intervir diretamente nos testes;
- Os casos de testes criados neste trabalho serão limitados a testes de Sistema, Caixa Preta e Automatizado.
- Limitação ao Apache Flex SDK versão 3.5.

1.5 Estrutura do trabalho

O presente trabalho está dividido da seguinte maneira:

Capítulo 1 – Apresenta a introdução do assunto, sua problemática, justificativa, e objetivos geral e específicos.

Capítulo 2 – Apresenta a revisão bibliográfica, tendo como foco a área de testes de software e sua automatização.

Capítulo 3 – Apresenta o Estado da Arte, relatando experiências similares na literatura atual para aprender com outros autores.

Capítulo 4 – Apresenta o estudo de caso de automação de testes proposto no trabalho, instrumentos de coleta de dados, e a aplicação do estudo de caso.

Capítulo 5 – Apresenta a execução do estudo de caso, relatando a experiência da aplicação de automação, abordando mais sobre a ferramenta escolhida, e relata os resultados obtidos.

Capítulo 6 – Apresenta conclusão do trabalho, dando as últimas informações sobre todo o trabalho realizado, o resultado final, e ideias de trabalhos futuros.

Capítulo 7 – Apresenta a bibliografia utilizada na pesquisa e desenvolvimento do trabalho.

2. FUNDAMENTAÇÃO TEÓRICA

Este capítulo dedica-se a apresentar a fundamentação teórica a respeito dos temas que cercam este trabalho. Dentre eles são: qualidade de software, teste de software, tipos de teste de software, processos de teste de software e informações sobre normas e modelos relacionados.

2.1. Qualidade de Software

As decisões tomadas no andamento de um projeto de software influenciam no resultado final de sua qualidade. Assim, o conjunto de resultados dos esforços feitos durante o desenvolvimento do software será refletido no produto final. Muitas dessas decisões em um projeto de software estão relacionadas à forma como será tratada a qualidade do produto final.

Segundo Pressman (2011), a qualidade de software pode ser definida como "conformidade para explicitar o estado funcional e requerimentos de performance, explicitar padrões de desenvolvimento documentados, e implicitar características que são esperadas de todo desenvolvimento profissional de software", e utiliza a definição para enfatizar três pontos importantes:

- Uma gestão de qualidade efetiva pode estabelecer a infraestrutura que dá suporte a qualquer tentativa de construir um produto de software de alta qualidade.
- Um produto útil que fornece o conteúdo, as funções e os recursos que o usuário final deseja, além disso, e não menos importante, deve fornecer confiabilidade e isenção de erros.
- Um software de alta qualidade gera benefícios para a empresa de software bem como para a comunidade de usuários finais.

Para uma melhor definição sobre Qualidade de Software, a ISO/IEC/IEEE 24765[5] (2010), descreve *Software Quality* como "A capacidade do software de satisfazer necessidades implícitas e explícitas em condições especificadas".

O SWEBoK[6](2014), traz a definição de outros autores e também explica que, atualmente, qualidade de software é definida tanto da forma descrita pela IEEE 24765 quanto como "O grau em que o software se encontra com os requisitos estabelecidos; porém, a qualidade depende do

[5]ISO/IEC/IEEE 24765 - System and Software engineering - Vocabulary
[6]SWEBoK - Guide to Software Engineering Body of Knowledge

grau que os requisitos foram precisamente estabelecidos que representam o que os interessados necessitam, querem, e esperam".

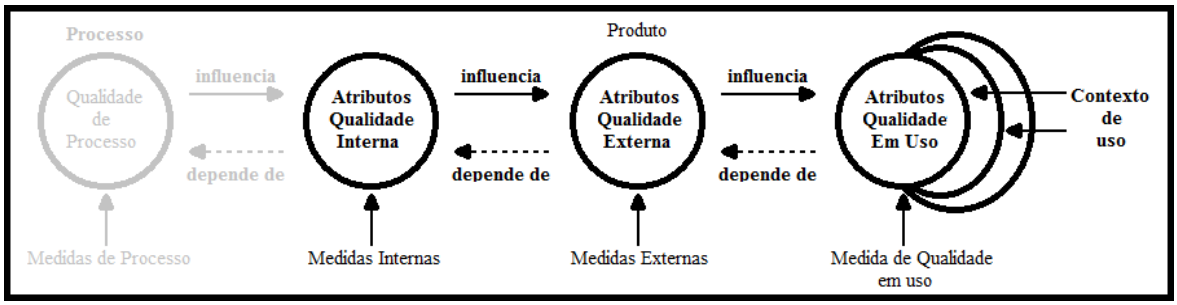
Conforme citado, nota-se a importância da qualidade de software, já que tanto o fornecedor quanto o cliente se beneficiam com isso, pois o software possuirá uma qualidade superior e os clientes, que são os interessados, terão consigo a ferramenta que precisam para suas necessidades. Existem diferenças entre qualidade de processo e qualidade de produto.

Qualidade de processo foca na não conformidade com as auditorias, o planejamento do produto em si, enquanto a qualidade de produto foca no produto visando sua versão final, pronto para ser entregue ao usuário (PRESSMAN, 2011).

Este trabalho foca na qualidade de produto, sendo realizada uma automatização de testes, sem levantar requisitos de como será feito o planejamento inicial ou o código fonte, mas é importante conhecer a qualidade de processo para se garantir uma qualidade superior do produto.

A **Error! Reference source not found.** mostra com clareza um ciclo de vida da qualidade de software proposto pela norma ISO/IEC 25010[7] (2011), mostrando as diversas influências entre Processo e Produto, demonstrando as influências e dependência de cada etapa.

Figura 2 - Ciclo de vida da Qualidade de Software



Fonte: adaptado da ISO 25010 (2011)

Os elementos que formam o ciclo proposto na norma ISO 25010 (2011) serão apresentados em detalhes nas próximas sessões.

2.1.1. Qualidade de Processo

Antes da explicação de qualidade de processo, é necessário entender "o que é um processo", e também"o que é um processo de software". O Instituto de Engenharia de

[7]<ISO/IEC 25010 - Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models>

Software[8](*Software Engineering Institute*, 1994) define processo como "uma sequência de passos realizados para um determinado propósito", enquanto Zahran (1998) define processo de software como "um conjunto de atividades que se deve cumprir, numa ordem estabelecida, para desenvolver e manter software e produtos associados a ele".

A Qualidade de Processo é o primeiro elemento do Ciclo de vida da Qualidade de Software pela ISO 25010. Este elemento especifica os processos que serão realizados na produção do produto, onde é feito o planejamento do produto, especificando os resultados esperados e sua faixa de tolerância. A não conformidade dos resultados obtidos com os resultados planejados fere a ideia da qualidade de processo.

Bartié (2002) passa a ideia de que é na qualidade de processo que devem ser avaliados todos os documentos que serão gerados durante o desenvolvimento do produto, como requisitos levantados, modelos e especificações de negócios, arquitetura visita, e modelo de dados e classes.

A Figura 2 mostra que, mesmo sendo duas qualidades diferentes, a qualidade de produto depende da qualidade de processo, já que é influenciada por ela. Koscianski (2007) reforça a ideia refletida pela imagem, pois sem um planejamento da qualidade de processo, perde-se tempo, mais erros irão aparecer conforme o produto é desenvolvido, e seu resultado final terá uma qualidade instável.

Analisando a figura, nota-se a importância e dependência do produto em relação a sua qualidade de processo, similar a como Rätzmann e De Young (2002) descrevem sobre testes de software: sem um planejamento adequado, não será possível uma comparação ou visão inicial do resultado esperado.

2.1.2. Qualidade de Produto

A qualidade de Produto, egundo elemento em diante do ciclo de vida da Qualidade de Software pela ISO 25010 (vide Figura 2), focado no produto, onde são definidas suas características relevantes e definidos os atributos e variáveis que devem conter. O objetivo de qualquer empresa, grupo ou organização é que seu produto final possua qualidade e o mínimo de queixas possível.

A ISO 25010 divide a qualidade de produto em três diferentes tipos que, como a qualidade de processo, dependem de e influenciam as outras etapas:

[8]<SEI - Software Engineering Institute>

2.1.2.1. Atributos de Qualidade Interna

É a capacidade de um grupo de atributos estáticos do produto satisfazer necessidades, implícitas e explícitas, quando for utilizado em condições específicas pelos usuários, segundo a ISO/IEC 9126 (2001). As qualidades internas de um produto de software são relacionadas ao seu código fonte. Então é necessário analisar e verificar este código fonte antes de procurar por bugs, e todas as etapas pelas quais um código fonte passa ao ser desenvolvido passam a ser relevantes nesse processo.

De acordo com SWEBoK (2014), desenvolver um produto de software envolve as principais etapas:

- Design: conceito ou invenção de uma forma para tornar os requisitos do usuário em um software operacional. É a atividade que liga os requerimentos do aplicativo para a codificação de debug do programa;
- Escrita: é a codificação do design em sua linguagem de programação apropriada;
- Teste: atividade para verificar se o código escrito faz o que deveria fazer;
- Debug: atividade para encontrar e arrumar bugs no código fonte, ou design;
- Manutenção: atividade para atualizar, corrigir e melhorar programas.

Ainda no SWEBoK (2014), o processo de desenvolvimento de um produto de software requer atenção em várias áreas, como o uso de uma estrutura de dados apropriada, por exemplo. Essa afirmativa reforça a influência da qualidade de processo sobre o desenvolvimento do código fonte.

2.1.2.2. Atributos de Qualidade Externa

É a verificação do resultado da qualidade interna, onde é realizada a verificação de bugs e, nos casos de uso do software, fatores externos são aqueles cuja presença ou falta num produto de software pode ser detectada pelos usuários do produto. Antes do produto ser liberado aos usuários, o produto seria verificado pela empresa contratada e teria a garantia que pelo menos os requisitos explícitos estariam funcionando adequadamente.

Uma das formas para garantir a qualidade externa do produto é com testes de software. Sommerville (2007), cita que a meta dos testes de software é convencer os desenvolvedores e clientes do sistema que o software é bom o suficiente para o uso operacional, e que o teste é um

processo voltado a atingir a confiabilidade de um software. O tópico de testes de software é discutido na 2.2 deste trabalho.

2.1.2.3. Atributos de Qualidade em Uso

É a verificação do produto quando está em uso, ou seja: está nas mãos do cliente. Esta é a última etapa da qualidade de produto. Similar aos atributos de qualidade externa, são feitas as verificações de bugs, unidas ao feedback dos usuários. É a qualidade pelo ponto de vista do usuário.

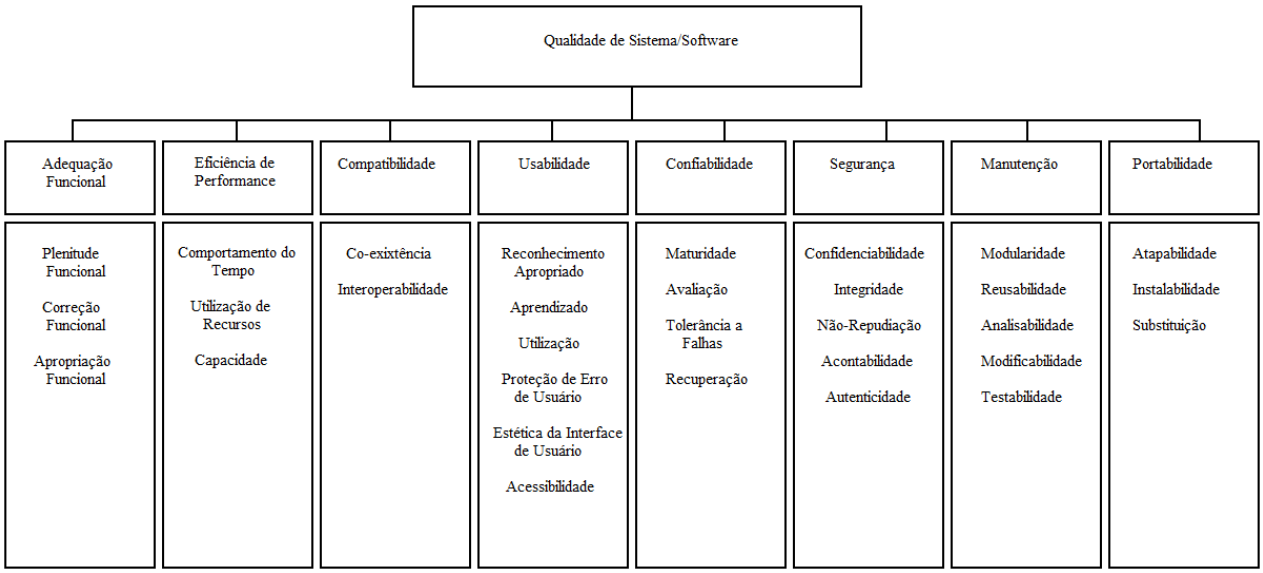
É como o passo anterior de qualidade externa, mas adicionando-se o usuário como uma fonte de dados. A ISO/IEC 9126(2001), substituída e melhorada pela ISO/IEC 25010 (2010), define os objetivos da qualidade em uso em quatro características:

- Eficiência: capacidade do produto de software de permitir ao usuário atingir metas específicas como completude, em um contexto de uso específico;
- Produtividade: capacidade do produto de software de permitir que seus usuários empreguem quantidade adequada de recursos em relação à efetividade alcançada em um contexto de uso específico;
- Segurança: capacidade do produto de software de apresentar níveis aceitáveis de riscos de danos a pessoas, negócios, software, propriedade ou ambiente em um contexto de uso específico;
- Satisfação: capacidade do produto de software de satisfazer usuários em um contexto de uso específico.

Estas características são avaliadas para verificar se o produto atende as necessidades de seus usuários.

De forma a auxiliar a especificação e avaliação das qualidades de software de um produto, a ISO/IEC 25010(2011), define um framework dividido em oito características que permitem uma abstração de sobre a definição de qualidade de software, conforme mostra a **Error! Reference source not found.**:

Figura 3 - Qualidade de Sistema de Software



Fonte: adaptado da ISO 25010 (2011)

- Adequação Funcional: definida como o grau na qual o software ou sistema provê funções que atendem necessidades determinadas e implícitas quando usadas em condições específicas;
- Eficiência de Performance: definida como a performance relativa à quantidade de recursos usados em condições específicas;
- Compatibilidade: definido como o grau no qual um produto, sistema ou componente pode trocar informação com outros produtos, sistemas ou componentes, e/ou realizar suas funções requeridas, enquanto divide o mesmo ambiente de hardware e software;
- Usabilidade: definido como o grau na qual um produto ou sistema pode ser usado por usuários específicos para alcançar objetivos com efetividade, eficiência, e satisfação em um contexto de uso específico;
- Confiabilidade: definido como o grau na qual o sistema, produto ou componente realiza funções específicas em condições específicas por um período de tempo específico;
- Segurança: definido como o grau na qual um produto ou sistema protege informação e dados para que pessoas ou outros produtos ou sistemas tenham um grau de acesso a dados apropriados para seu tipo e nível de autorização;
- Manutenção: definido como o grau de efetividade e eficiência com o qual um produto ou sistema pode ser modificado pelos devidos técnicos;

- Portabilidade: definido como o grau de efetividade e eficiência com o qual um sistema, produto ou componente pode ser transferido de um hardware, software ou outro operacional ou ambiente de uso para outro;

Essas definições de qualidade de software pela norma ISO/IEC 25010 são importantes para um melhor detalhamento da qualidade de software de dado produto. Essas características serão descritas mais adiante, onde será explicado sobre suas sub-características mais a fundo.

2.1.3.Níveis de Integridade do Produto de Software

A norma IEEE 829 (2008) define diversos níveis de integridade para produtos de software. Cada nível de integridade estabelece características que ajudam a definir a abrangência necessária da documentação de teste que deve ser considerada, com exemplos da documentação, com o que ela aborda, que podem ser usados em cada teste.

Os níveis de integridade definidos pela norma IEEE 829 (2008) são informados no **Error! Reference source not found.**, de acordo com seu nível de integridade e potencial de mitigação (ações para diminuir o risco ao diminuir a probabilidade do risco da ocorrência de um evento ou reduzindo o efeito caso ocorra):

- Catastrófico: o software deve executar corretamente ou graves consequências (perda de vida, perda do sistema, danos ao ambiente, perda econômica ou social) irão acontecer. Não é possível mitigar estes efeitos;
- Crítico: o software deve executar corretamente ou o uso esperado do software/sistema não será realizado, causando sérias consequências (ferimentos permanentes, grave degradação do sistema, dano ambiental, impacto econômico ou social). É possível mitigar completamente ou parcialmente estes efeitos;
- Marginal: o software deve funcionar corretamente ou o uso esperado do software/sistema não será realizado, causando leves consequências. É possível realizar mitigação completa;
- Negligenciável: o software deve funcionar corretamente ou o uso esperado do software/sistema não será realizado, causando consequências negligenciáveis. Não é necessário mitigar estes efeitos.

Com toda informação encontrada e analisada, nota-se a importância de garantir a qualidade do produto, garantindo assim uma satisfação aceitável pelo cliente e pelo fabricante. Uma das formas encontradas para isso é por meio da execução de testes de software, que são apresentados em detalhes na próxima seção.

2.2. Teste de Software

Esta seção trata de um dos principais pontos deste trabalho, pois um teste automatizado, independente de qual plataforma opere, ainda sim está definido como um teste de software. Um teste de software é uma verificação do software em situações especificadas, como realizar um login de sistema com usuário inválido, mas, deve-se pensar também "quais os objetivos dos testes de software " e "o que realmente constitui um teste de software?".

Sommerville (2007), detalha que testes de software possuem duas metas distintas:

- Demonstrar ao desenvolvedor e ao cliente que o software atende aos requisitos;
- Descobrir falhas ou defeitos no software que apresenta comportamento incorreto, não desejável ou em não conformidade com sua especificação.

O primeiro ponto indicado por Sommerville está mais voltado ao lado comercial dos produtos, já que garante ao desenvolvedor que ele está no caminho certo e que não precisa gastar tempo corrigindo problemas, economizando tempo que pode ser aplicado em melhorias, e mostra ao cliente que o que está sendo adquirido realmente funciona como esperado, que ele não terá nenhuma surpresa indesejada.

Já o segundo ponto citado é mais voltado ao lado de garantir que o produto está livre de comportamentos indesejáveis, como travamentos, interações indesejadas de outros sistemas, ou cálculos incorretos. Os desenvolvedores são humanos e estão suscetíveis a falhas, então os testes estarão sendo feitos para garantir que estas falhas sejam detectadas e corrigidas.

Sobre o que constitui um teste de software, Rätzmann e De Young (2002) definem que um teste inclui três etapas:

- Planejamento: passo em que se determina o que será testado, identificando casos de testes, especificando as atividades e indicando o resultados que devem ser definidos. É uma explicação simples do essencial da ideia do que deve ser testado, de como proceder e o que será testado e quando. Deve-se também ter uma ideia do resultado esperado para os testes.
- Execução: passo que também pode ser dividido em três etapas:
    - Preparo do ambiente de teste: é importante preparar o ambiente de teste para poder ser reproduzido, especialmente se o teste deve ser repetido múltiplas vezes, como uma verificação de performance após melhoria de

sistema feitas por um software: é melhor ter o ambiente preparado com a melhoria em todos os casos que os testes serão feitos

- Completar o teste: é a definição de como o teste será feito, como por exemplo se ele será feito através de uma interface, ou por um testador de forma interativa, ou seria um teste automatizado com Scripts ou baseado na interface.
- Determinar os resultados: Com o auxílio de algumas ferramentas, é possível documentar uma descrição do testes, seus passos, resultados esperados e resultados obtidos, tendo assim uma referência se o sistema está funcionando da forma esperada ou não, e assim o produto pode ser corrigido ou melhorado.

- Avaliação: passo onde se compara o resultado esperado, determinado na etapa de planejamento do teste, com o resultado obtido. Não é possível determinar se o resultado obtido é correto ou não se não se tem uma ideia inicial de como o resultado correto deveria acontecer no sistema.

Estas afirmações fazem sentido, pois testar um produto sem um preparo adequado não possui uma base de comparação do resultado final. Ao não se executar o teste não se possui resultado, e executar um teste e não avaliar o resultado faz com que o esforço de planejar e executar o teste não tenha servido para nenhum propósito: não se sabe se o resultado foi benéfico ou não ao produto.

Rätzmann e De Young (2002), definem, também que, caso algum desses passos esteja faltando, o processo dificilmente pode ser definido como um teste, e passam uma dica de sempre levar em consideração erros de testes anteriores, pois eles podem ser úteis durante o planejamento de novos casos de testes.

2.2.1. Tipos de teste de Software

Existem várias abordagens de teste que podem ser realizadas, com suas vantagens, desvantagens e limitações.

Rätzmann e De Young (2002), informam sobre três métodos de aproximação de testes baseados em função e estrutura, conforme a Figura 4 **Error! Reference source not found.Error! Reference source not found.Error! Reference source not found.**abaixo:

Figura 4 - Aproximação de testes Baseados em Função e Estrutura

Fonte: criado pelo autor, com base em Rätzmann e De Young (2002)

O método de aproximação de testes depende de quanto se sabe do módulo interno da aplicação, como o comportamento de entrada e saída de dados, e sua estrutura.

2.2.1.1. Teste Caixa Preta

Um teste Caixa Preta (do inglês *Black box*) é um teste voltado aos requisitos ou funções do sistema, mas sem verificar o código de software. As entradas e saídas são verificadas, sendo especificado no processo de planejamento do teste, esperando a obtenção do resultado que, caso não tenha sido previsto é considerado como erro.

Bartié(2002), define que o objetivo deste teste não é verificar os processos internos do software, mas sim garantir que o algoritmo utilizado gera resultados esperados.

Rätzmann e De Young (2002), informam que o desenvolvedor deve fazer estes testes antes do lançamento do produto, pois assim os erros e bugs podem ser encontrados antes que o usuário os encontre.

2.2.1.2. Teste Caixa Branca

Também conhecido como teste Estrutural, Caixa de Vidro ou caixa limpa, o teste Caixa Branca (do inglês *White box*) se diferencia do teste Caixa Preta no quesito de conhecimento do software, aqui o código de software é conhecido, e os casos de testes são feitos baseados nisso.

Enquanto o teste Caixa preta se volta a verificar o resultado das saídas do produto, este teste foca em verificar o que acontece dentro do produto.

De acordo com Rätzmann e De Young (2002), este teste faz ser possível garantir que o comportamento está correto sem a necessidade de verificar a interface do programa, mas pode ser verificado na estrutura de dados do produto, pois este ponto é conhecido pelo testador.

Sommerville (2007), descreve que testes estruturais são derivados do conhecimento da estrutura e implementação do software, por sua distinção do teste Caixa Preta.

2.2.1.3. Teste Caixa Cinza

Um teste Caixa Cinza (do inglês *Gray box*) é um conjunto de pontos do teste Caixa Preta com o teste Caixa Branca: Mistura-se a preocupação com a precisão e dependência de entradas e saídas do teste Caixa Preta, mas possuído o conhecimento dos dados internos, fluxos e estruturas do teste Caixa Branca.

Ainda com Rätzmann e De Young (2002), é dito que é o método de escolha para um teste de aplicação rápida, pois testes são assim limitados a testes funcionais, onde avaliação, confiança, performance, e robustez são funções de software, mas ainda assim é possível ter ideias para casos de testes efetivos por culpa do conhecimento da estrutura interna e seus riscos inerentes.

2.2.1.4. Testes de Software de acordo com a Granularidade

A granularidade de testes de software é pela divisão de o quanto o teste está verificando o sistema.

2.2.1.4.1. Teste de Unidade

Teste focado no esforço e verificação de componentes ou módulos, guiados por componentes e caminhos de controle que são usados para descobrir erros nos limites destes componentes ou módulos, validando seus dados de entrada e saída como válidos ou inválidos.

Pressman (2011) detalha que a interface é testada para assegurar quais informações irão fluir corretamente para dentro e para fora da unidade sendo testada, examinando-se a estrutura de dados local para garantir que o armazenamento temporário de dados esteja com sua integridade durante estes passos do algoritmo.

2.2.1.4.2. Teste de Componente

Teste de componente é feito em cada componente individual do sistema, cobrindo mais do que o teste de unidade.

Nunes (2006), diz que nesta etapa de testes os testadores irão visualizar o componente como uma parte do sistema e testá-lo separadamente, visando integrá-lo com outras partes. Para que este tipo de testes possa ser realizado com sucesso, o sistema de software precisa ter sido desenvolvido de forma componentizada.

2.2.1.4.3. Teste de Integração

O teste de integração funciona a partir de unir os testes de Unidades e Componentes. Sozinhos, eles funcionam corretamente, mas ao serem unidos por uma interface, podem acabar não funcionando da forma esperada. Neste teste a arquitetura de software é feita junto dos testes no intuito de descobrir os erros que se associam a sua interface.

Pressman (2011) explica que os dados podem ser perdidos através da interface, ou um componente deve ter um efeito inesperado ou adverso quando trabalha junto de outro componente, e até mesmo que subfunções quando combinadas podem não realizar sua função desejada.

2.2.1.4.4. Teste de Sistema

Teste feito de forma ampla, envolvendo todo o sistema do produto sendo criado (incluindo hardware, firmware e software), com a preocupação voltada aos aspectos gerais do produto. Rätzmann e De Young (2002) incluem em uma checklist os seguintes pontos:

- Completude funcional do sistema ou módulo adicional;
- Comportamento em diferentes configurações de hardware ou sistemas operacionais;
- Instalação e configuração em vários sistemas;
- Limites de capacidade, como tamanho máximo de arquivos, número de usuários concorrentes;
- Comportamento de resposta a problemas no ambiente de programação;
- Proteção contra acessos não autorizados a dados.

2.2.1.5. Teste de Aceitação

É tipicamente o último passo de processo de testes, necessária a participação do usuário/fornecedor de requisitos, ou de um representante dele, para a aprovação do teste.

Rätzmann e De Young (2002) detalham que os casos de testes devem ser baseados nas especificações de requisitos do cliente, prevenindo que este teste de aceitação formal seja um teste contrário a usabilidade do software.

Segundo Bartié (2002) os procedimentos de aceitação devem ser realizados a cada ciclo de implementação da solução, permitindo correções antecipadas. Também informa que neste momento o sistema está disponível para o usuário verificar se as funcionalidades previstas estão disponíveis no sistema.

2.2.1.6. Teste Não Funcional

Um teste não funcional aborda mais de uma categoria de testes, mas não tem objetivo de verificar funcionalidades ou requisitos, diferente dos testes caixa preta e caixa branca.

Os testes se baseiam a verificar itens na usabilidade do sistema e processo de instalação do produto, por exemplo.

2.2.1.7. Teste de Carga ou Performance

Teste que é voltado a medir a performance do produto, o quão rápido ele completa trabalhos específicos, avaliando a velocidade de processamento, verificando se ela é aceitável no programa.

Rätzmann e De Young (2002) dizem que, como regra, estes testes são fáceis de serem automatizados, e a captura e automatização de re-execuções em testes elimina variações em tempo de resposta.

2.2.1.8. Teste de Limites

O teste de limites é voltado a realizar testes nos limites dos valores de inserção de dados: o valor mínimo e o valor máximo, pois qualquer valor entre eles será aceitável.

Para o valor mínimo e o valor máximo, é encessário tester três valores: menor, igual, e maior, para garantir sua funcionalidade. Se uma variável do sistema, que pode ser criada pelo usuário, deve conter entre 5 e 50 caracteres, o teste de limites verificará se é possível criar esta variável com 4, 5, e 6 caracteres para o valor mínimo; 49, 50, e 51 caracteres para o valor máximo.

2.2.1.9. Teste de Stress

O teste de stress é voltado a analisar a performance do produto conforme sua carga de dados, como requisições de usuários, dados sendo processados, a frequência na qual o produto recebe uma mensagem, entre outros, aumentam.

Rätzman e De Young (2002), também definem este teste como Teste de Robustez, determinando quando o programa irá falhar à medida que a carga aumenta, até conseguir determinar este momento, também informam que comportamento do produto nestas situações é observado para comparações.

2.2.1.10. Teste de Instalação

O teste de instalação verifica se o programa pode ser instalado e desinstalado nas plataformas que foram designadas para possuírem suporte. Sua situação mais adequada é quando é possível que o cliente participe junto, podendo assim validar o produto.

Na visão de Rätzman e De Young (2002), uma ideia seria instalar versões antigas do produto para assim verificar a atualização mais recente do produto.

Mas segundo Bartié (2002), existem outras formas de simular a instalação do programa neste teste:

- Instalar o produto pela primeira vez;
- Instalar o produto em um ambiente que já possui o produto instalado;
- Instalar o produto em outros ambientes;
- Verificar se a instalação atente os requisitos do cliente.

2.2.1.11. Teste Manual

Testes manuais são aqueles que precisam da interação humana para serem realizados.

Bartié (2002) comenta que esta categoria iria requisitar um número grande de digitadores, com um controle eficiente de documentação que, uma vez alterados, devem ser atualizados.

Seguindo ainda as palavras de Bartié (2002), esta categoria possui um grande risco na área de incidência de erros humanos, tais como a não execução dos procedimentos de testes, ou valores digitados erroneamente.

Em sua visão, Molinari (2003) descreve que se deve estar atento as seguintes observações para os testes manuais valerem a pena:

- Testes em instalações, setups e/ou operações muito especificas;
- Testes de configuração e de compatibilidade;
- Teste de tratamento de erros;
- Teste de usabilidade;
- Documentação.

2.2.1.12. Teste Automatizado

Como o próprio nome sugere, testes automatizados são aqueles onde são utilizadas ferramentas que simulam as ações de usuários para obter seus resultados nos variados tipos de testes, substituindo a interação humana.

Bartié (2002) diz que o ganho de tempo, controle e confiabilidades nos testes são percebidos à medida que os testes são re-executados, demonstrando suas vantagens sobre testes manuais.

2.2.2. Classificação dos Testes Realizados Neste Trabalho

Após feita a análise dos mais diversos tipos de testes existentes, pode-se determinar que este trabalho envolverá será a realização de um teste de sistema, aceitação, caixa preta, e automatizado, conforme explicado no Quadro 2:

Quadro 2 - Classificação de Testes Realizados

Tipo de Teste	Justificativa
Sistema	O teste envolverá todo o sistema do software que será testado
Caixa Preta	O teste não irá envolver o conhecimento interno do software
Automatizado	O teste será automatizado pois será realizado sem intervenção humana direta para agregar as vantagens de uma possível redução de esforço e também reduzir o erro humano

Fonte: criado pelo autor

2.2.3. Estratégias de testes

Neste tópico, é descrita uma parte importante no processo de teste de software: como testar.

Rätzmann e De Young (2002) descrevem que o papel de testador é um papel onde ainda é visto que qualquer pessoa pode ocupar quando requisitado, seja depois de concluída a tarefa em desenvolvimento, ou em momentos convenientes do projeto, estes testes acabam as vezes se tornando trabalho do desenvolvedor de software.

Esta ideia de que o teste pode ser feito por qualquer pessoa, pela experiência de trabalho e pesquisas feitas neste trabalho, é uma visão imprecisa. Um teste manual feito por alguém com nenhuma experiência é mais provável de se deparar com resultados incoerentes, devidos a erros humanos, ou relatórios do resultado dos testes, ou até mesmo os casos de testes, podem ser escritos e descritos de forma incoerente, trazendo ainda mais dúvidas ao problema para outras pessoas que podem vir a verificar o relatório ou os casos de testes.

De acordo com a ISO 829 (2008), existem alguns pontos importantes a serem destacados:

O gerenciamento de testes está ativamente monitorando e avaliando todos os resultados de testes, sendo uma atividade que é praticada em todo o ciclo de vida de processos e atividades;

- O planejamento de testes define que os participantes devem participar na iniciação de um requerimento para proposta, preparação de resposta, planejamento de contrato, controle e execução, revisão e avaliação, e entrega e finalização das atividades;
- Os testes de atividades envolvem os componentes da integração de software, testes de aceitação de software, integração de sistema e testes de aceitação de sistema, com o objetivo de verificar se os requisitos de software e sistema são satisfeitos pela execução da integração de componentes, sistema, e testes de aceitação. Versões ou releases do software são testados, pela progressão (testar tanto as novas atividades quanto as corrigidas) e regressão (testar que nenhuma alteração não intencional ocorreu).

Ainda de acordo com a norma ISO 829(2008), focando-se nos testes de atividades, é recomendado as atividades de testes abaixo como atividades mínimas de requisito:

- Executar testes de integração de componentes;
- Avaliar o resultado dos testes de integração de componentes;
- Preparar o relatório dos testes de integração de componentes;
- Executar testes de sistemas;
- Avaliar o resultado dos testes de sistemas;
- Preparar o relatório dos testes de sistemas;

- Executar testes de aceitação;
- Avaliar o resultado dos testes de aceitação;
- Preparar o relatório dos testes de aceitação;
- Identificar os riscos;
- Identificar os problemas de segurança.

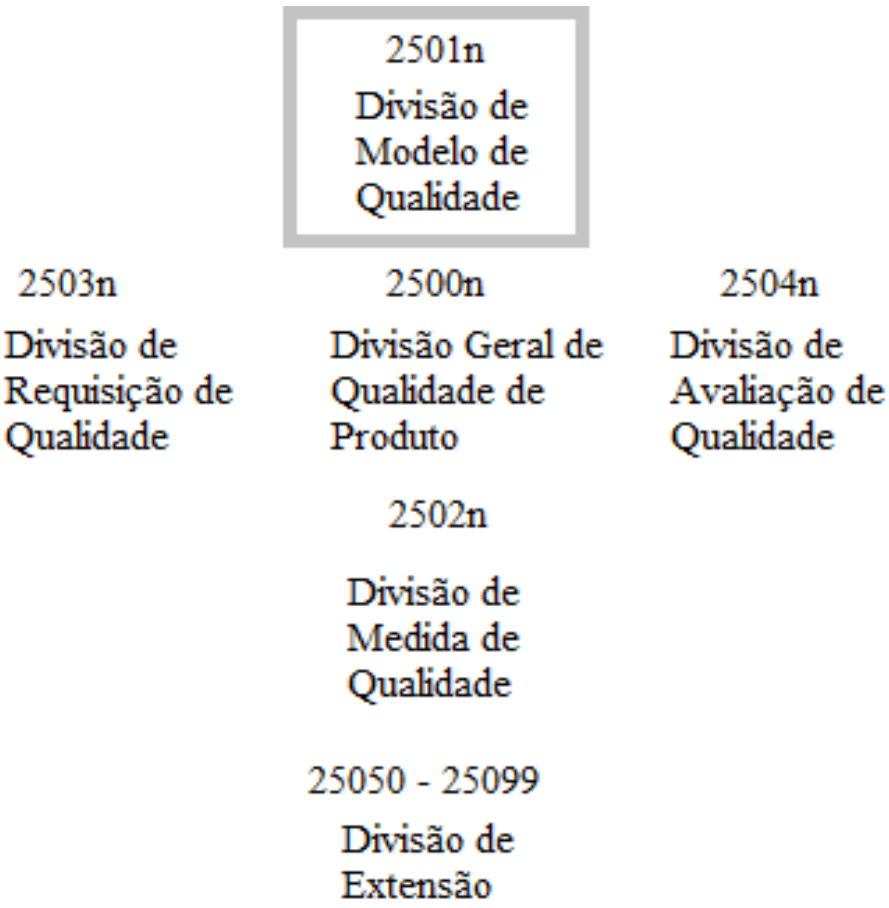
2.3. Normas e Modelos de Referência para Qualidade de Produto de Software

Nesta seção são apresentadas as Normas e Modelos de Referência usados em qualidade de produto de software que serão usadas como base neste trabalho.

2.3.1. ISO/IEC25000

A ISO/IEC 25000-2008, conhecida como SQuaRE (*Software product Quality Requirements and Evaluation*), é o projeto de requisição e avaliação da qualidade do produto que gerou a própria família de normas da série 25000, que é dividida nas normas mostradas na Figura 5:

Figura 5 - Organization of SQuaRE series of standards



Fonte: adaptado da ISO/IEC 25000-2008

Para o trabalho em questão, as normas mais relevantes são as normas 2501n e 2502n, com um foco maior na 2501n:

- 2501n: Divisão de Modelo de Qualidade: Detalha modelos de qualidades de software, qualidade em uso, e dados. A norma é um guia prático no uso da qualidade de software;
- 2502n: Divisão de Medida de Qualidade: Inclui referências e modelos de medidas da qualidade do produto de software, definições matemáticas da medida de qualidade, e um guia prático para sua aplicação.

2.3.1.1. Norma 2501n

A Norma 2501n-2010 é a Norma referente ao Modelo de Qualidade do projeto SQuaRE. Foca-se na definição de qualidade de software, como foi mostrado na **Error! Reference source not found.** na Seção 1, onde divide-se a qualidade de software em oito características já definidas.

De acordo com a norma 25010 (ISO, 2011), as características deste modelo são aplicadas em qualquer tipo de software, onde suas características e subcaracterísticas provêm terminologias consistentes para a qualidade de produto de software, e também características nas quais os requisitos de qualidade podem ser comparados para uma melhor completude.

Algumas atividades podem ser beneficiadas pelo uso do modelo de qualidade durante o desenvolvimento do produto de software:

- Validar a compreensão das definições de requisitos;
- Identificar os objetivos do Design de Software;
- Identificar os objetivos do Teste de Software;
- Identificar o critério de aceitação de qualidade.

A importância da aplicação da norma é percebida pela organização e diferencial gerado nos projetos, pois o conhecimento e aplicação da qualidade de software resultam em um melhor produto de software.

2.3.2.IEEE 1012

A IEEE 1012 (IEEE, 2012), conhecida como norma 1012 - Verificação e Validação, detalha um dos principais passos do ciclo de vida de software (IEEE, 2012).

A norma de Verificação e Validação (V&V), de acordo com sua documentação, é um padrão de processo endereçado a todo ciclo de vida de sistemas e softwares, incluindo o consenso, habilitação organizacional de projetos, projetos, implementação de software, suporte de software, e grupo de processo de reuso de software. A norma informa, porém, que nem todos estes processos são listados nesta norma.

Os processos de V&V determinam quando o desenvolvimento de produto de uma dada atividade e quando o produto satisfaz as necessidades do usuário e seu propósito. Estas determinações podem influir na análise, avaliação, revisão, avaliação, e teste de produtos e processos.

Esta norma define os processos de verificação e validação que são aplicados ao desenvolvimento de sistema, software, e hardware pelo ciclo de vida, incluindo: aquisição, demanda, desenvolvimento, operações, manutenção, e remoção.

Segundo a norma IEEE 1012:

- Verificação: Provê evidencias objetivas para quando o produto realiza alguma das ações:
    - Conformidade aos requisitos(como correção, completude, consistência, e precisão);
    - Satisfaz os padrões, práticas, e convenções durante o processo de ciclo de vida;
    - Completar com sucesso cada atividade de ciclo de vida e satisfaz todos os critérios para inicializar as atividades de ciclo de vida seguintes.
- Validação: Provê evidências objetivas para quando o produto realiza alguma das ações:
    - Satisfaz os requisitos do sistema alocado para os produtos no final de cada atividade do ciclo de vida;
    - Resolve o problema correto (como modelos das leis físicas corretas, implementação da regra de negócios, e uso das suposições de sistema corretos);
    - Satisfaz o uso devido e necessidades do usuário no ambiente operacional

Ou seja, verificar significa garantir que um produto atente aos requisitos que foram especificados, validar significa que um produto realiza suas funções corretamente no ambiente na qual foi desenvolvido para ser usado. O propósito da Norma IEEE 1012-2012 é auxiliar os testadores com suporte na verificação e validação do software, sendo encontrado da seguinte forma em sua documentação:

- Estabelecer um framework comum dos processos, atividades, e atividades em suporte de V&V (*Verification and Validation*) do processo de ciclo de vida de sistemas, softwares, e hardwares.
- Definir as atividades de V&V, entradas e saídas necesssárias em cada processo do ciclo de vida.
- Identificar o mínimo de atividades de V&V correspondentes ao esquema de integridade aos quatro níveis de teste de software (Teste de Unidade, teste de Integração, Teste de Sistema, e Teste de Aceitação).
- Define o conteúdo do plano de V&V de software (SVVP, *Software Verification and Validation Plan*).

2.3.3.IEEE 829

A IEEE 829-2008 é a terceira versão da norma 829 para Documentação de Teste de Software (*829 Standard for Software and System Test Documentation*), criada inicialmente em 1983, tendo a segunda versão revisada em 1998.

A norma IEEE 829-2008 apoia todo o ciclo de vida de processo de teste de software, cobrindo áreas de aquisição, suporte, desenvolvimento, operação e manutenção. A norma é compatível com todos os modelos de ciclo de vida, mas que nem todos os modelos de ciclo de vida usam os processos desta norma. A norma também é um dos passos do ciclo de vida de software da IEEE como a IEEE 1012 (IEEE, 2012).

A norma 829-2008 se aplica a todos os sistemas baseados em software, e é aplicável a sistemas e softwares sendo desenvolvidos, adquiridos, operados, editados, e/ou reusados (como legados, modificados, e Itens não desenvolvidos). Ao se conduzir os processos de testes, é importante examinar o software em sua interação com as outras partes do sistema.

Esta norma identifica as considerações do sistema que os processos de testes e atividades endereçadas em determinados sistemas e softwares corretamente e outros atributos (completude, precisão, consistência, e testabilidade) e o resultado da documentação de testes aplicados.

O propósito da Norma IEEE 829-2008 é auxiliar os testadores na pradronização e documentação de teste, sendo encontrada a informação da seguinte forma em sua documentação:

- Estabelecer um framework para processos de testes, e atividades em suporte a todo processo de ciclo de vida do software, incluindo processos de aquisição, suporte, desenvolvimento, operação, e manutenção.
- Definir as atividades de teste, juntamente com as entradas e saídas necessárias.
- Identificar o mínimo de atividades de testes recomendadas que corresponde ao esquema de integridade dos quatro níveis de teste de software.
- Define o uso e conteúdo do Plano de Teste Mestre (*Master Test Plan*) e o Nível dos Planos de Teste (*Level Test Plan*)
- Define o uso e conteúdo da documentação relativa dos testes.

3. ESTADO DA ARTE

Como o presente trabalho trata da automação de testes de uma aplicação Flex, este tópico é referente a relatos na literatura de experiências similares, procurando aprender com os diferentes pontos de vistas dos outros autores, como os problemas relatados neste trabalho têm sido tratados em experiências similares.

No intuito de buscar por trabalhos similares que representem o estado da arte, foram consultadas as seguintes ferramentas de pesquisa:

Google (http://www.google.com), Google Scholar (http://scholar.google.com), Portal CAPES (http://www-periodicos-capes-gov-br.Ez46.periodicos.capes.gov.br/index.php?option=com_phome), e IEEE Explore (http://ieeexplore. Ieee.org), pela facilidade de acesso e relevância como fontes de pesquisa.

Nestas ferramentas de busca, foram utilizadas as seguintes palavras de busca: Software "Teste automatizado", "Flex" e algumas variações/complementos: Automação de testes de software, benefícios de testes automatizados, como automatizar testes com Flex, Diferenças entre Flex e HTML. Também foram utilizadas nas pesquisas as traduções na língua inglesa dos mesmos termos.

Como o volume de links encontrados poderia ser muito extenso, os seguintes critérios de inclusão foram utilizados para selecionar os resultados encontrados:

- Ferramentas de automação de Testes com suporte à plataforma Flex: pois uma ferramenta para automatizar testes que não suporta a plataforma Flex, não é relevante ao trabalho;
- Plug-ins de automação de teste para Flex: tendo mesmo motivo de procura como foi relatado no critério acima;
- Relatos de experiência de automação de teste com Flex: para saber como ferramentas trabalham nesta plataforma e ter uma melhor decisão sobre o software escolhido.

Além disso, para selecionar e avaliar os trabalhos relacionados, foram levantados os requisitos apresentados no **Error! Reference source not found.**. Estes requisitos foram identificados com base:

- Nas características de ferramentas de automação de testes identificadas na literatura;

- Nas necessidades para automação de testes coletadas por meio de entrevistas com analistas de testes da empresa desenvolvedora do software na qual o teste automatizado será executado;
- Em entrevista com o gerente de projetos da empresa desenvolvedora do software na qual o teste automatizado será executado.

Os requisitos coletados são apresentados no Quadro 4:

Quadro 3 - Requisitos da ferramenta de automação

Código	Requisito
REQ01	A ferramenta deve possuir alguma versão gratuita.
REQ02	A ferramenta deve suportar testes automatizados.
REQ03	A ferramenta deve estar atualizada e possuir uma comunidade ativa (possuir grupos e fóruns online, com postagens nos últimos 6 meses e última versão publicada nos últimos 6 meses).
REQ04	A ferramenta deve suportar testes automatizados de aplicações que utilizam a plataforma Flex.
REQ05	A ferramenta deve suportar as versões mais recentes dos navegadores: Firefox, Chrome, Internet Explorer, e Opera.
REQ06	A ferramenta deve suportar testes automatizados em Aplicações Web.
REQ07	A ferramenta deve possuir documentação abrangente (manual e guia de instalação).
REQ08	A ferramenta deve ser capaz de utilizar um repositório central (SVN) para que múltiplos usuários possam utilizar os scripts gerados.

Fonte: criado pelo autor

Aplicando-se as palavras de busca nas ferramentas de busca citadas e aplicando-se os critérios de seleção nos resultados encontrados, foram encontradas algumas ferramentas de automação de testes como mostra o Quadro 4:

Quadro 4 - Ferramentas de Automação de Testes e Referências

Ferramenta	Referência
Flex-ui-Selenium	https://code.google.com/p/Flex-ui-selenium/
FlexMonkey	https://www.gorillalogic.com/Flexmonkey
Badboy	http://www.badboy.com.au/
Testmaker	http://www.pushtotest.com/index.php
Sikuli	http://www.sikuli.org/
Bugbuster Test Record	https://bugbuster.com/

Fonte: criado pelo autor

Para decidir sobre qual das ferramentas citadas no Quadro 4 seria utilizada neste trabalho, foi necessário pesquisar sobre cada uma das ferramentas, tentar instalá-la e utilizá-la sobre a aplicação em Flex. Para cada uma das ferramentas, os requisitos levantados são avaliados.

3.1. Flex-ui-Selenium

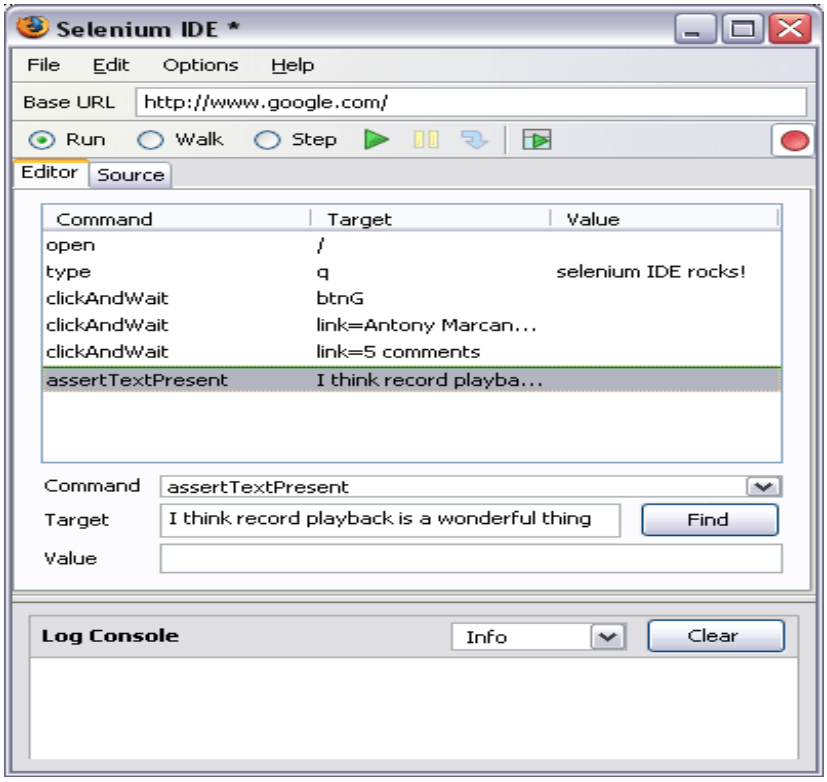
Extensão da ferramenta Selenium para a plataforma Flex, o Flex-ui-Selenium permite que o Selenium RC (*Remote Control*) possa interagir e testar aplicações em Flex.

Para se entender como funciona o Flex-ui-Selenium (também sendo referenciado como SeleniumFlex), é necessário entender a ferramenta Selenium sozinha. Selenium é um ambiente de desenvolvimento para scripts de testes automatizados, sendo usado uma extensão sua do Firefox, que permite gravar, editar, e depurar testes.

Seus scripts podem ser gerados em um padrão da própria ferramenta, ou em uma programação escolhida pelo usuário dentro das opções disponíveis, algumas delas são Java, C#, Perl, Ruby, e PHP.

A Figura 6 mostra a tela principal do Selenium IDE, informando URL de trabalho, comandos, e alvos da ação.

Figura 6 - Tela de ferramentas Selenium IDE



Fonte: Print Screen da ferramenta Selenium IDE.

A ferramenta foi referenciada por usuários do website Stackoverflow[9], porém, os tutoriais encontrados para a instalação do Selenium e, consequentemente, sua extensão para a plataforma Flex são muito confusos, mesmo seguindo os tutoriais, ainda não foi possível realizar testes com a ferramenta.

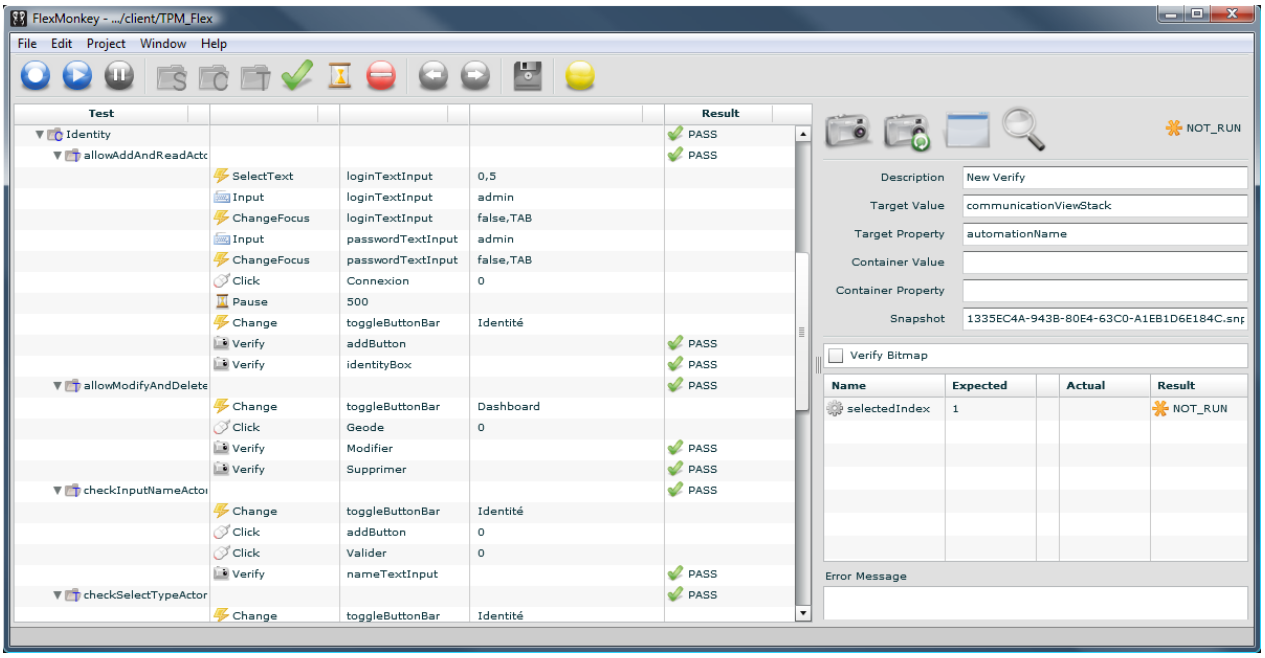
3.2. FlexMonkey

Ferramenta desenvolvida pela GorillaLogic, o FlexMonkey é um aplicativo Adobe AIR gratuito, que permite a criação de testes automatizados em plataformas AIR e Flex.

Foi a ferramenta que mais foi citada em experiências de usuários nas pesquisas feitas, sendo relatada como um dos melhores aplicativos para testes nestas plataformas (como no fórum de discussão do Stackoverflow).

A Figura 7 mostra a tela principal do FlexMonkey, com um teste detalhado, descrições do teste e o resultado de cada passo do teste.

Figura 7 - Tela da ferramenta FlexMonkey



Fonte: Print Screen da ferramenta FlexMonkey

Nos dados encontrados, a ferramenta se demonstrou completa, de fácil entendimento e utilização ao seguir os tutoriais da empresa. Infelizmente, a ferramenta foi descontinuada pela

[9]< http://stackoverflow.com/questions/72462/automated-testing-of-Flex-based-applications>

empresa em novembro de 2012, que decidiu focar no MonkeyTalk, uma ferramenta de testes para dispositivos móveis.

3.3. Badboy

O Badboy é uma ferramenta de automação de testes baseada na interface gráfica do usuário (*GUI - Graphics User Interface*), trabalhando com a movimentação do mouse e com um pequeno script.

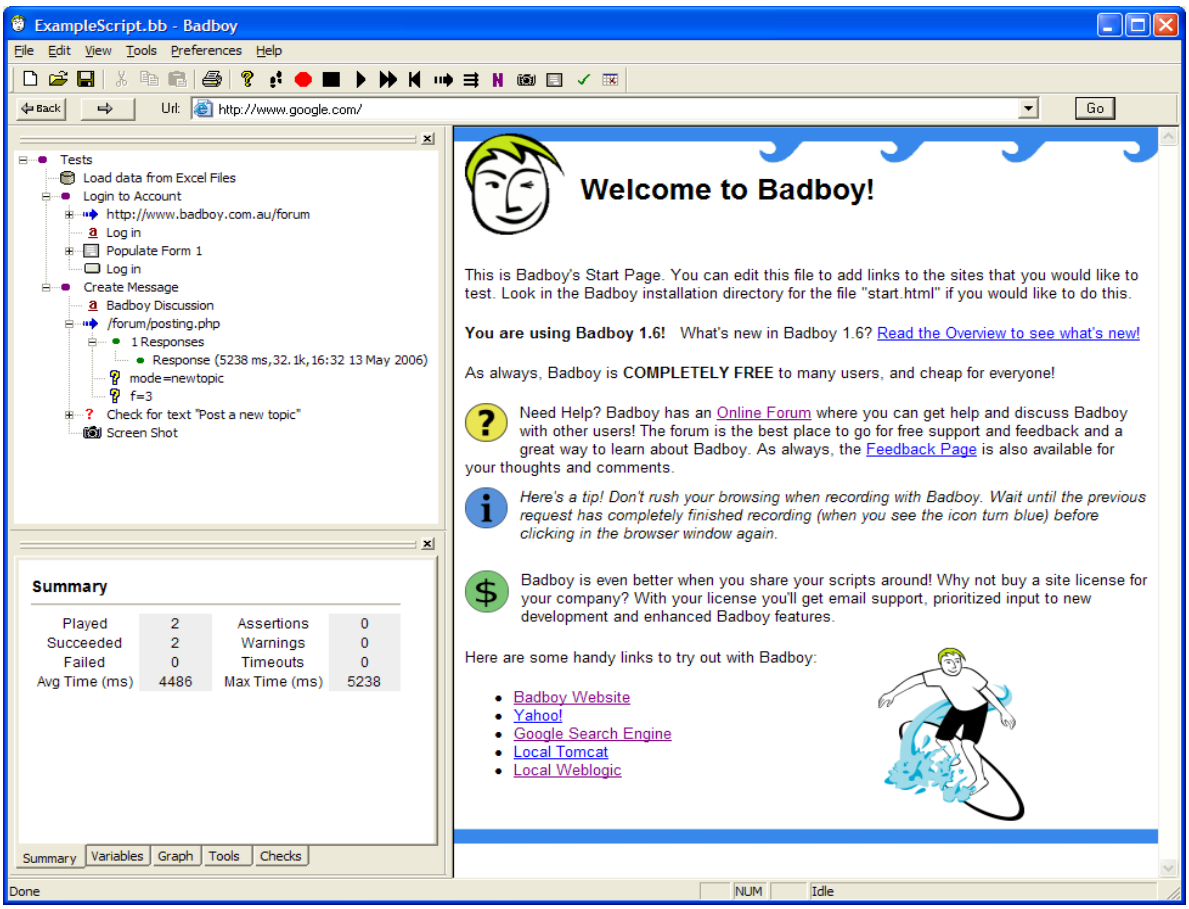
Os scripts utilizam o modo Record/Playback, estilo onde o usuário interage e suas ações são convertidas em script de teste pela aplicação, podendo ser executada automaticamente pela ferramenta.

Badboy possui dois modos de gravação de Script:

- Requisição (*Request*): a aplicação grava a requisição da página HTTP que são enviadas do browser ao server, incluíndo parâmetros e outras informações;
- Navegação (*Navigation*): a aplicação grava a navegação no browser, o elemento que foi selecionado. O Badboy vai procurar o elemento selecionado para interagir, no lugar de repetir as ações feitas.

A Figura 8 mostra a tela principal do Badboy, mostrando o sumário dos testes, a lista de testes, e a tela principal do programa quando é aberto.

Figura 8 - Tela da ferramenta Badboy



Fonte: Print Screen da ferramenta Badboy

Sendo uma ferramenta simples e de fácil interação, foram verificadas suas funcionalidades sobre a aplicação Flex, mas os testes realizados após a instalação do software, e pesquisas feitas em sua documentação, mostraram que o Badboy trabalha com aplicações HTTP, sem compatibilidade com aplicações Flex.

3.4. Testmaker

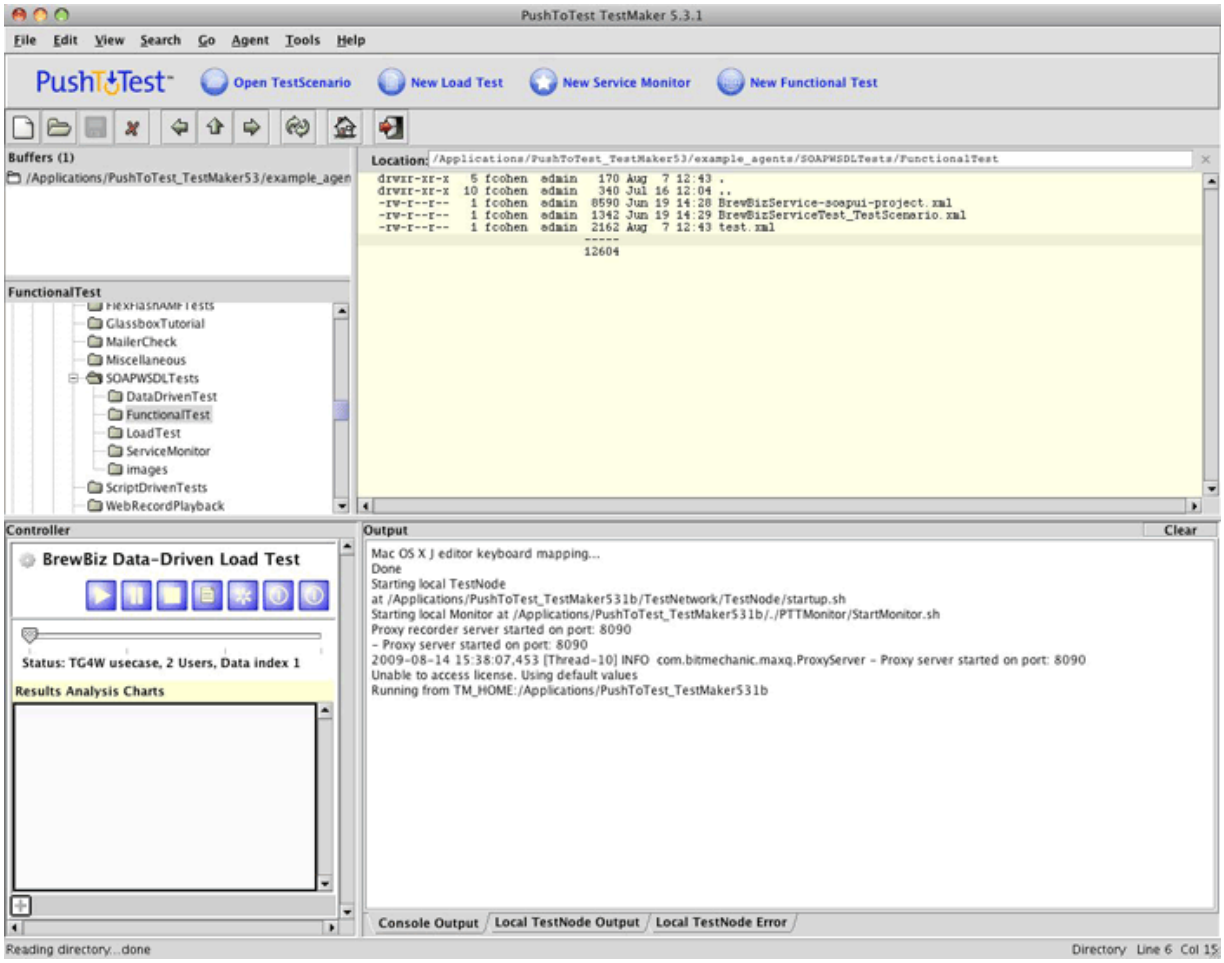
Testmaker é uma ferramenta gratuita que consegue tornar um script de teste em um teste funcional, um teste de carga, ou um teste de performance.

Desenvolvido pela PushToTest[10], o Testmaker pode gerar e rodar scripts de outras ferramentas: Selenium, Sahi, SoapUI. Também roda testes unitários nas linguagens Java, Ruby, Python, e PHP.

[10]<http://www.pushtotest.com/index.php>

A Figura 9 mostra a tela da ferramenta TestMaker após o início da realização de um teste, mostrando o Output e a organização de pastas.

Figura 9 - Tela de ferramentas Testmaker



Fonte: Print Screen daferramentaTestmaker

O software demonstra vários aspectos positivos, como sua capacidade de trabalhar em multi-linguagens e rodar scripts gerados por outras ferramentas de automação de testes. Seu problema maior reside na pouca documentação e tutoriais didáticos, tendo também o mesmo problema do Selenium-ui-Flex no momento de sua instalação. Também não foi possível encontrar informações suficientes para determinar se funciona sobre a aplicação Flex.

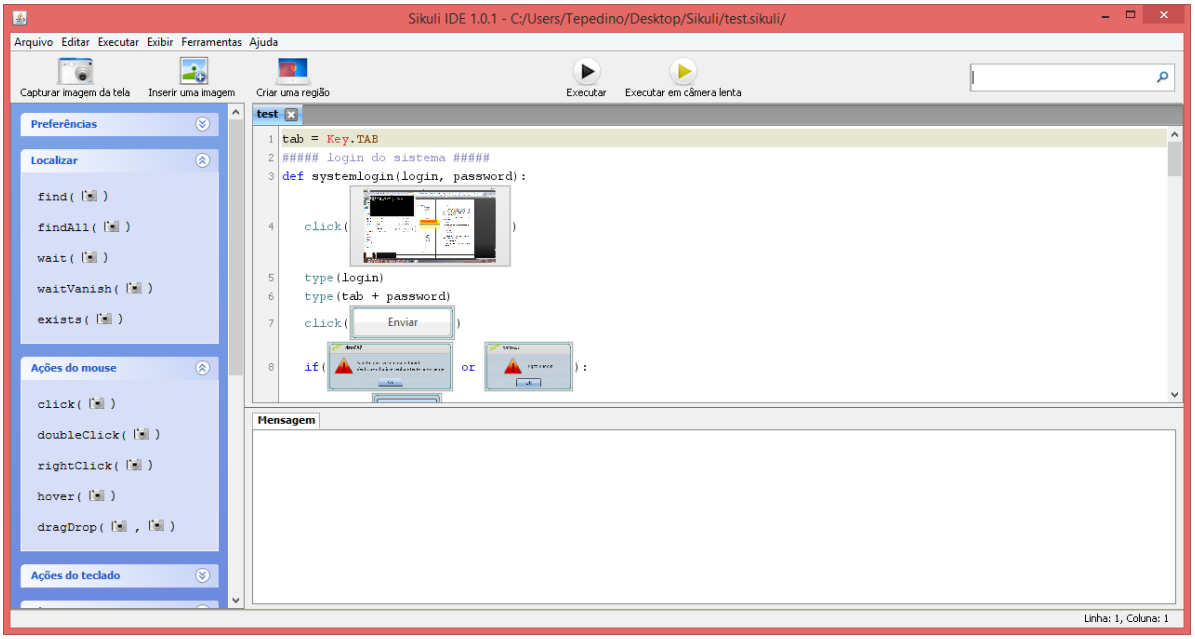
3.5. Sikuli

O Sikuli é uma ferramenta de automações de teste híbrida, na qual trabalha tanto com GUI, como com scripts de usuário. A ferramenta interage com a tela a partir da captura de imagens, ou regiões, definidas no script.

A ferramenta foi criada pelo MIT[11] (*Massachusetts Instituteof Technology*), com intuito de testar aplicações sobre WWW. Funciona a partir da linguagem Jython, uma implementação da linguagem Python para a plataforma Java.

A Figura 10 mostra a tela principal do Sikuli, com um script de testes pronto para ser executado.

Figura 10 - Tela de ferramentas Sikuli



Fonte: Print Screen da ferramenta Sikuli

A ferramenta é de fácil manuseio e interação, com atalhos e imagens intuitivas que auxiliam em sua utilização com o mínimo de tutoriais necessários. Sua maior dificuldade é na formatação de scripts mais complexos, necessitando conhecimento tanto de Python quanto de Java.

3.6. Bugbuster Test Record

O Bugbuster Test Record é uma extensão feita para o Google Chrome, permitindo a criação de testes a partir de cliques feitos na janela do browser.

Criado pela Bugbuster SA[12], sua premissa é "realizar testes que nenhum código precisa ser digitado", pois o bugbuster que cria o código. É similar ao Badboy no modo de captura “Navegação".

[11] <http://web.mit.edu/>
[12] <https://bugbuster.com/>

Possui uma aparência simples e de fácil utilização, com vídeo tutoriais didáticos e implementações diferenciais, como "criação de variáveis aleatórias". Infelizmente não foi encontrada informação sobre seu suporte à plataforma Flex, é limitado o uso em browsers Google Chrome e Safari, e uma vez gravado, este só pode ser editado mexendo no código gerado.

3.7. Discussão

Tendo por base os requisitos definidos anteriormente, as ferramentas foram avaliadas seguindo os passos:

- Acessar o site responsável pela ferramenta,
- Procurar e ler seu guia de instalação,
- Instalar a ferramenta,
- Elaborar um caso de testes simples para verificação de requisitos.

Dessa forma, a experiência obtida em cada ferramenta, foi possível analisar cada uma delas em relação ao atendimento dos requisitos, como mostra o Quadro 5.

Quadro 5 - Quadro Comparativo

Ferramenta	REQ01	REQ02	REQ03	REQ04	REQ05	REQ06	REQ07	REQ08
Flex-ui-Selenium	✔	✔	✔	✔	✔	✔	✖	✔
FlexMonkey	✔	✔	✖	✔	✔	✔	✔	✔
Badboy	✔	✔	✔	✖	✔	✔	✔	✔
Testmaker	✔	✔	✔	?	✖	✔	✖	✔
Sikuli	✔	✔	✔	✔	✔	✔	✔	✔
Bugbuster Test Record	✔	✔	✔	?	✖	✔	✔	✖

Fonte: criado pelo autor

✔ - Atende completamente

✖ - Atende parcialmente ou não atende

? - Não foi possível determinar

Conforme observado no Quadro 5, o Flex-ui-Selenium não cumpre o REQ07 por não possuir uma documentação abrangente, o que dificultou pesquisas e tentativas de utilização da ferramenta.

O FlexMonkey não cumpre o REQ03 pelo que foi descrito anteriormente: A ferramenta foi descontinuada, e não receberá mais atualizações.

A ferramenta Badboy falha no REQ04 pois não oferece suporte à plataforma Flex: A ferramenta espera um retorno do comando para dar continuidade as ações, mas o Flex não responde este comando.

Não foi possível descobrir se o Testmaker suporta a Plataforma Flex, pois não possuía uma documentação para poder até mesmo instalar a ferramenta com facilidade, e o TestMaker funciona apenas sobre Firefox.

A ferramenta Bugbuster foi possível de ser instalada, mas não foi possível determinar se a ferramenta suporta testes de aplicações na linguagem Flex, e a ferramenta apenas funciona na plataforma Google Chrome.

Dessa forma, pode-se notar que a ferramenta Sikuli foi a única ferramenta encontrada que atendeu satisfatoriamente aos 7 requisitos levantados. Assim, essa foi a ferramenta selecionada para o estudo de caso que é apresentado no capítulo 4.

4. ESTUDO DE CASO

Neste capítulo, é apresentado o estudo de caso de automação de testes para plataforma Flex. É apresentada uma breve descrição da organização desenvolvedora do sistema que será utilizado no estudo de caso, juntamente com os instrumentos de coleta de dados e, por fim, a aplicação do estudo de caso. Serão, assim, executados casos de teste manuais e automatizados, sendo que os resultados observados são descritos no capítulo seguinte.

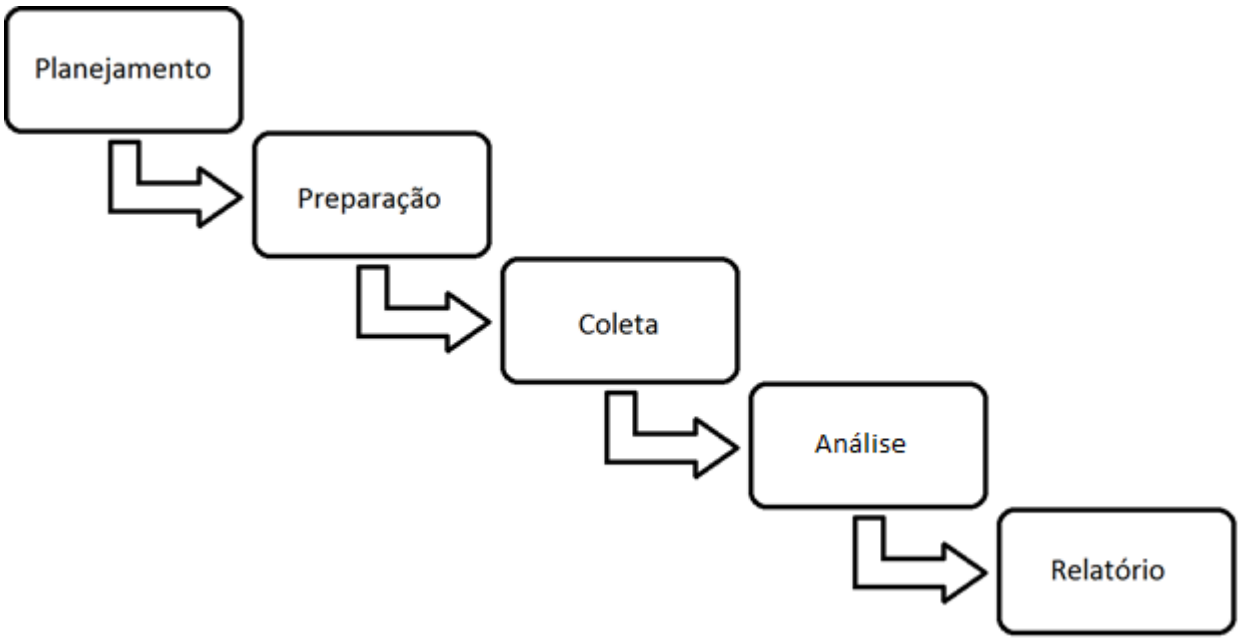
4.1. Definição do Estudo de Caso

Estudos de caso têm sido escolhidos, de forma recorrente, como alternativa de aplicação para pesquisas no meio acadêmico. Segundo Runeson (2009), pode-se definir que um estudo de caso é o estudo focado de um fenômeno em seu contexto, especialmente quando os limites entre o fenômeno e o contexto não são bem definidos.

Ainda segundo Runeson (2009), estudos de caso são usados em diversas áreas com o objetivo de aumentar o conhecimento sobre indivíduos, grupos e organizações. Runeson (2008) informa também que é compreensível aplicar estudos de caso às áreas relacionadas à engenharia de software tais como desenvolvimento, operação, e manutenção de software.

É necessário planejar o estudo de caso, com uma base bem definida, para se analisar as unidades, os métodos e técnicas para coletas de dados e para se definir o controle e os resultados da pesquisa. Assim, na Figura 11, são apresentadas as principais etapas utilizadas na realização deste estudo de caso, com base em Runeson (2008):

Figura 11 - Etapas de Estudo de Caso



Fonte: elaborada pelo autor, com base em Runeson (2009).

- **Planejamento**: objetivos são definidos e o estudo de caso é planejado (descrito neste capítulo);
- **Preparação**: procedimentos e protocolos para coleta de dados são definidos;
- **Coleta**: execução com os dados coletados no estudo de caso;
- **Análise**: onde os dados coletados são analisados e extraídas as conclusões (descrito no próximo capítulo);
- **Relatório**: onde os dados analisados são reportados.

Estes passos são praticamente os mesmos para qualquer tipo de estudo empírico, como cita Runeson (2008).

4.1.1.Planejamento do Estudo de Caso

De acordo com Runeson e Höst (2009), é crucial para o sucesso de um estudo de caso possuir um bom planejamento. Algumas questões devem ser planejadas, como que métodos serão utilizados para a coleta de dados. Na próxima seção, será apresentada a unidade de estudo de caso:

4.1.1.1. Unidade de Análise do Estudo de Caso

A Specto Painéis Eletrônicos Tecnologia LTDA [13] é uma empresa que iniciou suas atividades no início da década de 90, desenvolvendo e fornecendo produtos com aplicabilidade, conectividade e qualidade por meio de tecnologias que criam valor para seus clientes.

A organização possui três divisões que desenvolvem três linhas de produtos: divisão de gestão de atendimento (gestão de atendimento), divisão de informação ambiental (totens com informações ambientais), e divisão de prédios inteligentes (controle de acesso).

Para que os produtos sejam desenvolvidos, a empresa conta com um corpo de aproximadamente 80 funcionários, classificando-se como uma empresa de tecnologia de médio porte. Seus produtos, quando desenvolvidos, são suportados por um grande conjunto de tecnologias, ferramentas e frameworks.

No Quadro 6 são apresentadas as tecnologias usadas no processo de desenvolvimento do sistema utilizado no estudo de caso:

Quadro 6 - Tecnologias Utilizadas

Controle de versões	TortoiseSVN (Windows Explorer)
Controle de modificações	Channel (Formulário de Solicitações de Mudança no Projeto)
Construção	Apache Flex Emberjs Enterprise Architect Eclipse (Java Web) Notepad++ JSP Javascript jQuery Java SE Development Kit (JDK) PostgreSQL Servlet

Fonte: Specto, 2015.

A seguir a descrição das tecnologias utilizadas:

[13]http://www.specto. Com. Br

TortoiseSVN (Windows Explorer)[14]- É usado para controle de versões para Microsoft Windows com código aberto.

Channel[15]- A Plataforma Channel é um software corporativo que promove o alinhamento das operações e projetos de uma instituição ao seu planejamento estratégico. É utilizada para o gerenciamento dos projetos de desenvolvimento e manutenção de software da empresa.

Apache Flex - É um kit de desenvolvimento de software (*Software Development Kit*, ou SDK) para o desenvolvimento de plataformas de aplicação baseadas no Adobe Flash. Anteriormente era conhecida como **Adobe Flex**.

Emberjs[16]- É um framework open-source para aplicações JavaScript Web baseado no padrão de arquitetura de software *model-view-controller*. O framework permite que desenvolvedores criem aplicações com uma única web page.

Enterprise Architect[17]- É uma ferramenta multi-usuário, gráfica, projetada para ajudar as suas equipes a construir sistemas robustos e de fácil manutenção. É utilizada para a modelagem UML dos sistemas desenvolvidos além da rastreabilidade.

Eclipse [18] - É um ambiente de desenvolvimento Java, porém suporta várias outras linguagens com o uso de plugins, segue o modelo open-source de desenvolvimento de software.

Notepad++[19]- Editor de texto versátil e com suporte a várias linguagens de programação.

JSP[20] - *JavaServer Pages*(JSP) é uma tecnologia que auxilia os desenvolvedores de software a criarem páginas web geradas dinamicamente baseadas em HTML, XML ou outros tipos de documentos.

Javascript jQuery[21]- JavaScript é uma linguagem de programação interpretada. jQuery é uma biblioteca JavaScript desenvolvida para simplificar os scripts que interagem com o HTML.

Java SE Development Kit (JDK)[22]- É um Kit de Desenvolvimento Java, consiste em um conjunto de ferramentas que permitem criar aplicações para plataforma Java.

---

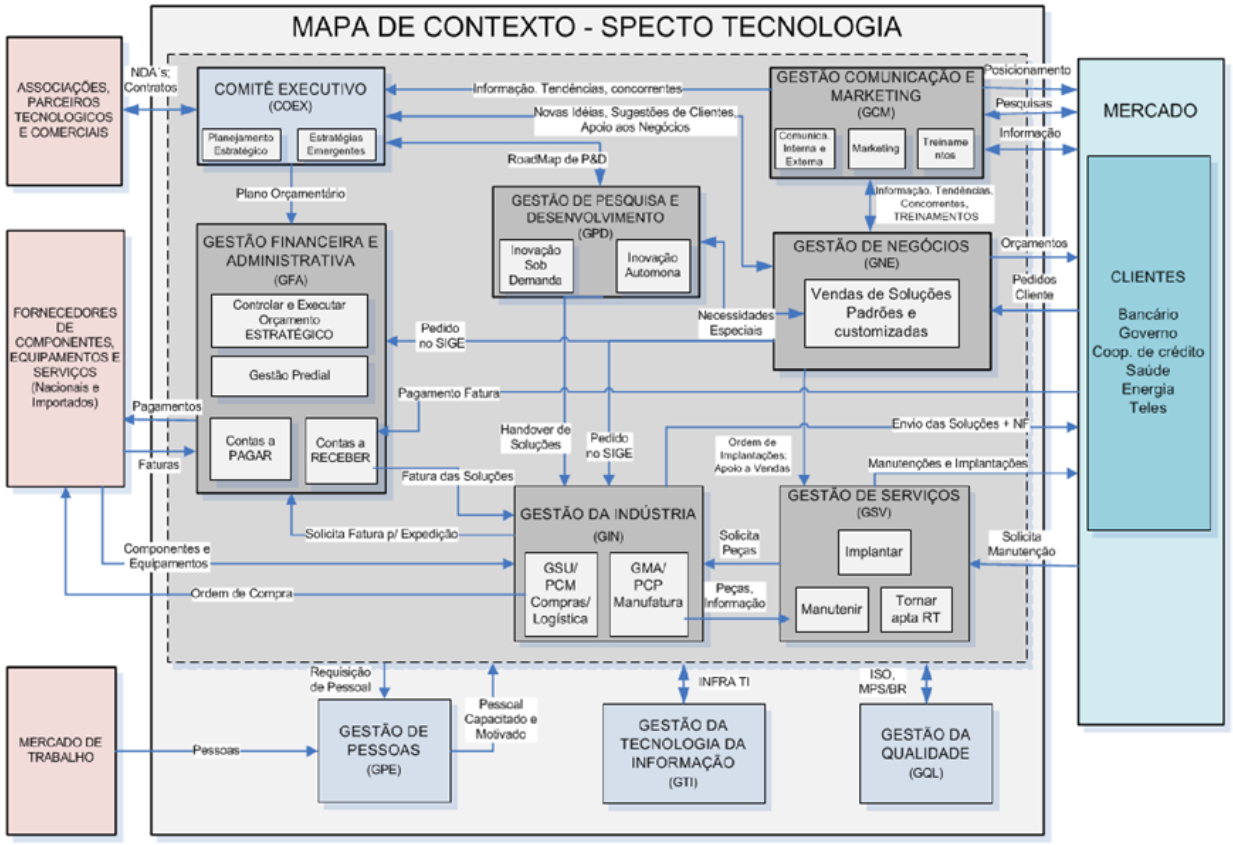
[14]<http://tortoisesvn.net/>
[15]<Intranet/channel>
[16]<http://emberjs.com/>
[17]<http://www.sparxsystems. Com/>
[18]<https://www.eclipse.org>
[19]<https://notepad-plus-plus.org/>
[20]<http://www.oracle. Com/technetwork/java/javaee/jsp/index.html>
[21]<https://jquery. Com/>
[22]<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

PostgreSQL[23] - Conhecido também como **Postgres**, é um Sistema de Gerenciamento de Banco de Dados (SGBD).

Servlet[24]- é uma tecnologia Java usada para estender as funcionalidades de um servidor e responder a requisições HTTP.

A Figura 12 mostra as divisões existentes na empresa Specto, juntamente com a interação que estas divisões possuem com as demais estruturas da empresa:

Figura 12 - Mapa de Contexto



Fonte: Specto, 2015

Para esse trabalho será apresentada a divisão de gestão de informação ambiental (DGI) da empresa Specto, onde é realizado o presente estudo de caso. A DGI tem como objetivo monitorar e gerenciar ambiente e Data Centers, permitindo por exemplo, alertar o usuário que um ambiente está com muita fumaça, ou que um dispositivo de monitoramento foi desconectado.

A divisão de gestão de informação ambiental atualmente é composta por 4 desenvolvedores, 1 gerente de projetos, e 2 membros da garantia da qualidade de sistema. Essa

[23]<http://www.postgresql.org/>
[24]<http://www.oracle. Com/technetwork/java/index-jsp-135475. Html>

equipe desenvolve o produto DataFaz[25] com três soluções: DataFaz Integração, DataFazUnity, que é uma versão mais compacta e simplificada do DataFaz Integração, e DataFaz Gestão de Ativos, que além de ter as funcionalidades do DataFaz Integração, possui um módulo de gestão de ativos dos Data Centers.

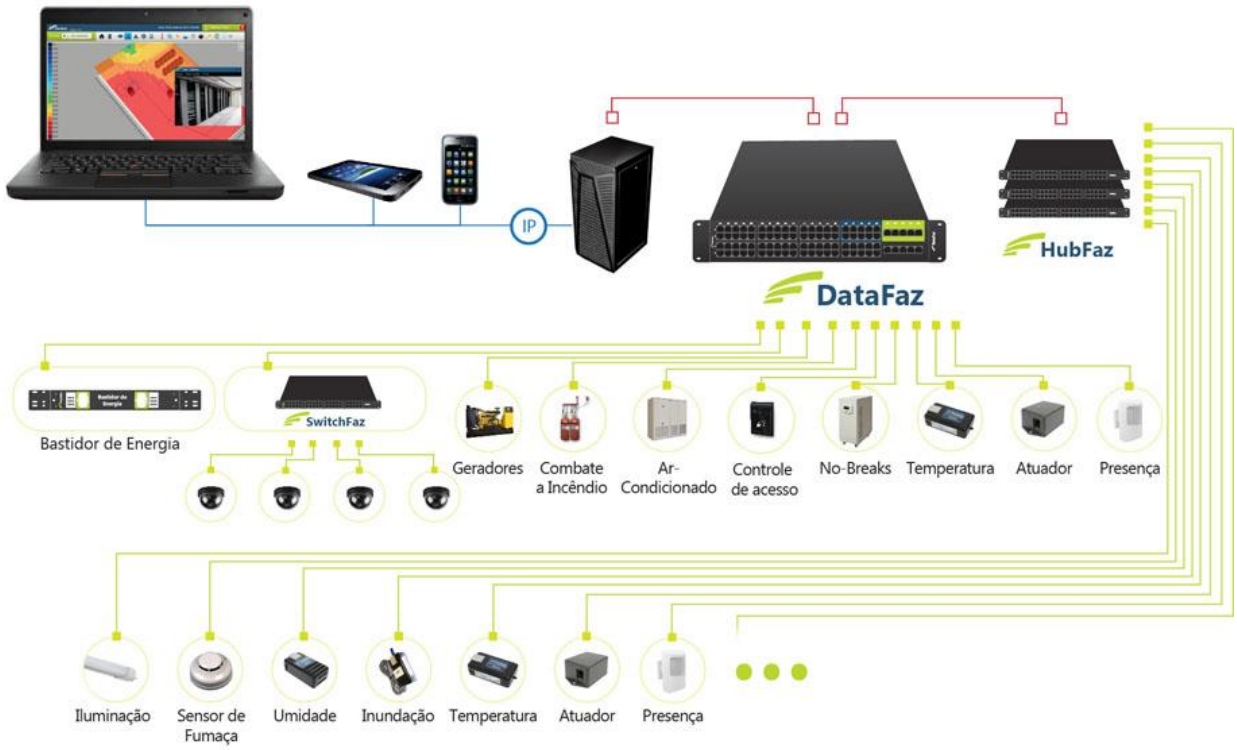
Neste estudo de caso serão envolvidos os membros responsáveis pelo desenvolvimento do Datafaz, e os membros da garantia de qualidade de produto: 3 analistas de testes e 2 testadores.

Sistema utilizado no Estudo de Caso

O sistema utilizado neste estudo de caso é o DataFaz Unity. O sistema DataFaz Unity é um gerenciador de infraestrutura de Data Centers, monitorando parâmetros físicos de ambientes de missão crítica (ambientes onde a perda de dados pode causar danos graves, financeiros e sociais) e do Data Center, realizando monitorando vários tipos, como umidade, temperatura, energia, e controle de acesso. Sua supervisão pode ser realizada pela Web, tablets ou celulares.

A Figura 13 mostra a arquitetura do sistema:

Figura 13 - Website DataFaz



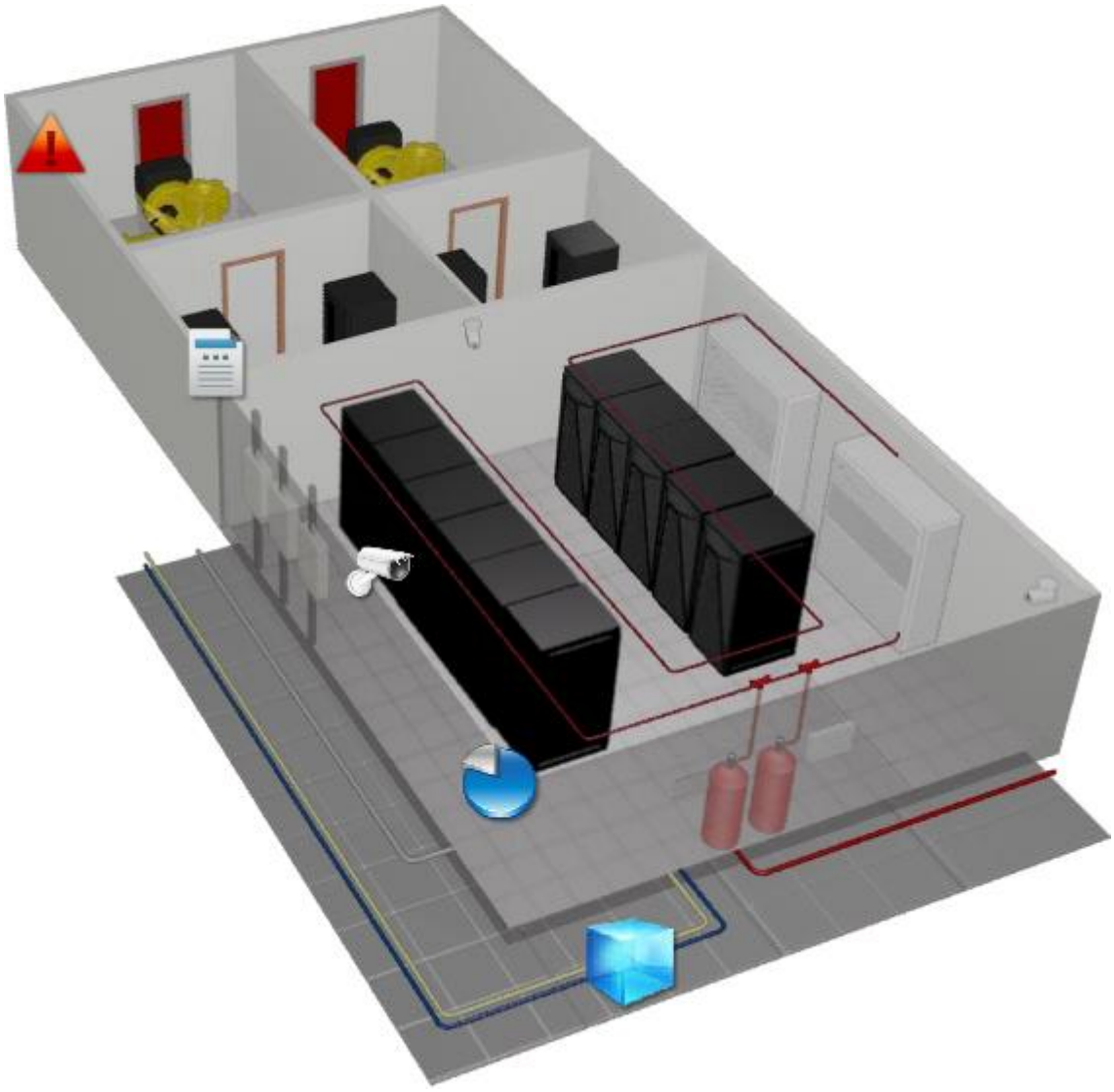
Fonte: Imagem do website da empresa (DataFaz, 2015)

[25]<http://www. Fazion. Com. Br/datafaz/?pg=inicial>

As três soluções informadas anteriormente têm como objetivo analisar e garantir que o ambiente de trabalho está sob condições estabelecidas nos sensores cadastrados em seu sistema.

A Figura 14 mostra o gerenciamento de um ambiente onde dispositivos e softwares de monitoramento estão relacionados ao local:

Figura 14 - Gerenciamento de Ambiente



Fonte: Imagem doWebsite da empresa(DataFaz, 2015)

O produto DataFaz Unity é uma versão mais compactada do DataFaz Integração, possuindo menos operações para realizar seu monitoramento. O DataFaz Unity está em produção atualmente em um cliente, mas suas outras versões já foram implantadas em quinze clientes.

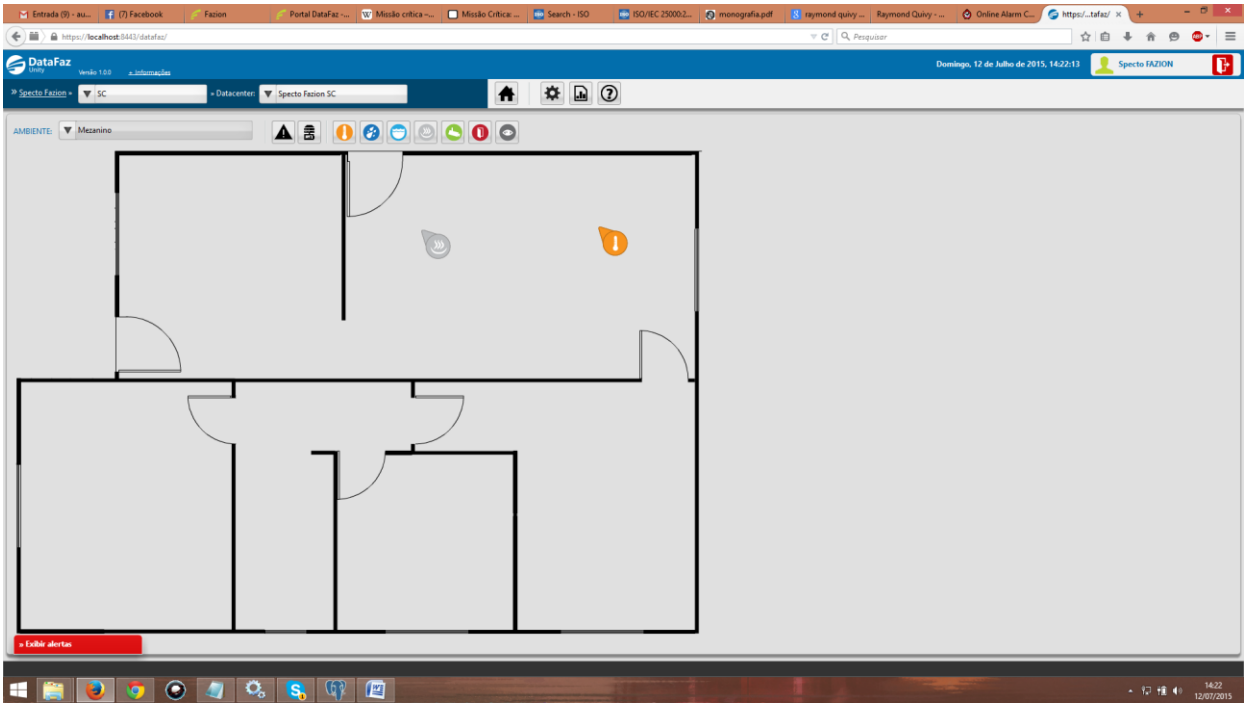
Dentre as tecnologias já apresentadas, as tecnologias empregadas especificamente para o desenvolvimento do sistema DataFaz Unity são: JSP, Servlet, Javascript jQuery, e a IDE utilizada é o Eclipse.

O sistema possui as seguintes principais funcionalidades:

- Cadastrar Usuário;
- Cadastrar Ambiente (Data Center, componentes 3D);
- Cadastro de Dispositivos (sensor de fumaça, sensor de temperatura, racks);
- Gerar relatório de acesso, monitoramento e cadastro;
- Visualizar ambientes via câmeras cadastradas;
- Edição de usuários do Sistema;
- Edição de dispositivos.

A Figura 15 - Tela principal DataFaz Unity ilustra a tela principal da aplicação, após o login do sistema:

Figura 15 - Tela principal DataFaz Unity



Fonte: Print Screen da ferramenta DataFaz Unity (SPECTO, 2015)

4.1.2. Coleta de Dados

A coleta de dados corresponde ao conjunto das operações, através das quais o modelo de análise é submetido ao teste dos fatos e confrontado com dados observáveis (Quivy et all, 1992, p. 157). O princípio da coleta de dados vem da utilização de várias fontes de evidência, criação de um banco de dados, de forma a encadear evidências de forma a buscar respostas para a pergunta de pesquisa deste trabalho: "Seria possível automatizar testes para um software Flex de forma a reduzir o esforço, tempo e retrabalho, tomando por base uma ferramenta já existente para modelagem e automatização de testes? ".

Para o presente estudo de caso, são coletados os seguintes dados:

- Quantidade de pontos de caso de uso;
- Quantidade de erros encontrados na execução dos testes automatizados;
- Avaliação pessoal dos resultados observados pelos envolvidos na área de testes;
- Quantidade de esforço realizado para a elaboração dos casos de testes automatizados;

O Quadro 7 o mostra como os dados serão extraídos durante a execução, para que seja criado um banco de dados com as evidências sobre o estudo de caso.

Quadro 7 - Método de coleta de dados

Dado Coletado	Procedimento de Coleta	Origem
Pontos de caso de uso	Contagem de caso de uso	Enterprise Architect
Quantidade de erros encontrados	Executar testes manuais e automatizados	Relatório de Teste manual e relatório de teste com ferramenta de automação de teste.
Resultados observados da automação	Aplicar entrevistas com os membros da equipe de desenvolvimento	Questionário para análise dos resultados observados
Tempo médio de execução dos casos de teste manuais	Verificar tempo utilizado para executar testes manuais	Channel e realização cronometrada do teste.
Esforço de desenvolvimento e execução dos testes manuais	Desenvolver e executar os testes manuais	Channel.
Tempo médio de execução dos casos de teste automatizados	Verificar tempo utilizado para executar testes automatizados	Channel e log do teste automatizado.
Esforço de desenvolvimento e execução dos testes automatizados	Desenvolver e executar os testes automatizados	Channel.

Fonte: elaborada pelo autor.

O dados mostrados no quadro acima foram coletados pela mesma pessoa, realizando os testes automatizados e manuais, na mesma máquina.

4.1.3.Início da realização do Estudo de Caso

O Quadro 8 contém os casos de testes que são utilizados neste trabalho, identificando-os, descrevendo-os, e informando a qual caso de uso cada caso de teste está relacionado. Estes casos de teste já eram executados manualmente e serão automatizados no contexto deste estudo de caso.

Quadro 8 - Descrição de Caso de Teste

Identificação	Nome Caso de Teste	Caso de Uso Relacionado	Descrição
TST-001.01	Login Padrão	US-001	Permite acesso padrão definido pelo sistema no primeiro acesso.
TST-002.01	Cadastrar Usuário	US-002	Essa parte do sistema tem como objetivo cadastrar usuários do sistema. Permite cadastrar usuário do tipo Regular e Master.
TST-002.02	Editar Usuário	US-002	Editar usuário já cadastrado, alterando suas permissões.
TST-003.01	Cadastrar Região	US-003	Cadastrar uma região no sistema para poder adicionar um Data Center a ser monitorado.
TST-004.01	Cadastrar Plataforma	US-004	Cadastrar uma plataforma no sistema para poder adicionar um tipo de dispositivo.
TST-005.01	Cadastrar Dispositivo	US-005	Cadastrar dispositivo no sistema para monitorar um ambiente de um Data Center.
TST -006.01	Download Relatório	US-006	Permite baixar relatórios do sistema à máquina.
TST-007.01	Enviar Comando	US-007	Permite enviar um comando manualmente ao dispositivo, como realizar envio de

			mensagens para garantir sua conexão, ou ler algum parâmetro do ambiente monitorado
TST-008.01	Cadastrar Alerta	US-008	Cadastrar um alerta para monitorar os ambientes cadastrados conforme a necessidade.
TST-008.02	Adiar Alerta	US-008	Adiar alerta que foi enviado para poder analisá-lo em outro momento.

Fonte: elaborada pelo autor.

Como primeiro passo na realização deste estudo de caso, os Casos de Teste elencados são documentados detalhadamente de forma a possibilitar a futura automação dos testes. O Quadro 9 - TST-001.01 descreve um exemplo detalhado de um caso de teste, seguindo a documentação alinhada às normas IEEE 829 e ISO/IEC 29119. Os demais Casos de Teste são apresentados no apêndice I.

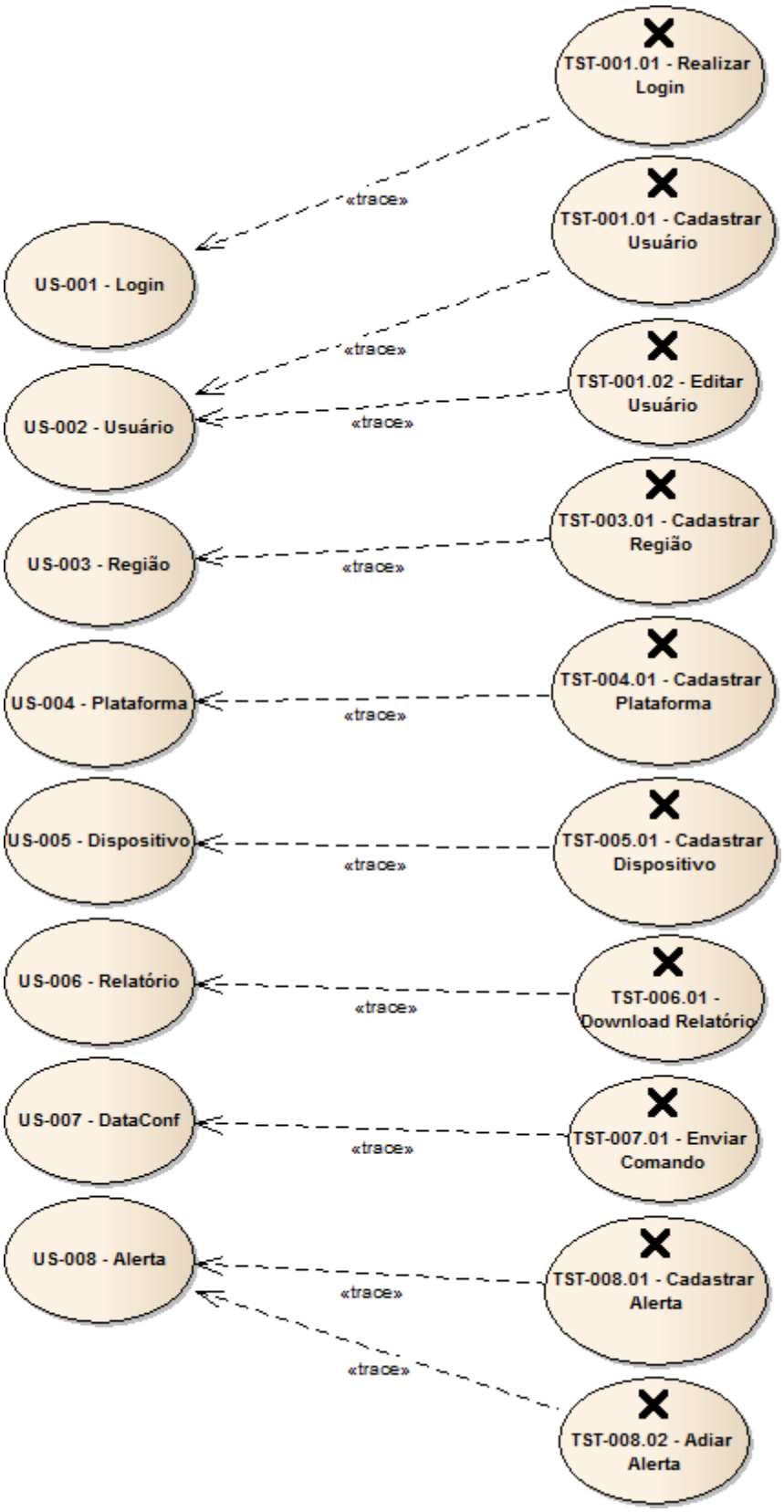
Quadro 9 - TST-001.01

Identificador	TST-001.01
Item de teste	Login Padrão
Especificações de Entrada	1. Usuário: fazion; 2. Senha de acesso: 22; 3. Clicar no botão "Ok".
Procedimentos	1. Inserir o dado "Usuário" na área "Nome de Usuário"; 2. Inserir o dado "Senha de acesso" na área "Senha"; 3. Selecionar o botão "Enviar".
Especificações de Saída	Após o procedimento ser realizado, a tela deve ser alterada, apresentando a tela principal da aplicação.
Ambiente necessário	DataFaz Unity, banco de dados.
Exigências especiais	Não aplicável.
Interdependências	Fazer uso do caso de teste específico e PST - Proposta Técnica.

fonte: elaborado pelo autor

Para manter a rastreabilidade, o diagrama mostrado pela Figura 16 apresenta uma relação entre os casos de uso com seus respectivos casos de teste. A nomenclatura dos Casos de Uso, apesar de não seguir as melhores práticas recomendadas, foi mantida desta forma pois sua alteração acarretaria em diversas alterações em outros artefatos do processo de desenvolvimento, o que está fora do escopo deste trabalho.

Figura 16 - Diagrama de Rastreabilidade



Fonte: elaborado pelo autor

Com os casos de testes detalhados da forma descrita no Quadro 9, o próximo passo será automatizar os testes, detalhar e analisar os resultados do estudo de caso, apresentados no Capítulo 5.

5. EXECUÇÃO E AVALIAÇÃO DO ESTUDO DE CASO

Este capítulo apresenta a execução do estudo de caso, relatando a experiência da aplicação da automação de testes realizada sobre a plataforma Flex utilizando a ferramenta Sikuli. Assim, o capítulo detalha mais sobre a ferramenta escolhida, explicando conceitos básicos, seu funcionamento, que linguagens suporta, e outras informações. Também é explicada a relação do caso de teste descrito no capítulo anterior com o código automatizado em detalhes, passo a passo. Por fim, os resultados observados no estudo de caso, com base nos dados coletados, são apresentados.

5.1. Execução do Estudo de Caso

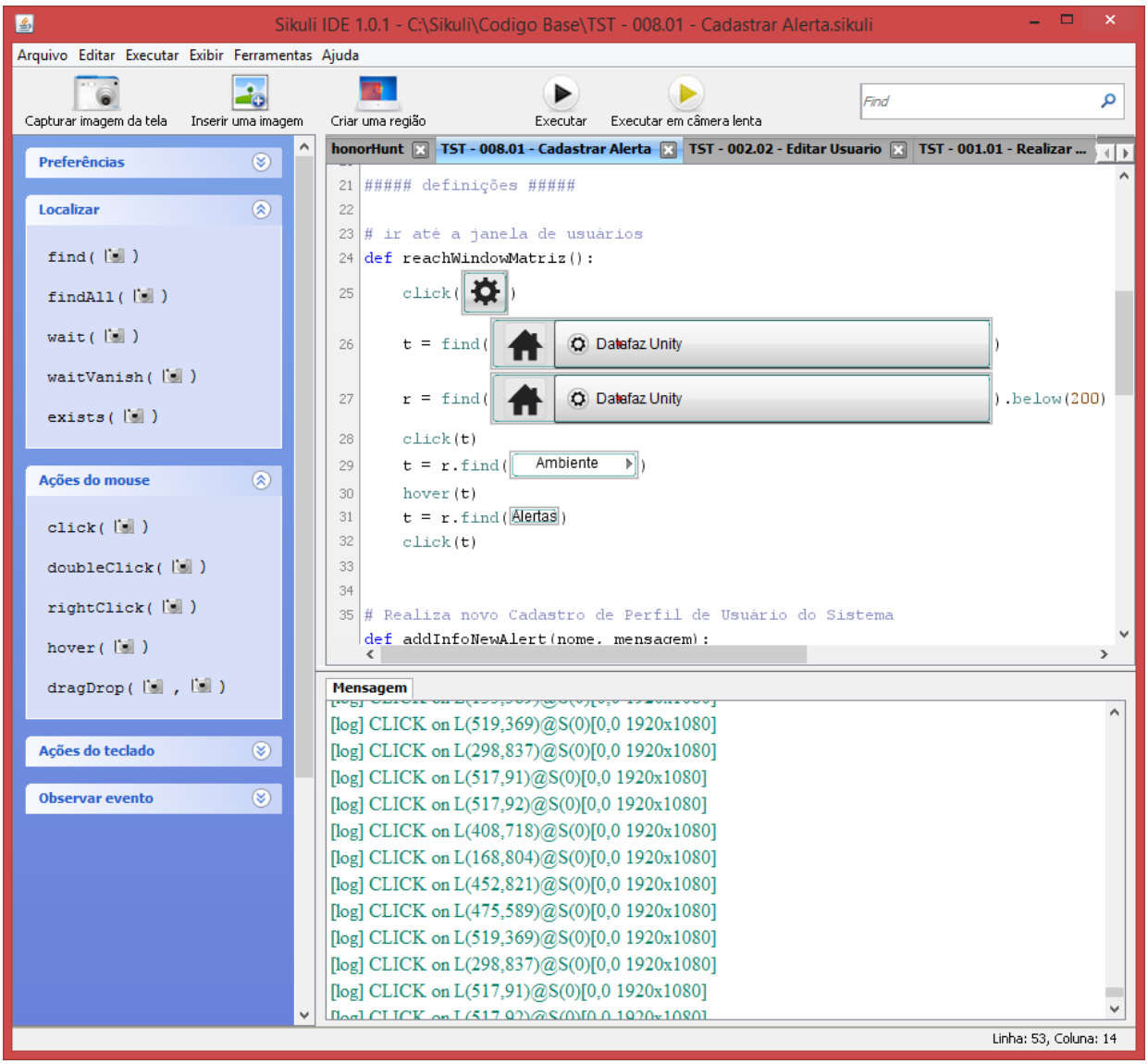
O estudo de caso foi executado no período de 5 de Maio de 2015 até 10 de Setembro de 2015, após toda a análise de qual ferramenta de automação seria utilizada, conforme estudo mostrado no Capítulo 3.

A ferramenta Sikuli ("Olho de Deus"[26], de acordo com o dialeto Indiano Huicholo. Significa "o poder para ver e entender coisas desconhecidas"[27]), conforme já apresentado,é uma ferramenta de automação de testes de GUI que mescla Scripts com imagens, que são procuradas quando o programa está rodando.

O Sikuli possui uma IDE própria, como mostra a Figura 17, que suporta linguagens Python e Ruby em uma plataforma Java: Jython e JRuby respectivamente, mas pode ser utilizado como um script de outras linguagens, como por exemplo Java, para utilização de outras IDEs, como o Eclipse e NetBeans. A IDE própria facilita o trabalho para capturar e adaptar imagens às necessidades de cada automação. Entretanto, sua utilização em outra IDE é mais complicada de se utilizar, porém pode ser integrada com outras linguagens e ferramentas.

[26]<http://groups. Csail.mit. Edu/uid/projects/sikuli/sikuli-uist2009.pdf>
[27]<https://en.wikipedia.org/wiki/God%27s_eye>

Figura 17 - Sikuli IDE



Fonte: Print Screen da ferramenta Sikuli.

Segundo MILLER et al (2009), a ideia de utilizar imagens para auxiliar testes GUI veio da própria interação humana. Por exemplo: ao perguntar "Diga-me mais sobre isso" apontando para um quadro, a pessoa na qual a pergunta foi direcionada entenderá que quem perguntou quer saber mais informações sobre o quadro, ou então "Coloque isso", apontando a um vaso, "naquela mesa" apontando uma mesa específica no meio de outras.

MILLER et al (2009) detalham que o Sikuli adota um método híbrido de busca, que usa um modelo de comparação que procura por pequenos padrões, cores, e traços de forma invariantes para definir um grau de similaridade aceitável às imagens na tela em comparação com a imagem sendo utilizada.

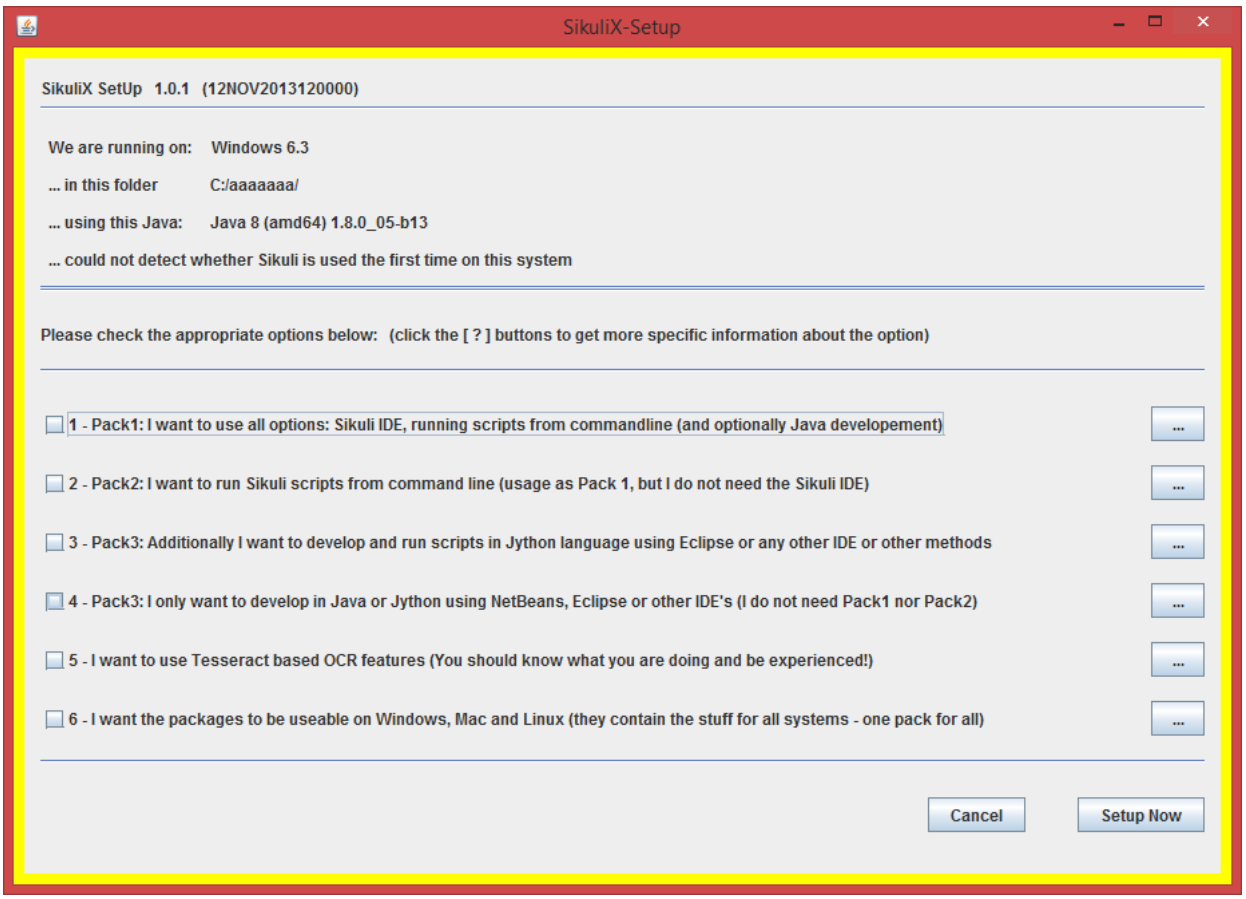
A imagem é utilizada como um parâmetro, similar a uma constante declarada em qualquer linguagem de programação, onde o usuário pode definir o nome da imagem, o grau de similaridade que procura na tela, e aonde irá selecionar a tecla quando necessário, sendo possível selecionar até mesmo outras áreas a partir daquela imagem. Possui também um modo de pré-visualização para comparação de imagem com a tela atual, para analisar o grau de similaridade.

Como um parâmetro, é necessário possuir a imagem em um repositório. A IDE do Sikuli cria uma pasta para cada projeto, com um nome padronizado em "NomeProjeto.sikuli", onde ficam armazenadas as imagens que são capturadas pelo projeto via IDE, o script "NomeProjeto.py" que é o código do projeto, e um HTML "NomeProjeto.html" para poder compartilhar o código com mais facilidade pela Web, caso a imagem não se encontre na pasta, o programa.

A ferramenta funciona como qualquer IDE, já que o código é escrito e, ao selecionar "Executar", o programa executa e irá parar apenas se acontecer algum erro ou ao final da execução. Erros próprios do Sikuli são, na maioria das vezes, relacionados à imagem. Um erro comum é o "FindFailed", que acontece quando o programa não encontra a imagem na tela.

Para poder utilizar a IDE, foi necessário instalar a ferramenta, escolhendo que tipo de instalação seria feito como mostra a Figura 18.

Figura 18 - Setup Sikuli



Fonte: Print Screen do Setup do Sikuli

Cada uma das opções descritas no Setup é diferente:

1. Pack1: instalar todas as opções (a IDE, poder rodar os scripts pelo cmd sem precisar da IDE, e um opcional para desenvolvimento em Java);
2. Pack2: instalar apenas a opção para rodar os scripts pelo cmd;
3. Pack3: pack adicional para poder desenvolver e rodar scripts na linguagem Jython usando o Eclipse ou qualquer outra IDE ou outros métodos;
4. Pack3: apenas desenvolver em Java ou Jython usando Netbeans, Eclipse, ou outra IDE, sem precisar da Pack1 ou Pack2;
5. Quer usar Funções OCR do Tesseract, contendo um aviso: "Você deve saber o que está fazendo e ter experiência!";
6. Quer os pacotes par usar no Windows, Mac e Linux, cada um contido em um pack cada.

Para o presente trabalho, foi escolhido o Pack1.

Foi necessário possuir o Java JDK instalado na máquina, porém não foi necessário adicionar nenhum caminho ao Path do Java, pois o Setup realiza a ação automaticamente. Não foi possível integrá-lo ao Eclipse e utilizar a linguagem Java pois não foi encontrada documentação que facilitasse a utilização do script com a ferramenta.

Com a ferramenta instalada e pronta para rodar os testes de forma automatizada, foi realizada a criação do script de automação de testes para cada um dos casos de teste, como mostra o Quadro 10 - TST-001.01 Automatizado. Nesse script cada um dos passos do caso de teste original de acordo com o quadro com um caso de teste descrito no Capítulo 4 como mostra o Quadro 10 - TST-001.01 Automatizado:

Quadro 10 - TST-001.01 Automatizado

Identificador	**TST-001.01**	**Código Sikuli IDE**
Item de teste	Login Padrão	

Especificações de Entrada	1. Usuário: fazion;
	2. Senha de acesso: 22;
	3. Clicar no botão "Ok".
		# Especificações de entrada:
		login = "fazion" #Login do sistema
		password = "22" #Senha do sistema

Procedimentos	1. Inserir o dado "Usuário" na área "Nome de Usuário"; 2. Inserir o dado "Senha de acesso" na área "Senha"; 3. Selecionar o botão "Enviar".	# Definição para abrir uma nova Janela do Chrome defopenff(): ffLoc = r'C:\Program Files (x86)\Mozilla Firefox\firefox. Exe' App(ffLoc).focus() # opens FF if exists("HttpBar.png",10): keyDown(Key.CTRL + "l") paste("https://localhost:8443/datafaz/") type(Key.ENTER) # Definição para realizar Login defsystemlogin(lgin, pssw): # Procura o padrão da imagemimageme seleciona com o mouse um campo 47 pixels deslocados para baixo no eixo X. # Em código puro: # t = find(Pattern("LoginTab.png").targetOffset(0,47))

t = find(

)

while not exists( ,10):

click(t)

wait(2)

#1. Inserir o dado "Usuário" na área "Nome de Usuário":

type(lgin)

type(tab)

wait(2)

#2. Inserir o dado "Senha de acesso" na área "senha":

paste(pssw)

t = r.find(

)

#3. Selecionar o botão "Enviar"

click(t)

Programa

Trypara verificação de Erros.

try:

Abre um Firefox e encaminha para a pagina de login

openff()

#Condicional para existência da janela de login

ifexists(

,10):

Definição da região r, para reduzir a área de procura e acelerar a pesquisa

r = find(

).below(200)

Realiza o Login, enviando usuário e senha

systemlogin("fazion", "22")

		# Condicional para verificar se o teste foi realizado corretamente ou não ifexists( ,10): logger.info(testName+ " realizado com sucesso!") # Exceção quando ocorre um erro, escreve em forma de DEBBUG no log except: logger.debug("Falha ao realizar " + testName + ": " + str(sys. Exc_info()[0]) + " " + str(sys.exc_info()[1]))
Especificações de Saída	Após o procedimento ser realizado, a tela deve ser alterada, apresentando a tela principal da aplicação.	# Após todo programa rodar, fecha a pasta de log finally: logging.shutdown()
Ambiente necessário	DataFazUnity, banco de dados.	

Exigências especiais	Não aplicável.	##### Importações #####
		import logging
		
		##### Inicialização de variáveis #####
		# Para alterar o nome do caso de teste mais rápido no log
		testName = "Caso de Teste TST - 001.01"
		
		#Variáveis auxiliares:
		tab = Key.TAB
		
		# Definição do log e em que pasta será salvo:
		my_dir = "C:\\Sikuli\\"
		logger = logging.getLogger(my_dir)
		hdlr = logging.fileHandler(my_dir + "log.txt")
		formatter = logging.formatter('[%(asctime)s %(levelname)s] %(message)s')
		hdlr.setFormatter(formatter)
		logger.addHandler(hdlr)
		logger.setLevel(logging. DEBUG)

Interdependências	Fazer uso do caso de teste específico e PST - Proposta Técnica.	

fonte: elaborada pelo autor

Da forma descrita no Quadro 11, é possível compreender como o teste automatizado realiza os passos do caso de teste na ferramenta proposta.

5.2. Dsenvolvimento de automação de testes

Nesta seção, é apresentada a execução da automação dos testes, mostrando a execução de testes manuais, automatizados, e as ferramentas utilizadas neste protótipo. Após, são mostrados os casos de testes manuais e automatizados, realizando uma comparação entre os dois tipos de testes.

5.2.1. Execução Manual e Automatizada dos casos de testes

Para efetuar os testes manuais, configurou-se um computador para possuir o sistema DataFaz Unity instalado localmente, o Enterprise Architect com a modelagem do DataFaz Unity (para organizar e documentar os casos de uso, e casos de teste com seus cenários), o PGadmin como banco de dados, e o arquivo "backup_UNITY.backup", que é um backup do banco de dados limpo (somente com a configuração inicial), usado como cenário inicial dos casos de teste.

Os casos de testes foram desenvolvidos com cenários positivos, possuindo entradas válidas, e um caso de teste onde será possível reproduzir um bug.

5.2.1.1. Teste Manual

Os testes manuais foram introduzidos conforme a sequência de teste, definido pela ordem de utilização do sistema. O quadro abaixo ilustra como é esta sequência:

Quadro 11 - Sequência de Teste

Sequência	Identificador Caso de teste
1	TST-001.01
2	TST-002.01
3	TST-002.02
4	TST-003.01
5	TST-004.01

6	TST-005.01
7	TST-006.01
8	TST-007.01
9	TST-008.01
10	TST-008.02

fonte: elaborada pelo autor

Cada caso de teste é analisado para elaboração dos testes e o seu tempo médio de execução dos casos de teste sendo realizados em sequência:

Quadro 12 - Dados teste manual

Métricas	Teste Manual
Tempo gasto para elaboração	05h00min
Tempo médio de execução	00h08min

fonte: elaborada pelo autor

O tempo gasto para elaboração dos testes envolveu um total de cinco horas. Este tempo foi composto pelo estudo da modelagem, descrição dos casos de teste, e descrição do documento de teste que contém os casos de teste.

5.2.1.2. Teste Automatizado

Os testes automatizados utilizaram os mesmos casos de teste, com os mesmos cenários que foram utilizados nos testes manuais. A mesma sequência de testes com os mesmos parâmetros de entrada e saída esperados foram utilizadas.

O quadro abaixo mostra a análise do tempo decorrido para a elaboração dos testes e o tempo médio de execução dos testes automatizados, de forma idêntica aos testes manuais, mas com a adição do tempo gasto para a elaboração dos Scripts dos testes automatizados

Quadro 13 - Dados teste automatizado

Métricas	Teste Automatizado
Tempo gasto para elaboração	05h00min
Elaboração Scripts de Testes Automatizados	07h30min
Tempo médio de execução	00h02min30seg

fonte: elaborada pelo autor

O tempo gasto para elaboração dos testes automatizados envolveu um total de doze horas e trinta minutos. Este tempo foi composto pelo estudo da modelagem, descrição dos casos de teste, e da elaboração dos scripts de testes automatizados.

A seguir, tem-se um quadro comparativo, mostrando as duas situações apresentadas nos casos de testes.

Quadro 14 - Comparações entre teste manual e teste automatizado

Métricas	Teste Manual	Teste Automatizado
Tempo gasto para elaboração	05h00min	05h00min
Elaboração Scripts de Testes Automatizados	-	07h30min
Tempo médio de execução	00h08min	00h02min30seg
Total de horas gastas	05h08min	12h32min30seg

fonte: elaborada pelo autor

Com base no Quadro 14 - Comparações entre teste manual e teste automatizado, nota-se que o total de horas gastas com a elaboração do teste automatizado foi superior ao total de horas gastas com a elaboração do teste manual. Este tempo é justificado nos próximos testes de software que virão a ser realizados nestes casos de teste, já que após a elaboração dos testes, e dos scripts, o tempo médio de execução que se torna relevante.

Para apoio na comparação entre a proposta de automação, foi realizada a aplicação de um questionário, apresentado no Apêndice I, com quatro questões, tendo elas cinco alternativas. O questionário foi aplicado a três profissionais da área de teste e qualidade de software, cujos dados podem ser vistos no Quadro 16, para que fosse possível levantar a opinião dos envolvidos.

Quadro 15 - Dados dos profissionais

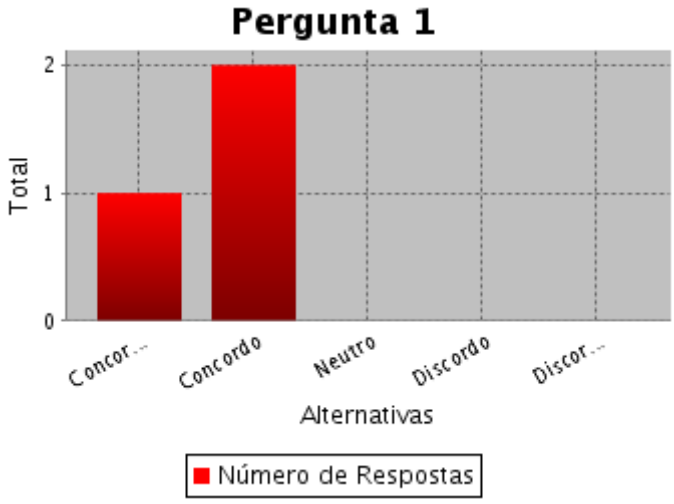
Nome	Idade	Escolaridade	Ocupação
Wilker Martins	28 Anos	Superior Completo	Analista da Qualidade do Produto (7 anos)
Gabriel Turnes	29 Anos	Superior Completo	Analista da Qualidade do Produto (3 anos)
Renata Faraco	28 Anos	Superior Completo	Engenheira (4 anos)

fonte: elaborada pelo autor

Para as respostas do questionário foi utilizada a escala de Likert[28], que se compreende como um tipo de escala de resposta psicométrica. Os gráficos mostrados abaixo mostram as respostas, quantas vezes foram selecionadas, para cada pergunta:

Pergunta 1: Os casos de testes disponibilizados foram facilmente compreendidos? Justifique sua resposta.

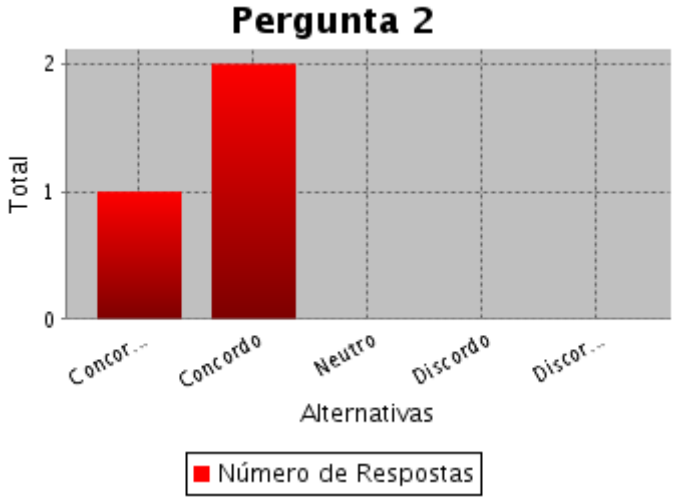
Gráfico 1 - Número de Respostas para a pergunta 1



fonte: elaborada pelo autor

Pergunta 2: Os cenários de testes disponibilizados cobriram a funcionalidade proposta pelos casos de testes? Justifique sua resposta.

Gráfico 2 - Número de Respostas para a pergunta 2

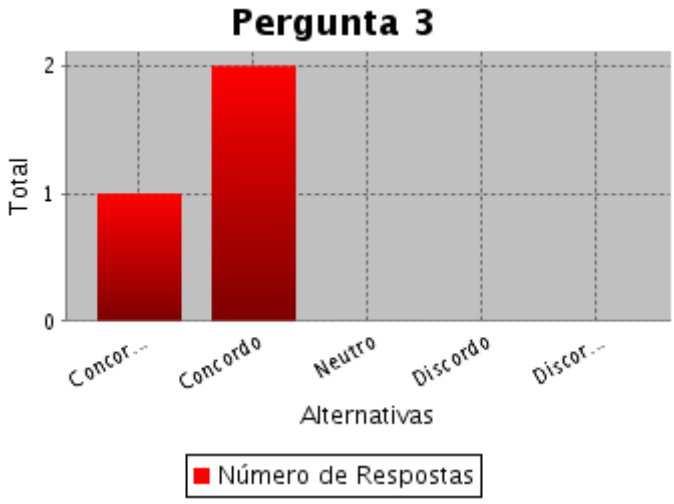


fonte: elaborada pelo autor

[28]<http://www.simplypsychology.org/likert-scale. Html>

Pergunta 3: A infraestrutura disponibilizada para o ambiente de testes foi satisfatória para a execução dos testes? Justifique sua resposta.

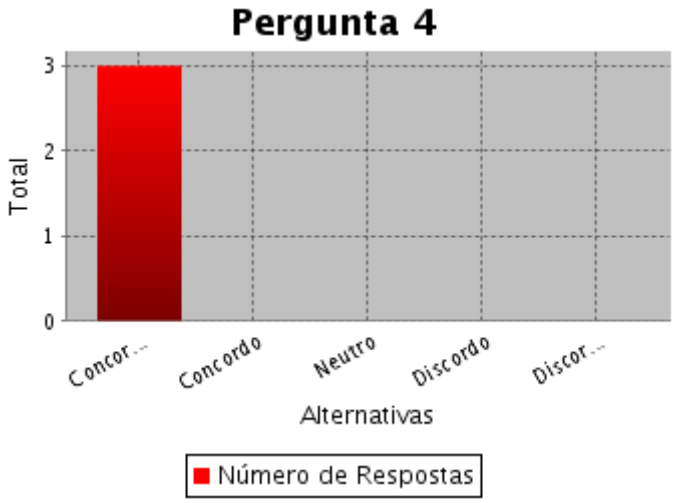
Gráfico 3 - Número de Respostas para a pergunta 3



fonte: elaborada pelo autor

Pergunta 4: Percebe-se que o tempo gasto com teste automatizado diminui esforços de custo e tempo com execuções. Como você avalia esta afirmação? Justifique sua resposta.

Gráfico 4 - Número de Respostas para a pergunta 4



fonte: elaborada pelo autor

O Quadro 16 informa as alternativas que foram utilizadas no questionário.

Quadro 16 - Alternativas questionario

Alternativas
Concordo Plenamente
Concordo
Neutro
Discordo
Discordo Plenamente

fonte: elaborada pelo autor

Os gráficos indicam uma impressão positiva da automação de testes da aplicação desenvolvida, aprovando as mudanças que foram introduzidas na organização citada como estudo de caso.

5.3. Discussão

Nesta seção, será comentado um pouco do que foi tratado no capítulo, analisando os dados coletados e resultados obtidos.

O capítulo detalhou os passos realizados para a automação de testes com a ferramenta de automação escolhida, detalhando como a ferramenta funciona, como são seus passos de instalação, como os casos de teste foram adaptado para código como foi mostrado no Quadro 10 - TST-001.01 Automatizado.

A pesquisa de automação de testes na plataforma Flex caminhou gradativamente, mostrando problemas em relação tanto ao Flex quanto às ferramentas de automação encontradas.

Alguns problemas foram relatados no decorrer do trabalho, como alguns navegadores impedirem o Flex de ser utilizado normalmente, sendo necessário trocar de navegador, ou encontrar uma versão estável. Em relação ao software de automação, existiu um caso, o login do sistema, em que a automação demora mais do que um teste manual, pois a demora na alteração da tela é mais lenta para o software do que ao ser humano.

Assim, os dados coletados na pesquisa levantam indícios de que automações de testes são bem-vindas. Ao serem repetidos, os testes automatizados tendem a reduzir o esforço normalmente necessário dos testadores para testar a mesma funcionalidade a cada nova versão publicada.

5.3.1. Ameaças à validade

Infelizmente, o estudo de caso ocorreu apenas com uma única empresa, não sendo possível abranger seus resultados de forma genérica, mesmo que positivos. Seria necessário encontrar mais empresas que utilizem a plataforma Flex, desenvolver casos de testes ao software destas empresas e coletar seus resultados.

Além disso, a empresa possui poucos analistas de testes, isso fez com que os resultados obtidos fossem escasso, mais analistas trariam mais respostas, e o questionário seria mais preciso.

Como descrito na 4.1.2, o autor do trabalho esteve ativo na execução dos testes manuais e automatizados, e este tempo ativo pode gerar viés de interpretação de dados coletados quando analisados e executados os testes.

6. CONCLUSÃO

Este trabalho trata do desenvolvimento de automação de testes para uma aplicação desenvolvida sobre a plataforma Flex. Um estudo de caso é conduzido onde a automação de testes é desenvolvida e avaliada.

O primeiro passo do trabalho consistiu em escolher uma ferramenta de automação que fornecesse facilidade na criação dos testes automatizados, sendo desenvolvidos requisitos para a escolha da ferramenta que se adequasse a estas necessidades levantadas.

Foram envolvidos estudos de literatura sobre automações de testes, em seguidas foram pesquisadas as limitações do Flex. A plataforma, baseada em Flash, possui falhas de segurança que permite que hackers invadam sistemas, e alguns navegadores, como o Firefox, começaram a bloquear a plataforma[29].

Tendo em mente como o Flex funciona com suas limitações, foram analisadas diferentes abordagens de uso, desde automação baseada totalmente em código utilizando webdrivers, como totalmente baseada em GUI. Foi possível notar, neste passo, que nenhuma ferramenta iria poder automatizar a aplicação em sua totalidade, então foram criados requisitos para escolher qual ferramenta seria utilizada no trabalho. A ferramenta Sikuli que cobriu a maior parte dos requisitos propostos foi escolhida.

Em seguida, foi necessário criar os casos de teste do sistema, que consequentemente viriam a contribuir diretamente no desenvolvimento dos scripts de testes automatizados.

Mais estudos, pesquisas, e instalações de produtos, tais como banco de dados, foram necessárias para poder utilizar a ferramenta escolhida sobre a Plataforma Flex, para então conseguir implementar e avaliar a automação dos testes.

Por fim, todo o esforço exercido garantiu que o esforço de testadores, e também para toda a área de teste, fosse reduzido. Também permitiu que o software possuísse uma melhor forma de ser testado, com menor possibilidade de possuir erros pela falha humana.

Os dados coletados de esforço e tempo no estudo de caso levantam indícios que o ganho na automação reduz o erro humano quando a automação realiza os passos, mas o ganho de tempo é pequeno, mas que não pode ser ignorado.

[29] <http://thenextweb. Com/insider/2015/07/14/firefox-rings-the-death-knell-for-flash/>

Um questionário aplicado para os membros da área de testes da organização demonstrou que os testadores aprovaram a automação de testes.

Espera-se que a organização envolvida no estudo de caso obtenha benefícios diretos com a redução de custos de retrabalho e aumento de confiabilidade da aplicação. Assim, é possível observar que o objetivo principal deste trabalho foi atingido com o desenvolvimento e avaliação da automatização de testes de uma aplicação Web desenvolvida utilizando plataforma Flex.

Portanto, nota-se que a área de tecnologia é abrangente, oferecendo diversas possibilidades e maneiras de resolver problemas do dia-a-dia que parecem não ter uma melhor solução.

6.1. Trabalhos Futuros

No presente trabalho, foi apresentado um estudo sobre uma aplicação já existente, sendo desenvolvidos cenários com testes funcionais. Diversas vertentes para estudos futuros podem ser abertos a partir deste trabalho.

Outros tipos de testes, alguns descritos no presente trabalho (carga, stress), podem ser aplicados, podendo ser verificado seu uso com a mesma ferramenta ou com ferramentas mais adequadas a este tipo de teste.

Além disso, a utilização de ferramentas de modelagem de testes seria uma ideia interessante, documentando os resultados do teste em uma ferramenta específica para organização de testes. Seria possível também unir com ferramentas de captura de log, casos de testes negativos e suas formas automatizadas.

A possibilidade de integração com o Jenkins[30] foi discutida para adicionar ao trabalho, caso fosse obtido sucesso, permitiria que os testes automatizados fossem realizados em momentos específicos para melhor controle e verificação do sistema. Porém, por falta de tempo, decidiu-se deixá-lo como uma possibilidade futura. Portanto, conclui-se que é possível abranger os testes com outras ferramentas, capturas de logs podem ser usados para corrigir bugs, determinar que a cada novo dia um pacote de testes já automatizados irá rodar o sistema, onde o analista de testes poderia chegar no trabalho e apenas verificar o log para ver o que houve e o que pode ser corrigido.

[30]<https://jenkins-ci.org/>

7. BIBLIOGRAFIA

ALVES, T. L., Silva, P., & Dias, M. S. (2014). Applying ISO/IEC 25010 Standard to Prioritize and Solve Quality Issues of Automatic ETL Processes. *2014 IEEE International Conference on Software Maintenance and Evolution*, 573–576. doi:10.1109/ICSME.2014.98

ALVES-MAZZOTTI, lta Judith; GEWANDSZNAJDER, Fernando. E. ed. *O método em ciências naturais e sociais*: pesquisa quantitativa e qualificativa. São Paulo: Thonsom, 1999.

BARTIÉ, Alexandre. **Garantia da qualidade de software:** As melhores práticas de Engenharia de Software aplicadas à sua empresa. Rio de Janeiro: Campus, 2002.

DataFaz. Disponível em <http://www.datafaz. Com > Acessado em: 10 de julho de 2015.

Engineering, S., Committee, S., & Computer, I. (2008). *IEEE Standard for Software and System Test Documentation*. *IEEE Std 829-2008* (Vol. 2008). doi: 10.1109/IEEESTD.2008.4578383

Escobar, J., Losavio, F., & Ortega, D. (2013). Standard quality model to Enterprise Architecture support tools. *2013 XXXIX Latin American Computing Conference (CLEI)*, 1–12. doi:10.1109/CLEI.2013.6670663

IEEE. (2008). *IEEE Std 829-2008, IEEE Standard for Software and System Test Documentation* (Vol. 2008).

IEEE. (2012). *IEEE Std 1012, IEEE Standard for Software Verification and Validation* (Vol. 2012).

IEEE. (2014). *Guide to the Software Engineering Body of Knowledge Version 3.0 (SWEBOK Guide V3.0)*.

ISO/IEC 9126. (2003), *Software engineering -- Product quality*. Disponível em <http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=22750> Acessado em Maio de 2015.

ISO/IEC 25000. (2014), *Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- Guide to SQuaRE*. Disponível em <http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=64764> Acessado em Maio de 2015.

KOSCIANSKI, André; SOARES, Michel dos Santos. Qualidade de Software: Aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software. 2. ed. São Paulo: Novatec, 2007

Kumhyr, D., Merrill, C., & Spalink, K. (1994). *Internationalization and Translatability*. *Conference of the Association for Machine Translation in the Americas* (Vol. 1). Retrieved from http://www.mt-archive.info/AMTA-1994-Kumhyr.pdf.

LAKATOS, Eva M.; MARCONI, Marina de A. Fundamentos de metodologia científica. 3ed. São Paulo: Atlas S.A., 1991.

MOLINARI, Leonardo. **Testes de Software:** Produzindo sistemas melhores e mais confiáveis. São Paulo: Érica, 2003.

Nunes, P. R. D. A. F., Ana, P., & Melo, C. V. De. (n. D.). Monografia : Teste de Componentes (2006).

PRESSMAN, Roger S. Engenharia de software: uma abordagem profissional; Tradução Ariovaldo Griesi e Mario moro Feccio; Revisão Técnica Reginaldo Arakaki, Julio Arakaki e Renato Manzan de Andrade. 7ª ed. São paulo: AMGH, 2011.

QUIVY, R. et al. (1992). Manual de Investigação em Ciências Sociais. Lisboa: Gradiva.

RÄTZMANN, Manfred; DE YOUNG, Clinton. Galileo Computing - Software Testing and Internationalization; Bonn: Galileo Press GmbH, 2002.

RUNESON, P.; HOST, M. Guidelines for conducting and reporting case study research in software engineering. Empir. Software Eng. 14, 131-164, 2009.

SABINO, Carlos A. *El processo de investigación*. Buenos Aires: Lumen-Humanitas, 1996.

SILVA. E. L da; MENEZES, E.M. Metodologia da pesquisa e elaboração de dissertação. 4. ed. Florianópolis: UFSC, 2005.

SOMMERVILLE, Ian. Engenharia de Software; tradução Selma Shin Shimizu Melnikoff, Reginaldo Arakaki e Edílson de Andrade Barbosa; Revisão técnica KechiHirama. 8ª edição. São Paulo: Pearson Addison Wesley, 2007.

Standard, I. (2010). INTERNATIONAL STANDARD ISO / IEC / IEEE, *2010*.

Standard, I., & Verification, S. (2012). *IEEE Standard for System and Software Verification and Validation* (Vol. 2012).

Stackoverflow sobre Flexmonkey. Disponível em: <http://stackoverflow.com/questions/7497645/i-want-to-automate-a-flash-application-using-Flex-monkey> Acessado em 31 de Maio de 2014

Seleniumhq. Disponível em: <http://www.seleniumhq.org/> Acessado em 20 Abril de 2015

SOFTWARE ENGINEERING INSTITUTE. The Capability Maturity Model: Guidelines for Improving the Software Process. Carnegie Mellon University, 1994. ISBN 0-201-54664-7.

Wikidocs do Adobe. Disponível em <https://wikidocs.adobe.com/wiki/display/Flex/Get+oriented+to+Flex> Acessado em: 17 de dezembro de 2014.

Weber, S. (2005). ASPE / MSC: Uma Abordagem para Estabelecimento de Processos de Software em Micro e Pequenas Empresas, 1–178.

ZAHRAN, S. Software Process Improvement: Practical Guidelines for Business Success. Edinburgh: Addison-Wesley,1998. ISBN 0-201-17782-X

ANEXOS

ANEXO A – CASOS DE TESTE

A seguir, encontram-se os casos de testes restantes e o primeiro cenário de cada um deles. Os demais cenários estão armazenados/documentados na ferramenta de modelagem.

Identificador	TST-002.01
Item de teste	Cadastrar Usuário
Especificações de Entrada	1. CPF: 929.934.782-49; 2. Nome: Márcio Eduardo; 3. E-mail: meduardo@datafaz. Com. Br; 4. Login: meduardo; 5. Senha: 1; 6. Perfil: Master; 7. Estado: SC; 8. Cidade: Florianópolis.
Procedimentos	1. Selecionar o botão "Configurações do sistema"; 2. Selecionar o botão "Datafaz Unity"; 3. Mover o mouse sobre a opção "Acesso"; 4. Selecionar o botão "Usuários" 5. Selecionar o botão "Novo"; 6. Inserir dado "CPF" no campo "CPF"; 7. Inserir dado "Nome" no campo "Nome"; 8. Inserir dado "E-mail" no campo "E-mail"; 9. Inserir dado "Login" no campo "Login"; 10. Inserir dado "Senha" no campo "Senha"; 11. Inserir dado "Perfil" no campo "Perfil"; 12. Selecionar a aba "Endereço"; 13. Inserir dado "Estado" no campo "Estado"; 14. Inserir dado "Cidade" no campo "Cidade"; 15. Selecionar o botão "Salvar".
Especificações de Saída	Os dados "Nome", "CPF", "E-mail, e "Status" do usuário aparecerão na lista de usuários cadastrados no sistema.
Ambiente necessário	DataFazUnity, banco de dados.
Exigências especiais	1. Estar logado no sistema [TST-001.01]; 2. Sistema não deve possuir um usuário com o nome "Marcio Eduardo" e/ou CPF "929.934.782-49" cadastrado no sistema.
Interdependências	Fazer uso do caso de teste especifico e PST - Proposta Técnica.

Identificador	TST-002.02
Item de teste	Editar Usuário
Especificações de Entrada	1. Nome: Pedro Henrique; 2. E-mail: phenrique@datafaz. Com. Br; 3. Login: phenrique.
Procedimentos	1. Selecionar o botão "Configurações do sistema"; 2. Selecionar o botão "Datafaz Unity"; 3. Mover o mouse sobre a opção "Acesso"; 4. Selecionar o botão "Usuários"; 5. Selecionar o usuário "Marcio Eduardo"; 6. Selecionar o botão "Editar"; 7. Alterar dados dos campos "Nome" pelos dados da Especificação de Entrada "Nome"; 8. Alterar dados dos campos "E-mail" pelos dados da Especificação de Entrada "E-mail"; 9. Alterar dados dos campos "E-mail" pelos dados da Especificação de Entrada "Login"; 10. Selecionar o botão "Salvar".
Especificações de Saída	Os dados "Nome", "CPF", e "E-mail" do usuário "Marcio Eduardo" devem ser alterados com os dados descritos nas Especificações de Entrada.
Ambiente necessário	DataFazUnity, banco de dados.
Exigências especiais	1. Estar logado no sistema [TST-001.01]; 2. Sistema deve possuir um usuário com o nome "Marcio Eduardo" cadastrado no sistema.
Interdependências	Fazer uso do caso de teste especifico e PST - Proposta Técnica.

Identificador	TST-003.01
Item de teste	Cadastrar Região
Especificações de Entrada	1. Descrição: Região Sul; 2. Imagem "Região Sul" no computador;
Procedimentos	1. Selecionar o botão "Configurações do sistema"; 2. Selecionar o botão "DatafazUnity"; 3. Mover o mouse sobre a opção "Ambiente"; 4. Selecionar o botão "Região" 5. Selecionar o botão "Novo";

	6. Inserir "Descrição" no campo "Descrição"; 7. Selecionar o botão "Carregar imagem..."; 8. Navegar até a pasta em que a imagem está; 9. Selecionar a Imagem "Região Sul" 10. Selecionar "Abrir"; 11. Selecionar "Salvar".
Especificações de Saída	O nome da Descrição, e seu status, aparecerão na lista de regiões cadastradas no sistema.
Ambiente necessário	DataFazUnity, banco de dados.
Exigências especiais	1. Não possuir uma região com Descrição "Região Sul" Cadastrada no sistema.
Interdependências	Fazer uso do caso de teste especifico e PST - Proposta Técnica.

Identificador	TST-008.03
Item de teste	Cadastrar Plataforma - alterar para associar alerta
Especificações de Entrada	1. Nome; 2. DataCenter; 3. ; 4. ; 5. ; 6. ; 7. ; 8. .
Procedimentos	1. Selecionar o botão "Configurações do sistema"; 2. Selecionar o botão "DatafazUnity"; 3. Mover o mouse sobre a opção "Acesso"; 4. Selecionar o botão "Usuários" 5. Selecionar o botão "Novo"; 6. ; 7. ; 8. .
Especificações de Saída	.
Ambiente necessário	DataFazUnity, banco de dados.
Exigências especiais	1. ; 2. .
Interdependências	Fazer uso do caso de teste especifico e PST - Proposta Técnica.

Identificador	TST-005.01
Item de teste	Cadastrar Dispositivo
Especificações de Entrada	1. Plataforma: SPECTO; 2. Rack: DCR UNITY 01; 3. Ambiente: Sala de Servidores; 4. Tipo Sensor de Fumaça; 5. ID0x01 - 1.
Procedimentos	1. Selecionar o botão "Configurações do sistema"; 2. Selecionar o botão "Datafaz Unity"; 3. Mover o mouse sobre a opção "Hardware"; 4. Selecionar o botão "Dispositivos"; 5. Selecionar o botão "Novo"; 6. Inserir o valor "Plataforma" no campo "Plataforma"; 7. Inserir o valor "Rack" no campo "Rack"; 8. Inserir o valor "Ambiente" no campo "Ambiente"; 9. Inserir o valor "Tipo" no campo "Tipo"; 10. Selecionar a aba "Hardware"; 11. Inserir o valor "ID" no campo "Adicionar ID"; 12. Selecionar "Salvar".
Especificações de saída	O Código, Nom, Nome Componente, Empresa, Plataforma, Tipo, Rack, Matriz, e seu status, aparecerão na lista de dispositivos cadastradas no sistema..
Ambiente necessário	DataFazUnity, banco de dados.
Exigências especiais	1. Não possuir um dispositivo com mesmo "TIpo" e "ID" cadastrado no sistema; .
Interdependências	Fazer uso do caso de teste especifico e PST - Proposta Técnica.

Identificador	TST-006.01
Item de teste	Download Relatório
Especificações de Entrada	1. Categoria: Cadastro; 2. Tipo: Usuários; 3. Data Center: SpectoFazion SC; 4. Relatorio: Total de Usuários.
Procedimentos	1. Selecionar o botão "Relatórios"; 2. Inserir o valor "Categoria" no campo "Categoria"; 3. Inserir o valor "Tipo" no campo "Tipo";

	4. Remover seleção do Checkbox"Considera período"; 5. Inserir o valor "Data Center" no campo "Data Center"; 6. Inserir o valor "Relatorio" no campo "Tipo de Relatório"; 7. Selecionar o botão "PDF"; 8. Selecionar o botão "OK".
Especificações de Saída	Um PDF com o total de usuários cadastrados no Data Center deve ser baixado para o computador.
Ambiente necessário	DataFazUnity, banco de dados.
Exigências especiais	1. Possuir o Data Center "SpectoFazion SC" cadastrado no sistema.
Interdependências	Fazer uso do caso de teste especifico e PST - Proposta Técnica.

Identificador	TST-007.01
Item de teste	Enviar Comando
Especificações de Entrada	Dispositivo: Mezanino - 0x03 - Sensor de Fumaça; Senha: 1.
Procedimentos	1. Selecionar o botão "Configurações do sistema"; 2. Selecionar o botão "Datafaz Unity"; 3. Mover o mouse sobre a opção "Automação"; 4. Selecionar o botão "DataConf" 5. Abrir a opção do Data Center "SpectoFazion SC"; 6. Abrir a opção da Plataforma "SPECTO"; 7. Abrir a opção do dispositivo "DCR UNITY 01"; 8. Selecionar o Checkbox do Dispositivo do valor "Dispositivo"; 9. Selecionar o botão "Comandar"; 10. Selecionar o botão do comando "Ativar"; 11. Inserir o valor "Senha" no campo "Senha de Confirmação"; 12. Selecionar o botão "Validar".
Especificações de Saída	A mensagem "Comando Enviado com Sucesso" deve aparecer, e os dados do dispositivo e comando devem aparecer na lista de comandos enviados na janela do DataConf.

Ambiente necessário	DataFazUnity, banco de dados.
Exigências especiais	1. Possuir o Dispositivo"Mezanino - 0x03 - Sensor de Fumaça" cadastrado e inativo no sistema.
Interdependências	Fazer uso do caso de teste especifico e PST - Proposta Técnica.

Identificador	TST-008.01
Item de teste	Cadastrar Alerta
Especificações de Entrada	1. Nome: Alerta Baixa Temperatura; 2. Mensagem: A temperatura está muito baixa!; 3. E-mail: True; 4. SMS: False; 5. Aviso Sonoro: False; 6. Prioridade: Alta Prioridade; 7. Áudio "ABT" no computador.
Procedimentos	1. Selecionar o botão "Configurações do sistema"; 2. Selecionar o botão "Datafaz Unity"; 3. Mover o mouse sobre a opção "Acesso"; 4. Selecionar o botão "Ambiente" 5. Selecionar o botão "Alerta"; 6. Selecionar o botão "Novo"; 7. Inserir valor "Nome" no campo "Nome"; 8. Inserir valor "Mensagem" no campo "Mensagem"; 9. Se valor de "E-mail" for True, Selecionar Checkbox da opção "Enviar e-mail". Senão ignorar; 10. Se valor de "SMS" for True, Selecionar Checkbox da opção "Enviar SMS". Senão ignorar; 11. Se valor de "Aviso Sonoro" for True, Selecionar Checkbox da opção "Exibir Aviso Sonoro". Senão ignorar; 12. Selecionar o botão "..." do campo "Som"; 13. Navegar até a pasta em que a imagem está; 14. Selecionar o som do valor "ABT"; 15. Selecionar "Abrir"; 16. Selecionar "Salvar".
Especificações de Saída	O nome e status do alerta devem aparecer na lista de Alertas cadastrados no sistema.
Ambiente necessário	DataFazUnity, banco de dados.

Exigências especiais	1. Não possuir um Alerta "Alerta Baixa Temperatura" cadastrado no Sistema.
Interdependências	Fazer uso do caso de teste especifico e PST - Proposta Técnica.

Identificador	TST-008.02
Item de teste	Adiar Alerta
Especificações de Entrada	1. Tempo: 15 minutos
Procedimentos	1. Selecionar aaba "Alertas"; 2. Selecionar o botão "Resolver Alertas"; 3. Selecionar Alerta "Alerta Baixa Temperatura"; 4. Adicionar o valor "tempo" no campo "hh:mmn:ss" ao lado do botão "Adiar"; 5. Selecionar o botão "Adiar"; 6. Selecionar "OK" .
Especificações de Saída	O Alerta deve ser removido da lista de alertas que aparecem na aba "Alertas" e da janela "Alertas", retornando após o período do valor "Tempo" terminar, caso ainda esteja acontecendo.
Ambiente necessário	DataFazUnity, banco de dados.
Exigências especiais	1. Possuir o Alerta "Alerta Baixa Temperatura" Cadastrado e ativo no sistema [TST-008.01]; 2. O Alerta "Alerta Baixa Temperatura" deve estar acontecendo.
Interdependências	Fazer uso do caso de teste especifico e PST - Proposta Técnica.

ANEXO B – QUESTIONÁRIO

UNIVERSIDADE FEDERAL DE SANTA CATARINA
Título do Trabalho: Automação de testes para plataforma FLEX
Aluno: Augusto Boehme Tepedino Martins
Orientador: Jean Carlo Rossa Hauck
QUESTIONÁRIO – TESTE AUTOMATIZADO PARA PLATAFORMA FLEX
Informações: Este questionário tem por objetivo levantar as percepções dos participantes sobre o teste automatizado suportando por uma ferramenta de modelagem, bem como suas opiniões a respeito das rotinas de teste e ferramentas utilizadas. Solicitamos a sua colaboração no preenchimento das questões.

IDENTIFICAÇÃO DO PARTICIPANTE

Qual seu nome e idade?

_____

Qual é a sua escolaridade?

_____
_____

Atua na área de testes e/ou qualidade de *software*? Se a resposta for positiva, qual é o seu cargo e há quanto tempo atua nesta área?

_____
_____
_____

ROTINA DE TESTE

1. Os casos de teste disponibilizados foram facilmente compreendidos? Justifique sua resposta.

( ) Concordo plenamente.
( ) Concordo.
( ) Neutro.
( ) Discordo.
( ) Discordo plenamente.

_____
_____

2. Os cenários de testes disponibilizados cobriram a funcionalidade proposta pelos casos de teste? Justifique sua resposta. Justifique sua resposta.

( ) Concordo plenamente.
( ) Concordo.
( ) Neutro.
( ) Discordo.
( ) Discordo plenamente.

_____
_____

3. A infraestrutura disponibilizada para o ambiente de teste foi satisfatória para a execução dos testes? Justifique sua resposta.

( ) Concordo plenamente.
( ) Concordo.
( ) Neutro.
( ) Discordo.
( ) Discordo plenamente.

_____
_____

4. Percebe-se que o tempo gasto com teste automatizado diminui esforços de custo e tempo com execuções. Como você avalia essa afirmação? Justifique sua resposta.

( ) Concordo plenamente.
( ) Concordo.
( ) Neutro.
( ) Discordo.
( ) Discordo plenamente.

_____
_____

ANEXO C – TERMO DE AUTORIZAÇÃO

TERMO DE AUTORIZAÇÃO PARA DIVULGAÇÃO DE INFORMAÇÕES DE EMPRESAS

Razão Social: Specto Painéis Eletrônicos Ltda.

Fantasia: Specto Tecnologia **CNPJ:** 95.849.642/0001-76

Inscrição Estadual: 252.631.242 **Inscrição Municipal:** 9005248

Endereço completo: Endereço: Rua Walter José Correia, Lote 12 Sertão do Maruim - São José - Santa Catarina. **Cep:** 88122-035

Representante da Empresa: Moacyr Franco Neto

Telefone: (48) 3049-9900 **E-mail:** specto@specto.com.br
Site: www.specto.com.br

Tema: Automação de testes para plataforma FLEX

Autor: Augusto Boehme Tepedino Martins

Código de Matrícula: 10202973

Orientador: Jean Carlo Rossa Hauck

Curso: Ciência da Computação

Como representante da empresa acima nominada para este trabalho, declaro que as informações contidas no corpo do trabalho de conclusão de curso podem ser publicadas

Moacyr Franco de M. Neto — Representante da empresa

Florianópolis, 15 de Setembro 2015 — Local e Data