INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SUL-RIO-GRANDENSE - CAMPUS PASSO FUNDO
CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

EDUARDO RODRIGUES DOS SANTOS

ANÁLISE DE FERRAMENTAS PARA AUTOMAÇÃO DE TESTES DE SOFTWARE EM SISTEMAS WEB

PASSO FUNDO
2018

EDUARDO RODRIGUES DOS SANTOS

ANÁLISE DE FERRAMENTAS PARA AUTOMAÇÃO DE TESTES DE SOFTWARE EM SISTEMAS WEB

Monografia submetida como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet no Curso de Tecnologia em Sistemas para Internet do Instituto Federal Sul-Rio-Grandense, Campus Passo Fundo.

Orientador: Me. André F. Rollwagen

PASSO FUNDO

2018

EDUARDO RODRIGUES DOS SANTOS

ANÁLISE DE FERRAMENTAS PARA AUTOMAÇÃO DE TESTES DE SOFTWARE EM SISTEMAS WEB

Trabalho de Conclusão de Curso aprovado em ____/____/____ como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet

Banca Examinadora:

_____

André Fernando Rollwagen

_____

Carmen Vera Scorsatto

_____

Rafael Marisco Bertei

_____

Coordenação do Curso

PASSO FUNDO

2018

RESUMO

O avanço computacional possibilitou o surgimento de diferentes plataformas de softwares, como a própria internet, que é a mais utilizada por aplicações WEB, portais e outros tipos de aplicações. Muitos dos sistemas passam por etapas de testes, tanto no desenvolvimento quanto em produção. A automação de testes torna-se interessante, quando for necessário testar uma aplicação funcionalmente, poupando tempo do testador, e as ferramentas de automação facilitam essa etapa. O presente trabalho traz um estudo, analisando as ferramentas de teste de software Badboy, Selenium IDE e Sikuli. Um sistema WEB foi desenvolvido como ambiente de testes, permitindo aplicar as ferramentas e testar as suas funcionalidades. Assim, vantagens e limitações foram identificadas através de diferentes aspectos analisados, como métodos, tratamento com código, tempos de execuções e documentação**.**

Palavras-chave: Teste de software, Automação de Testes, Ferramentas de teste.

ABSTRACT

The computational advancement allowed the emergence of different software platforms, such as the internet itself, which is the most used among thousands of websites, portals and other types of programs. Many systems go through testing steps, both in development and in production. Test automation is an alternative to testing a tester's time-consuming system, and automation tools make this stage even easier. The present work brings a study, analyzing the software testing tools Badboy, Selenium IDE and Sikuli. A WEB system was developed as a test environment, allowing applying the tools testing their functionalities. Thus, advantages and limitations were identified through different aspects analyzed, such as functionalities, code handling, execution times and documentation.

Keywords: Software Testing, Test Automation, Test Tools.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Com o avanço tecnológico a velocidade de desenvolvimento de software está tonando-se cada vez mais ágil. A entrega de um software que poderia demorar meses, pode ser finalizada em semanas, dependendo da complexidade do mesmo. Devido ao surgimento de frameworks, bibliotecas prontas e as próprias linguagens de programação que vieram evoluindo com o tempo, tornaram o processo de desenvolvimento mais ágil e simples, é possível entregar mais softwares em menos tempo. Porém com toda essa agilidade, a qualidade de muitos dos programas é posta à prova, além de o sistema ser finalizado com bugs e falhas.

Uma prática a ser usada no desenvolvimento de programas é acrescentar etapas de testes de software antes de ser concluído e após sua finalização. Um teste pode, além de agregar qualidade ao produto final, prevenir grandes perdas como: tempo, dinheiro e diversos outros tipos de danos. Os sistemas que são classificados como aplicações WEB, exigem um tipo de teste mais especifico, com inserção de valores repetidamente, os quais são testados usando as métricas de automação de testes. Para facilitar a aplicação destes testes, que exigem do usuário uma atenção redobrada para analisar e testar cada parte do sistema, cujo pode ser extremamente grande e fazer com que a experiência do testador seja exaustiva, se torna muito mais viável a utilização de ferramentas de automação de testes. Tais ferramentas podem auxiliar o testador a criar um teste e replicar seu script a todas as situações semelhantes a que ele deseja testar. Na automação de testes existem muitas ferramentas que auxiliam no processo, porém são apresentadas com foco no teste como um todo, essa generalização pode prejudicar um usuário que deseja utilizar uma ferramenta para automatizar seus testes, pelo fato dessas não possuírem documentação que fosse auxiliar na escolha de qual ferramenta pode ser mais adequada em relação a sua necessidade.

1.1 JUSTIFICATIVA

As diferentes características presentes nas ferramentas de automação de testes de software, permitem a elaboração de um estudo através de uma análise comparativa, verificando vantagens e limitações que estas podem apresentar ao implementar determinadas rotinas de teste em um sistema.

Os aspectos negativos podem influenciar ao definir qual ferramenta pode ser utilizada pelo testador e os aspectos positivos, quando unificados, podem ser empregados no desenvolvimento de uma aplicação que complete os recursos mais vantajosos identificados em cada teste executado por cada ferramenta. “Uma ferramenta de teste reduz a intervenção humana nos resultados obtidos, aumentando a qualidade de teste. Influencia diretamente a confiabilidade do software testado” (SOMMERVILLE, 2007, p. 68).

Uma comparação entre ferramentas de teste que tem o mesmo seguimento, como, por exemplo, automação, influencia diretamente na escolha do software para testar uma aplicação. Com uma ferramenta de testes que possui uma descrição que apresenta seus pontos positivos em determinado modelo de testes, ela, se torna mais recomendada em relação a outra que obteve um resultado inferior, agregando qualidade no teste como um todo.

1.2 OBJETIVOS

O objetivo desse trabalho é realizar um comparativo entre as ferramentas de testes automatizados de software para o desenvolvimento de aplicações Web. A partir desse objetivo geral, alguns objetivos específicos foram definidos para o desenvolvimento do trabalho:

- Projetar um sistema Web como um ambiente de testes para o estudo.
- Implementar testes automatizados de software com as ferramentas selecionadas na plataforma de testes.
- Analisar e comparar as ferramentas de teste em relação às suas diferentes características.
- Identificar vantagens e desvantagens em cada ferramenta de automação de teste de software.

1.3 ESTRUTURA DA MONOGRAFIA

Visando a melhor organização e estruturação, esta monografia foi dividida em capítulos:

O Capítulo 2 (Referencial Teórico): Descreve os fundamentos necessários para a aplicação de testes de software e utilização de ferramentas de automação de testes. Após são apresentados os Trabalhos Relacionados sobre testes de softwares apresentando pontos positivos e negativos dos mesmos. No Capítulo 4, a Metodologia, apresenta as técnicas e métodos utilizados no desenvolvimento do sistema e na aplicação dos testes automatizados. Em seguida são exibidos os Resultados, onde estão descritos e ilustrados os resultados obtidos a partir da metodologia empregada.

2 REFERENCIAL TEÓRICO

Neste capítulo são discutidos aspectos relacionados aos testes de softwares, bem como suas técnicas e ferramentas que auxiliam sua execução. Foram realizadas pesquisas bibliográficas que serviram como base para o desenvolvimento do presente trabalho.

2.1 TESTES DE SOFTWARE

O teste é destinado a mostrar que um programa faz o que é proposto a fazer e para descobrir os defeitos do programa antes do uso. “Quando se testa o software, o programa é executado usando dados fictícios. Os resultados do teste são verificados à procura de erros, anomalias ou informações sobre os atributos não funcionais do programa” (SOMMERVILLE, 2011, p.144).

O processo de testes pode ter muitos objetivos, dependendo de quem lhe executará, mas é ressaltado por grande parte da comunidade teste de software[1], que o processo de testes tem dois objetivos distintos, os quais são:

- Mostrar ao cliente, que vai receber o software e ao programador dessa aplicação, que o programa tem condições de atender aos seus requisitos, cujos, foram informados no projeto pelo cliente. Pensando em softwares customizados, é necessário que haja ao menos um teste para cada especificação do documento de requisitos. Softwares que abrangem grande parte das necessidades do mercado, denominados softwares de prateleira, são definidos como “programa de computador produzido em larga escala de maneira uniforme e colocado no mercado para aquisição por qualquer interessado sob a forma de cópias múltiplas” (NASRALLAH, 2016, p.1), são imunes ao processo de testes, partindo da idéia de que o programa que deu origem as cópias já tenha sido testado.
- O segundo objetivo busca levar aos testes de defeitos, os casos de teste que são feitos para expor as falhas. Os testes não podem demonstrar se o software é livre de defeitos ou se ele se comporta conforme foi especificado em qualquer situação. É sempre possível que um teste que tenha sido esquecido seja aquele que poderia descobrir mais problemas no sistema. Como expressivamente citou Edsger Dijkstra, um pioneiro colaborador das práticas de desenvolvimento da engenharia de software (DIJKSTRA, 1972).

O teste faz parte de dois passos cruciais, a verificação e a validação. Verificação e validação não são a mesma coisa, embora sejam frequentemente confundidas. Porém Justen (2015) alerta que os ciclos de testes devem ser respeitados, mesmo se o testador já sabe como implementá-los do início ao fim.

O objetivo dos processos de validação e verificação é verificar se o software em seu desenvolvimento e aprimoramento, irá satisfazer as suas recomendações e permitir a funcionalidade solicitada pelos clientes. Os processos que realizam as verificações, são iniciados a partir do momento em que os requisitos são obtidos, e persistem por todas as etapas do desenvolvimento. A verificação tem como objetivo, fiscalizar a aplicação, de modo que a mesma atenda os requisitos funcionais e os requisitos não funcionais. A validação, trata-se de um desenvolvimento mais amplo,

[1] "Fóruns/Comunidades de teste de software | Agile Testers." 21 out. 2015, http://agiletesters.com.br/post/842. Acessado em 24 out. 2017.

ou seja processos menos detalhados, tem como objetivo, levar a garantia de que o software irá suprir as necessidades e atender as expectativas do cliente. Esse processo não só realiza a verificação das especificações, mas também, demonstra que a aplicação funciona de maneira que suas expectativas sejam superadas.

Para Sommerville (2011) os processos de validação e verificação, tem como finalidade, estabelecer a confiança e segurança, para que a aplicação esteja concluída de acordo com suas especificações. Isso quer dizer que o sistema como um todo, precisa ser útil o suficiente de acordo com seu propósito. A confiança que será exigida, vai depender dos requisitos iniciais do sistema, das opiniões dos clientes e das possibilidades dos usuários.

Na finalidade do software, o mais importante é a confiança que o mesmo dispõe. O nível de aceitação necessário para uma aplicação controlar um sistema de segurança, é no mínimo maior do que os riscos previstos. Controlar as expectativas de um usuário é uma tarefa complicada, pois, caso o cliente já tenha passado por uma experiência com softwares defeituosos ou porventura tenha obtido alguma falha na segurança, deixa de se criar grandes expectativas dos usuários em relação a qualidade de software. Mesmo após uma falha do sistema, o cliente não fica mais surpreso e quando é imposto um novo sistema, os clientes têm uma tolerância maior a falhas, porque os benefícios de usabilidade compensam os custos para recuperar as falhas. Tendo essas hipóteses em vista, o teste de software pode ser parcialmente ignorado. No entanto, com o tempo o sistema vai amadurecendo, os usuários irão esperar que ele venha a ser mais confiável e tolerante a falhas, logo, testes mais completos de versões futuras podem ser necessários.

Introduzir inspeções no código fonte do sistema, tem como objetivo observar os requisitos ou o modelo do projeto. Após inspecionar um sistema, pode ser usado para revelar erros : conhecimento do sistema, domínio da aplicação, linguagem de programação ou a modelagem desse sistema. Bem como a procura por defeitos de programa, uma inspeção pode considerar outros atributos de qualidade de um programa, como a conformidade com os padrões, portabilidade e manutenibilidade. Pode-se procurar ineficiências, algoritmos inadequados e um estilo pobre de programação que poderiam tornar o sistema de difícil manutenção e atualização (SOMMERVILLE, 2011). As inspeções de programa são uma ideia antiga, e vários estudos e experimentos demonstraram que as inspeções são mais eficazes na

descoberta de defeitos do que os testes de programa. Fagan (1986) relatou que mais de 60% dos erros em um programa podem ser detectados por meio de inspeções informais de programa. No processo *Clean room*[2] (PROWELL , 1999), afirma que mais de 90% dos defeitos podem ser descobertos em inspeções de programas. A esse respeito, Dutra (2014) declara: " O uso desta abordagem tem levado a construção de softwares com poucos erros. Os custos desses projetos foram comparáveis a outros projetos que usaram técnicas de desenvolvimento convencionais" (p.1).

2.2 TÉCNICAS DE TESTE DE SOFTWARE

Entre os métodos usados para executar testes de software, encontram-se nomenclaturas que denominam esses meios, como: Teste unitário, teste de persistência, teste de base de dados, entre outros. Os quais são encontrados em muitos materiais na internet e mais detalhadamente em livros da área de engenharia de software. Os tipos de testes, além de facilitar a tarefa para o testador no momento de escolher o método a ser aplicado, servem como esclarecimento ao cliente, informando o que realmente foi testado em seu sistema, pois cada tipo de teste é conceituado e possui documentação a seu respeito. Segundo Pressman (1995), técnicas de teste de software consistem em elaborar casos de teste capazes de varrer o software a procura de erros ainda não constatados. O objetivo dos testes não é apontar um culpado pelo erro. Os testes têm como meta encontrar problemas e, logo após, buscar sua correção, assim o sistema conta com uma dose a mais de qualidade. Afirma Sommerville (2003) que desfrutando de uma lógica ou método de teste, ao concluir o mesmo, é necessária uma avaliação dos resultados obtidos, comparando-os com as expectativas iniciais e se por ventura for detectado erros, deve ser feita uma depuração.

[2] "Cleanroom – Metodologia muito utilizada no desenvolvimento de software.

2.2.1 TESTE ESTRUTURAL OU TESTE DE CAIXA BRANCA

Para os testes de estruturas, chamados de teste de caixa branca, é exigido do testador um conhecimento aprofundado da aplicação, pois os testes são baseados no código fonte da mesma ou em meios de testar o próprio hardware. “Assim, os testes serão elaborados para: testar as decisões lógicas (verdadeiro/falso), testar os "*loops*" até o limite, as variáveis estáticas e dinâmicas, dentre outros” (COUTINHO,2011, p.12). Os métodos usados na técnica de caixa branca, retratam ideias paralelas aos testes de caixa preta, porém ambos buscam o mesmo sentido de testes. “Nesta metodologia os casos de teste são gerados, tendo se conhecimento da estrutura interna (lógica) do programa” (OTTONI ,2010, p.3). São também denominados pelo autor como testes estruturais:

- Cobertura Lógica (Critérios Myers);
- Método dos Caminhos Básicos.

Teoricamente, um teste de caixa branca seria capaz de avaliar e classificar um software como 100% correto, isso porque o teste de caixa branca percorre todos os caminhos lógicos possíveis. Em contrapartida, resulta num teste exaustivo e complexo, pois mesmo em um programa pequeno, os caminhos lógicos são muitos (PRESSMAN, 1995).

2.2.2 TESTE FUNCIONAL OU TESTE DE CAIXA PRETA

O teste funcional de software possui o codinome de teste de caixa preta. O teste de caixa preta é basicamente a comparação de resultados esperados pelo testador de acordo com a sua aplicação, assim é possível corrigir os erros e proporcionar mais qualidade ao sistema. Este teste consiste em validar os requisitos funcionais do sistema, o objetivo é encontrar erros de interface, erros relacionados ao desempenho, ao acesso ao banco de dados, erros de inicialização e término, falha ou ausência de funções (PRESSMAN, 1995).

> O teste funcional também é conhecido como teste caixa preta pelo fato de tratar o software como uma caixa cujo conteúdo é desconhecido e da qual só é possível visualizar o lado externo, ou seja, os dados de entrada fornecidos e as respostas produzidas como saída. Na técnica de teste funcional são

> verificadas as funções do sistema sem se preocupar com os detalhes de implementação. O teste funcional envolve dois passos principais: identificar as funções que o software deve realizar e criar casos de teste capazes de checar se essas funções estão sendo realizadas pelo software . As funções que o software deve possuir são identificadas a partir de sua especificação. Assim, uma especificação bem elaborada e de acordo com os requisitos do usuário é essencial para esse tipo de teste. (BARBOSA, MALDONADO, VINCENZI, 2014, p.7)

Os testes funcionais de software são melhor implementados com o auxílio de ferramentas para automação destes testes. Essas ferramentas que ajudam na automação, fazem o trabalho duro, que apenas aplicados pelos testadores não seriam totalmente suficientes, ainda mais com sistemas de grande porte, em que os possíveis resultados e valores de retorno tornam-se exaustivos, ao serem aplicados por mãos humanas.

2.3 TESTES EM APLICAÇÕES PARA WEB

Testes em softwares que não se comunicam com a internet, possuem uma enorme diferença no momento em que estiverem sendo realizados, com relação aos softwares WEB (*WebApp*). Comparações entre esses dois tipos de software tornam-se inviáveis, pelo fato de que são totalmente diferentes das suas lógicas de desenvolvimento, logo, seus padrões e técnicas de testes no sistema também serão distintas. Para Pressman (2011) o universo dos programas desenvolvidos para a WEB, tem como objetivo, além das funcionalidades rodando corretamente, a velocidade do desenvolvimento e a segurança que o sistema possuirá. O cliente que necessita de um sistema pela WEB procura sempre a praticidade, mas esse é um requisito que se torna delicado, perante aos infinitos meios de ameaças que cercam um sistema com conexão com a internet.

Zanoni (2014) ressalta algumas dificuldades encontradas ao testar uma aplicação como: Falsa sensação de segurança; olhar viciado; concentração polarizada e concentração em um único ponto. Estratégias e planejamentos de teste estabelecem o que é crítico. Campos (2009) aponta a importância do mapeamento das partes de um projeto, com todas as fases e o fluxo do processo e representação em gráficos, níveis de detalhes diferenciados que caracterizam as atividades de trabalho e tarefas de responsabilidades.

2.3.1 AMBIENTES WEB

Quando o foco do assunto é desenvolver um sistema para web, ou até mesmo usufruir de um, além dos cuidados em atender os diferentes tipos de dispositivos que farão o uso desse sistema, é extremamente importante que o sistema interaja minimamente bem com o ambiente. Um ambiente web que não dá suporte a uma determinada aplicação, certamente está defasado ou não recebe atualizações por meio de seus desenvolvedores. Além de atender incontáveis sistemas, os ambientes web interagem cada vez mais com aplicações desktop, o que proporciona através de *plugins* de interação o uso de ferramentas que trocam informações com a navegação na nuvem.

Para Cavalcanti (2006) é necessário propiciar a interatividade ambiente/aplicação, a aplicação deve criar atividades que facilitem a comunicação com o ambiente, ou seja, que proporcionem uma troca de códigos entre os navegadores para que, juntos, construam a finalidade que foi inicialmente proposta. "Quando se trata de aplicações web, todo o seu meio informativo é referido a sistemas ou sites onde grande parte da programação fica hospedada em servidores na internet, e o usuário (cliente) normalmente não precisa ter nada instalado em sua máquina para utilizá-las, além de um navegador (browser)" (FARIA, 2015, p.17).

O ambiente tem um papel importante no diagnóstico de todos os erros encontrados durante o teste do *WebApp*, em algumas situações (por exemplo, teste de conteúdo), o local dos erros é óbvio, mas em muitos outros tipos de teste do *WebApp* (por exemplo, teste de navegação, teste de desempenho, teste de segurança) a causa subjacente do erro pode ser consideravelmente mais difícil de determinar (PRESSMAN, 2011).

2.3.2 TESTE DE CONTEÚDO

O teste de conteúdo tem como objetivo: apontar imperfeições e erros na estrutura do programa, visando a apresentação final ao usuário. Para que o teste de conteúdo seja válido, é necessário que o teste descubra os erros de sintaxe

(ortografia, gramática) em arquivos de texto, conteúdos de URL e outros meios. Vitalmente o teste de conteúdo tem como objetivo principal descobrir os erros de integridade das informações, pois sem isso, nem uma visualização do sistema será permitida. Para Pressman (2011) mesmo que alguns dos erros que ocorrem no conteúdo de uma aplicação desenvolvida para WEB sejam pequenos, exibindo uma informação incorreta, pode acarretar sérios reparos futuros, que levam tempo e colocam a prova a qualidade do sistema. Uma das bases fundamentais do teste de conteúdo do sistema é a preocupação com leis e propriedades intelectuais, que venham a fazer parte do universo de que está sendo desenvolvido o sistema. O teste de conteúdo tenta descobri-los antes que os mesmos sejam percebidos pelos próprios usuários. Para a aplicação mais eficiente da técnica de teste de conteúdo, o profissional envolvido deve buscar padronização de testes. Por isso ele precisa ser crítico diante do problema que precisa ser corrigido (CANDELORO, 2015).

2.4 SOFTWARE DE CÓDIGO ABERTO

O software de código aberto é o programa de computador com o seu código fonte disponibilizado e licenciado com uma licença de código aberto. Na definição de GUGIK (2009), grande parte dos aplicativos podem ser modificados e redistribuídos livremente, mas o desenvolvedor tem o direito de estabelecer algumas restrições. Software de código aberto frequentemente têm desenvolvimento público, de maneira cooperativa. Atualmente existe uma organização que controla as normas e regras para o segmento de softwares de código aberto, trata-se da *Open Source Initiative* (OSI)[3], que foi criada para incentivar uma aproximação de entidades comerciais com o software livre.

[3] "Open Source Initiative: News." https://opensource.org/. Acessado em 7 nov. 2017.

2.5 FERRAMENTAS DE TESTES DE SOFTWARES

O universo dos testes de software abrange muitas partes da computação, para que sejam melhor executados, eles se dividem em métodos de testes e cada método apresenta sua documentação e recomendações de onde se deve aplicá-los. Por fim, envolvendo esses métodos existem ferramentas que auxiliam nos inúmeros tipos e métodos para se testar softwares. Segundo Sommerville (2007), uma ferramenta de teste tem como base, deter a intervenção humana nos resultados obtidos, assim podendo ocorrer o aumento da qualidade dos testes. As ferramentas usadas para auxiliar no processo de teste de um software, podem ser utilizadas por vários tipos de teste, seguindo o plano funcional ou estrutural. Algumas ferramentas serão descritas, essas foram escolhidas por critério de popularidade. Uma pesquisa na internet foi feita para selecionar três ferramentas de automação para testes de software, as ferramentas que apareceram em maior quantidade nos sites visitados foram escolhidas e descritas no presente trabalho.

Uma ferramenta que dá suporte ao teste de software se torna útil quando, visando uma maior agilidade nas atividades do processo de teste, a sua utilização de suporte ao teste pode contribuir para consideráveis ganhos de tempo (ELIZA, 2013).

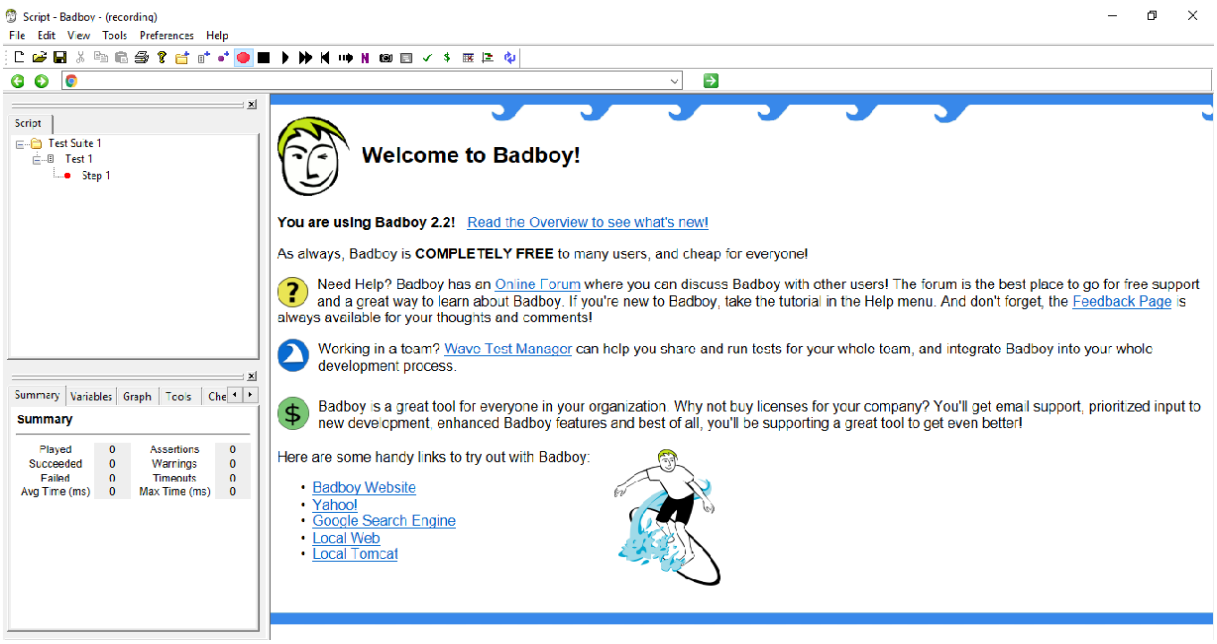
2.5.1 BAD BOY

A ferramenta de teste de software Bad Boy é um programa que auxilia na implementação de testes automatizados de software, por esse motivo foi selecionada, com a finalidade de ser avaliada pelo autor do presente trabalho. Além de ser uma das ferramentas mais citadas em artigos e muito utilizada por profissionais da área de testes em softwares, é destacada por vários *ranks* de qualidade das melhores ferramentas de testes automatizados de software, como o site de testes *riceconsulting* [4] que é um padrão de consulta e acesso pelos profissionais da área de informática. Em praticamente todas as páginas web que necessitam receber os dados do usuário, é utilizado um elemento HTML chamado

[4] "Rice Consulting Services, Inc.." http://www.riceconsulting.com/. Acessado em 4 out. 2017.

"*Form*". O "*Form HTML*" consiste em caixas de entrada de dados e um botão de submissão destes dados. O BadBoy trabalha com esse elemento HTML através do "*Form Populator*". Ele basicamente verifica se existe algum elemento *Form* na página e absorve todos os dados do formulário, trazendo a informação consigo para poder popular com dados. Não é necessário utilizar o *Form Populator* para a maioria dos scripts criados, uma vez que os dados já são transmitidos por parâmetros no *request* (requisição). A funcionalidade disponibilizada pelo *Form Populator* pode ser utilizada em casos que não são possíveis, de forma fácil, entrar com dados em um formulário. Porém o Bad Boy possui uma interface simples e opcionalidades bem visíveis ao usuário, conforme mostra a figura 1 :

Figura 1: Representação da tela inicial da ferramenta de teste BadBoy



Fonte: (DO AUTOR, 2018)

Além dos seus mais variados ramos de funções disponibilizadas ao usuário, o Bad Boy possui um navegador dentro do seu programa, que faz a interação com seus componentes de gravação. Informa Lima (2014) sobre o modo *request*, definido como padrão que captura as requisições enviadas pela aplicação ao servidor, ou seja, as chamadas às URLs e os envios de parâmetros via GET ou POST (métodos de envio). Uma grande vantagem deste modo de gravação é que ele não depende da interface gráfica da aplicação, outra seria que os scripts gravados neste modo podem ser utilizados para testes de performance.

Ressalta Lima (2014) que a ferramenta BadBoy dispõe de diversos relatórios de execução, que apresentam, entre outros, informações sobre o tempo das respostas às requisições, quantidade de falhas, quantidade de sucessos, testes executados e *Screenshots*. É capaz de repetir uma sequência de atividades de navegação, sucessivamente se torna um mecanismo muito útil para depurar e testar um site. Um identificador que é digitado deve ser exclusivo. A inserção do mesmo valor duas vezes gera um erro, mesmo assim é possível tratar esse erro no *script* que é gerado após salvar o teste. Isso ocorre, por exemplo, se houvesse a tentativa de criar um script que se registra para uma conta com um ID de usuário. A primeira vez que isso é executado a conclusão é bem-sucedida, mas na repetição tentando se inscrever para o mesmo ID novamente é gerado um erro. Mesmo com o teste gravado o script de um servidor (por exemplo, uma caixa de desenvolvimento local), torna-se necessário executá-lo em um servidor diferente, ou no mesmo servidor com tratamento a uma outra porta. Isso gera problemas de conexão o qual é necessário substituir o nome do host para onde os pedidos são direcionados. O Badboy ajuda a resolver esses tipos de problemas de várias maneiras: Editando parâmetros; Variáveis de Script e uma variedade de ferramentas para auxílio, como Pesquisar e Substituir.

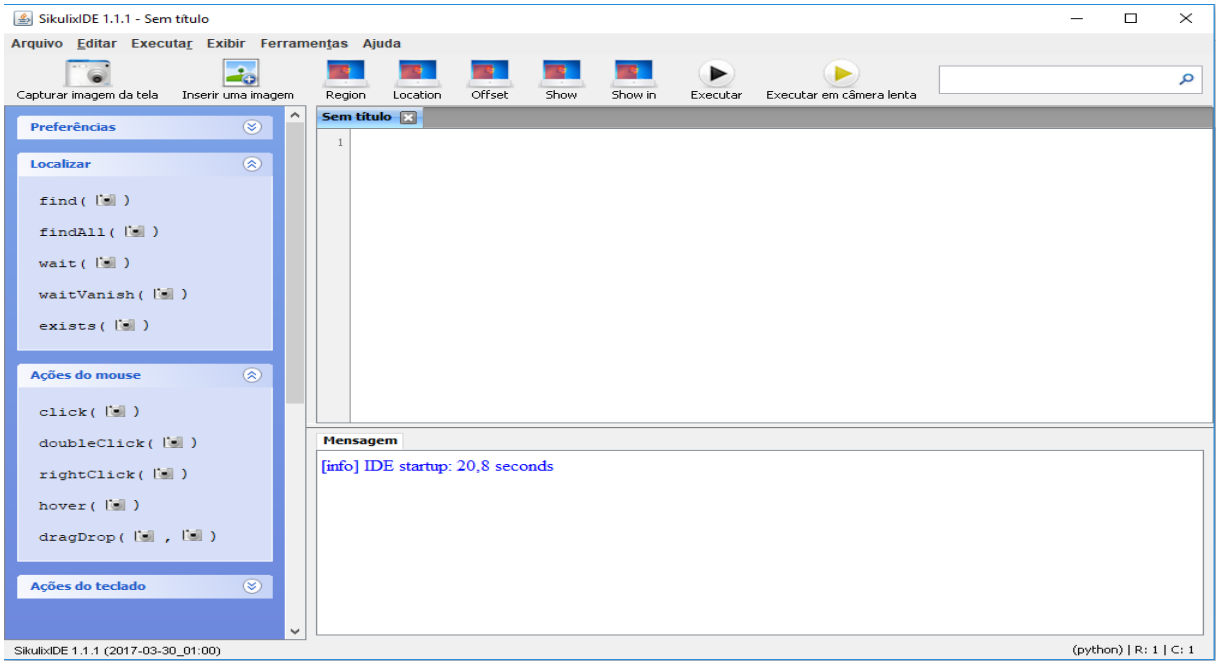
2.5.2 SIKULI IDE

A ferramenta baseia-se em reconhecimento de imagem para realizar ações na tela do computador do usuário, como clicar, mover o mouse e digitar, entre outras ações, podendo assim ser utilizada para testar software independente da interface utilizada. O propósito do Sikuli é poder testar qualquer aplicação que apresenta uma interface com o usuário, é baseado na linguagem Jython (Java + Python). A linguagem Python pode ser utilizada para a criação da biblioteca, uma vez que apresenta uma linha de aprendizagem extensa e pode ser integrada ao ambiente. O foco do programa é o teste automatizado de software, comparada com as demais ferramentas do mesmo seguimento, o Sikuli possui singularidades consideráveis, as quais, tornam a ferramenta uma das únicas disponíveis que trabalha com o conceito de passos de imagem. Seus planos de teste são criados em sequências de

screenshots (imagens referentes a captura de tela), para que quando for salvo o script de teste, seguir a ordem em que as imagens foram selecionadas.

Costa (2014) destaca que a ferramenta Sikuli automatiza qualquer ponto que está visível na tela do computador. Ele usa reconhecimento de imagem para controlar e identificar os componentes de interface, usando pontos específicos da tela do dispositivo em que a ferramenta está sendo executada. Isso torna a usabilidade inferior na ferramenta, no sentido de testar mais de uma aplicação ao mesmo tempo, pois só é permitido o uso de suas funções para pontos que são visíveis em tempo real na atividade do dispositivo, porém é permitido fazer com que passo a passo a ferramenta abra mais de uma aplicação e realize os testes, mas isto acarretaria em um script de teste muito grande e complexo de alterar. É muito útil quando o aceso ao código interno ou fonte de um programa, seja inviável. Sua interface é simples e intuitiva, com funcionalidades diretas e ícones que auxiliam na escolha do usuário, além da linguagem ser em português é possível alterá-la para: Inglês, alemão, espanhol, francês ou italiano. Além de possuir poucos botões de funcionalidades na tela inicial, é possível obter mais funções com os botões do seu menu, no canto superior esquerdo, e funções de ações de mouse e teclado no canto esquerdo e inferior esquerdo, conforme mostra a figura 2:

Figura 2: Representação da tela inicial da ferramenta de teste Sikuli



Fonte: (DO AUTOR, 2018)

É possível fazer a interação da ferramenta com uma linguagem de programação, basta saber se a linguagem possui uma biblioteca de integração com o Sikuli, programando pela sintaxe da linguagem é necessário declarar variáveis, que guardaram os caminhos onde serão salvos os scripts de teste. Feito essa conexão, o programador tem a liberdade de realizar todas as opcionalidades de automação oferecidas pela ferramenta, apenas com trechos de códigos. O Sikuli IDE edita e executa *scripts Sikuli*, integra captura de tela e um editor de texto personalizado (*Sikuli Pane*) para otimizar a escrita dos scripts. Para mostrar imagens embutidas no *Sikuli Pane*, todos os literais de *string* que terminam com ". Png" são substituídos por um objeto *JButton* personalizado (*ImageButton*). Se um usuário ajusta a similaridade padrão de imagem, um “Pattern()” é automaticamente criado no topo da imagem. Para executar um *script Sikuli*, o Sikuli IDE cria um interpretador “org.Python.util.PythonInterpreter” e automaticamente passa algumas linhas de cabeçalhos (por exemplo, para importar módulos Sikuli do Jython, e para definir o caminho para diretórios .Sikuli) para o interpretador. Uma vez que estes cabeçalhos são definidos, o script .py é simplesmente executado pelo “PythonInterpreter.execfile()”. A ferramenta oferece uma documentação suficiente para a sua utilização de maneira intermediaria, ela não ampara um usuário na questão de tratamentos de falhas da IDE. O suporte oferecido pela ferramenta Sikuli a respeito de interação com códigos, ou qualquer funcionalidade e dúvidas de funcionamento, podem ser consultados na sua documentação [5].

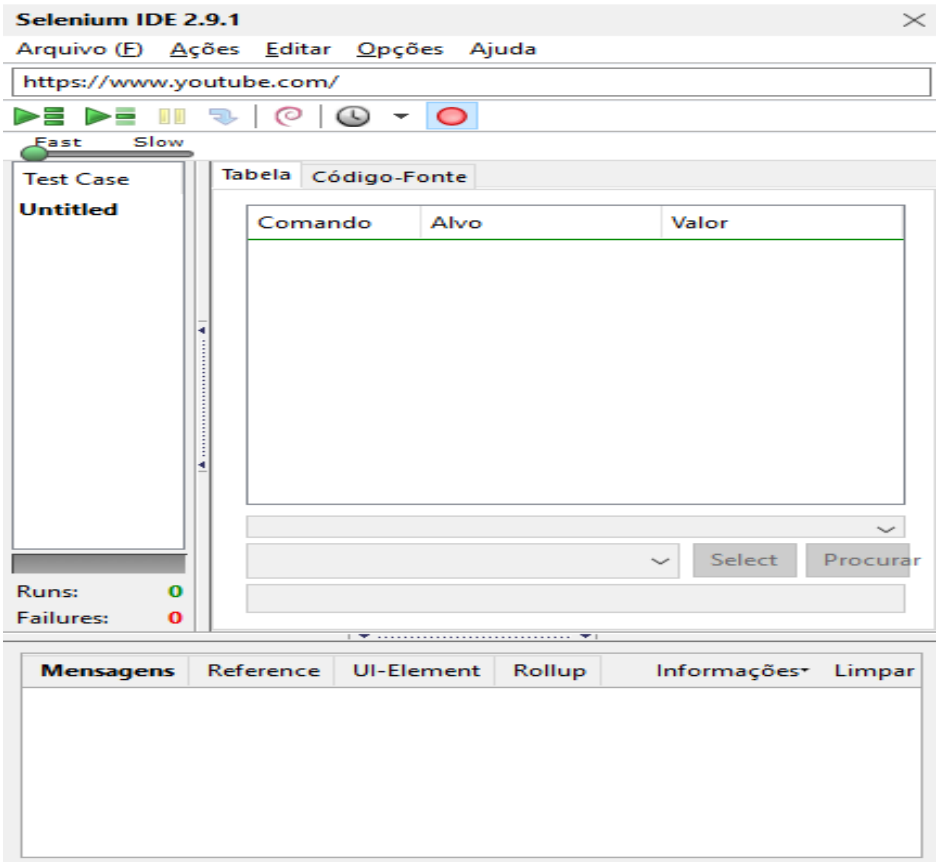
2.5.3 SELENIUM IDE

A ferramenta de teste de software Selenium IDE é baseada em métodos funcionais de teste, que são derivados de gravações na tela. A grande diferença do Selenium IDE com as demais ferramentas, é a sua instalação, pois ele é instalado como um *plugin* no navegador, diferente das outras ferramentas que são instaladas nas máquinas. Uma das grandes vantagens do Selenium é a possibilidade da realização de testes em qualquer navegador que tenha suporte ao JavaScript.

[5] "Sikuli / *Documentation* " https://sikulix-2014.readthedocs.io/.

A esse respeito, Nogueira (2014, p.1) declara: “Selenium IDE (*Integrated Development Environment*) é uma ferramenta utilizada para o desenvolvimento de scripts de teste com o Selenium através de um plugin para o Firefox[6] ”. O que torna o desenvolvimento dos scripts mais eficientes pelo método Record *and* Replay (Gravação e Execução) é a mobilidade que a ferramenta possui, por ser acoplada a um navegador. Existe somente o Selenium IDE para o Firefox, não existindo para o Internet Explorer e qualquer outro browser. Para uma futura execução dos scripts do Selenium IDE para outros browsers é necessário utilizar o Selenium Remote Control. Apesar de tantas funcionalidades, sua interface é simples, como mostra a figura 3:

Figura 3: Representação da tela inicial da ferramenta de teste Selenium



Fonte: (DO AUTOR, 2018)

[6] "Baixe o Firefox — Navegador web gratuito — Mozilla." https://www.mozilla.org/pt-BR/firefox/new/. Acessado em 5 out. 2017.

Rodrigues (2014), destaca a importância da ferramenta Selenium IDE para o desenvolvimento WEB, com HTML e JS, no seguinte trecho:

> O advento da Internet e o aumento expressivo na utilização e construção de softwares incentivam a uma constante procura por ferramentas, técnicas e metodologias adequadas para a garantia da qualidade. Como as tendências atuais têm mostrado que existe um movimento para automatizar testes para facilitar a realização de testes de regressão. Selenium é uma das mais populares suítes para automatizar testes. O Selenium foi desenvolvido de forma a apoiar e incentivar a automatização de testes funcionais de aplicações web em uma grande variedade de navegadores (browsers). O Selenium é *open source* e tornou-se uma das ferramentas mais aceitas no mercado atualmente. O Selenium não é apenas uma ferramenta e sim uma suíte de teste que engloba várias ferramentas para atender diferentes requisitos e ambientes de teste. O Selenium é constituído dos seguintes conjuntos de ferramentas: Selenium *Integrated Development Environment* , Selenium *Remote Control* , Selenium *WebDriver* e Selenium *Grid*. (RODRIGUES, p.6, 2014).

A IDE do Selenium permite que o caso de teste seja executado de diferentes maneiras como, por exemplo: rodar o caso de teste inteiro de uma só vez; rodar o caso de teste linha por linha; iniciar e parar a execução do caso de teste ou até mesmo rodar um único comando que você esteja desenvolvendo. Essa facilidade propiciada pela ferramenta ajuda muito o testador pois ele consegue verificar um problema sem dificuldades, além de não ser necessário aplicar de maneira codificada o caso de teste por completo quando não for solicitado pelo testador. Existe uma outra facilidade na execução dos casos de teste, em que é possível executar o mesmo caso de teste em domínios diferentes apenas mudando a URL no campo URL Base. Essa facilidade é muito útil quando já existe uma versão do site em produção e outra em desenvolvimento. Dessa forma aproveita-se o mesmo caso de teste para executar tanto na versão que está rodando quanto na versão que ainda está em construção. Além de ter a condição de varrer um script, o que dá ao usuário a liberdade de aplicar um teste no seu próprio código, usufruindo das mesmas opções de gravação. Se necessário, a IDE Selenium também pode abranger quase todas as áreas de testes funcionais, pelo fato de possuir extensões que complementam as funcionalidades da IDE, basta adicionar uma nova extensão que o Selenium terá acesso aos demais padrões de testes.

2.6 TRABALHOS RELACIONADOS

Neste item são apresentados dois trabalhos que tem relação com o tema proposto para esta pesquisa, ou seja, tem por objetivo testar ferramentas de testes de software. Os trabalhos são: “Uma avaliação de ferramentas para testes em sistemas de informação móveis baseada no método dmadv” e “ Análise comparativa de ferramentas de teste para aplicações em banco de dados ”.

O primeiro artigo analisado apresenta uma avaliação entre as principais ferramentas de teste para sistemas de informação móveis baseado na metodologia Six Sigma[7] utilizando o método DMADV[8], desenvolvida pelos autores: Ismaily Santos, Carla Bezerra, Gustavo Monteiro, Ítalo Araújo e Talisson Oliveira. O objetivo do trabalho é facilitar o aumento do giro de lançamento de novos produtos da empresa no mercado. Com o método DMADV e os aspectos divulgados no trabalho, possibilitou aos autores comparar determinadas ferramentas de testes para aplicações móveis. Essa comparação é importante pois fornece informações relevantes para identificar a ferramenta que melhor se adequa às exigências e necessidades de um determinado projeto de desenvolvimento ou em determinada empresa. É importante salientar que o uso da ferramenta certa pode facilitar a execução dos testes no projeto, proporcionando maior qualidade do sistema.

Usando um conjunto com vários critérios, sendo oito próprios para aplicações móveis, os autores avaliaram, ferramentas gratuitas de teste de sistemas de informações. Os resultados indicaram que as ferramentas que cumprem os critérios selecionados, para analisar as ferramentas foram o Robolectric, o Scirocco e o Robotium. “Contudo, a avaliação também revelou que boa parte dos critérios não são atendidos pelas ferramentas atuais e que somente uma combinação das ferramentas possibilita uma cobertura maior das necessidades dos testadores, uma limitação deste trabalho foi à não execução de todas as fases do método DMADV ” (SANTOS, BEZERRA, MONTEIRO, ARAÚJO, 2017, p.563). O estudo seguiu a prática de testes em aplicações móveis, utilizando o método DMADV, o qual, obteve o cumprimento de critérios pré-selecionados em 3 das ferramentas testadas. A

[7] "Six Sigma – abordagem desenvolvida para aumentar a qualidade e, consequentemente, a rentabilidade das empresas.

[8] "DMADV: Definir (Define), Medir (Measure), Analisar (Analyse), Desenhar (Design) e Verificar (Verify).

metodologia ágil DMADV antes da pesquisa ser aplicada, era pouco conhecida como técnica de teste, pois as aplicações móveis além de serem um marco recente na tecnologia, ainda não possuem tantos meios de se aplicar testes ou ferramentas existentes que auxiliam nesse caminho.

O segundo trabalho analisado teve como base de teste as ferramentas DB Stress[9] e JMeter[10], comparando-as. Conceitos básicos de teste foram apresentados e algumas ferramentas para teste em aplicações de banco de dados. A ferramenta DB Stress, segundo os testes feitos na pesquisa, pode ser indicada para empresas que trabalham com muitos dados, altos volumes de informação, e também serve para administradores de banco de dados, devido às vantagens de possuir vários mecanismos de configuração, simulação de teste com usuários em tarefas diferentes e também as funcionalidades que ajudam a analisar os resultados dos testes executados. Ao contrário da ferramenta JMeter que é uma aplicação de código aberto e também multiplataforma, com funcionalidades para testar aplicações na WEB. A ferramenta possui a desvantagem de avaliar os resultados apenas através de gráficos, diante disto pode-se indicar esta ferramenta aos desenvolvedores para realização de teste de carga, desempenho e stress, com o aumento da acurácia nos acertos o JMeter é uma das ferramentas com o maior número de downloads na página de programas do apache[11] (PIERAZO,RODRIGUES,SOARES, 2013, p.14).

O foco da pesquisa foi a aplicação e avaliação das ferramentas em bancos de dados, o que basicamente, simula testes de performance, apesar de os resultados serem satisfatórios para os pesquisadores, as dificuldades encontradas só foram supridas pela ferramenta DB Stress, pois a mesma possui funcionalidades pagas que serviram de auxílio na resolução dos erros encontrados.

---

[9] "DTM DB Stress, database stress testing software, product overview." http://www.sqledit.com/stress/software.html. Acessado em 6 out. 2017.

[10] "Apache JMeter - Apache JMeter™." http://jmeter.apache.org/. Acessado em 6 out. 2017.

[11] "Welcome to The Apache Software Foundation!." https://www.apache.org/. Acessado em 17 out. 2017.

3 METODOLOGIA

Neste item estão descritas as etapas da metodologia usada para o seguimento do presente trabalho:

- Modelagem da plataforma de testes
- Implementação da Plataforma
- Casos de Teste

3.1 MODELAGEM DA PLATAFORMA DE TESTES

Nesta abordagem, são criados esquemas lógicos de uma plataforma de testes. Segundo Sommerville (2011), uma plataforma de testes é conceituada como um modelo independente de plataforma, em princípio, gerando um programa de trabalho para servir de teste.

3.1.1 REQUISITOS FUNCIONAIS:

A listagem de requisitos funcionais formada por:

- Cadastrar usuários;
- Excluir usuários;
- Alterar usuários;
- Listar registros;
- Efetuar Login;
- Efetuar Log out;
- Busca Registros;
- Filtrar Registros;
- Imprimir;
- Download;

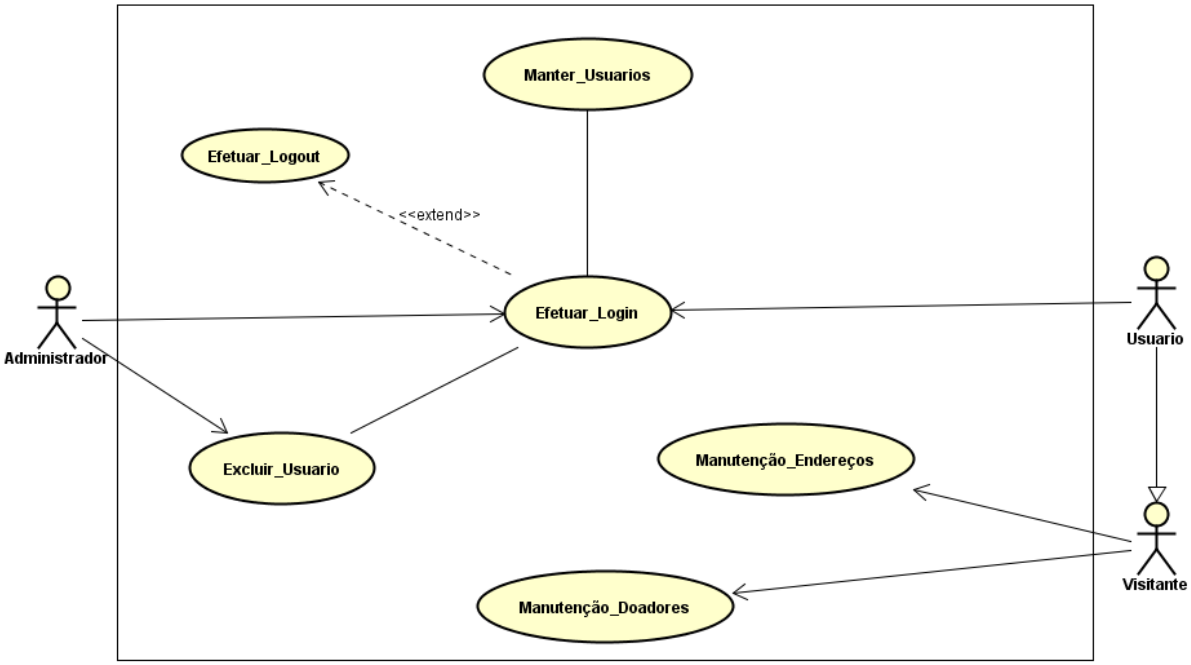
3.1.2 REQUISITOS NÃO FUNCIONAIS

- Aplicação para WEB;
- Persistir registros em um banco de dados e exibi-los na tela do sistema;
- Controle de permissões;

3.1.3 DIAGRAMA DE CASO DE USO

O diagrama de casos de uso tem o objetivo de documentar o que o sistema propõe ao ponto de vista do usuário. “Ele descreve as principais funcionalidades do sistema e a interação dessas funcionalidades com os usuários do mesmo sistema, porém, não nos aprofundamos em detalhes técnicos que dizem como o sistema faz” (RIBEIRO, 2016, p.1). A figura 5 ilustra os casos de uso da plataforma de teste do sistema WEB desenvolvido.

Figura 4: Diagrama de caso de uso da Plataforma de Teste



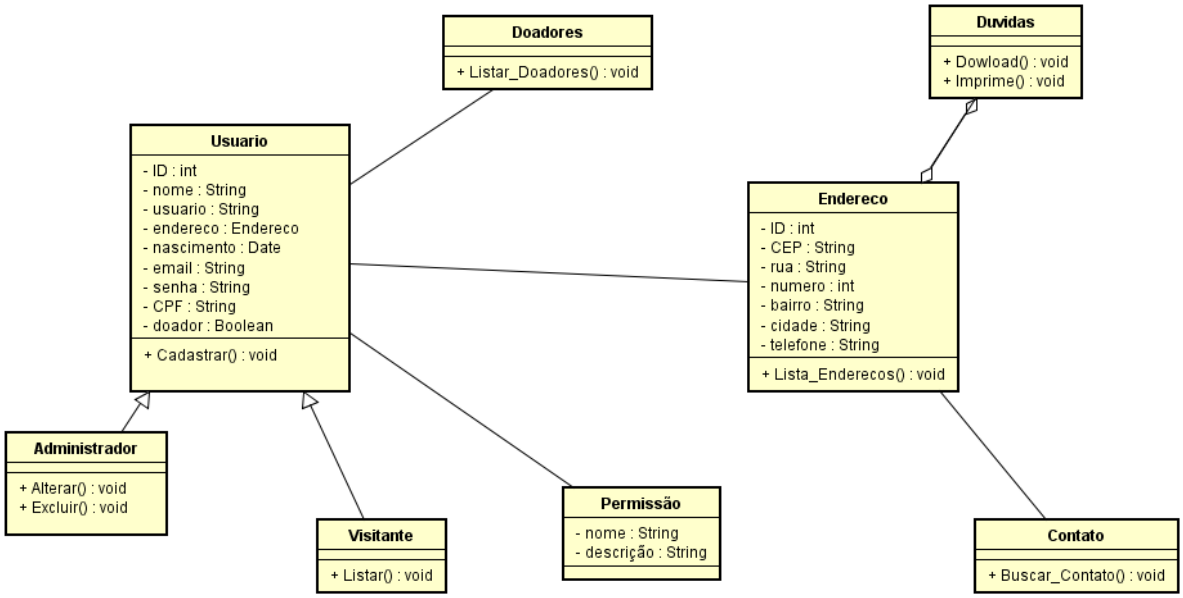
Fonte: (DO AUTOR, 2018)

Com base na figura 5, o diagrama de casos de uso apresenta 3 atores, como Administrador, Usuário e Visitante. Logo o ator Administrador tem acesso a todos os casos que necessitam de autenticação.

3.1.4 DIAGRAMA DE CLASSE

Um diagrama de classe descreve o objeto e insere informações de estruturas usadas pelo seu sistema. Para Faria (2015) esse tipo de diagrama vai agir internamente, para facilitar a comunicação com seus usuários. O diagrama apresentado na figura 4 representa as classes especificadas para a plataforma de testes.

Figura 5: Diagrama de classes da Plataforma de Testes



Fonte: (DO AUTOR, 2018)

Com base na figura 4 o diagrama de classes apresenta uma classe de usuário e endereço, que são as principais funcionalidades do sistema. A classe usuário possui um filho que é a classe Administrador, a classe Doadores é uma classe abstrata, que é implementada pela classe Usuário.

3.1.5 DESCRIÇÃO DOS PRINCIPAIS CASOS DE USO

3.1.5.1 CASO DE USO EXCLUIR USUÁRIO

Nome do Caso de Uso	Excluir Usuário
Ator Principal	Administrador
Atores Secundários	
Resumo	Este caso de uso caracteriza a ação de excluir um usuário do sistema
Pré-condições	Estar logado como administrador
Pós-condições	ID válido
Fluxo Principal	
Ações do ator	Ações do sistema
Selecionar o usuário	verificar a exclusão
Restrições/Validações/Regras de Negócio	1. O usuário deve estar cadastrado no sistema / 2. O ID deve ser válido.

3.1.5.2 CASO DE USO CADASTRAR USUÁRIO

Nome do Caso de Uso	Cadastrar Usuário
Ator Principal	Administrador
Atores Secundários	Usuário
Resumo	Este caso de uso caracteriza a ação de cadastrar um usuário no sistema
Pré-condições	Autenticação no sistema
Pós-condições	Persistir com sucesso
Fluxo Principal	
Ações do ator	Ações do sistema
Clicar no botão “novo”	Proceder a permissão do usuário
	Permitir ou Restringir a abertura do formulário de cadastro, dependendo da condição.
Preencher formulário de cadastro	
Restrições/Validações/Regras de Negócio	1. O usuário deve estar logado com conta cadastrada no banco de dados.
	2. Os campos do formulário devem ser válidos .

Estrutura de Dados
1. Nome de usuário
2. Usuário
3. Senha

3.1.5.3 CASO DE USO ALTERAR USUÁRIO

Nome do Caso de Uso	Alterar Usuário
Ator Principal	Administrador
Atores Secundários	Usuário
Resumo	Este caso de uso caracteriza a ação de alterar um usuário no sistema
Pré-condições	Estar logado como administrador
Pós-condições	Persistir com sucesso
Fluxo Principal	
Ações do ator	Ações do sistema
Clicar no botão “alterar”	Proceder a permissão do usuário
	Permitir ou Restringir a abertura do formulário de edição, dependendo da condição.
Alteração dos dados	
Restrições/Validações/Regras de Negócio	1. O usuário deve estar logado como administrador
	2. Os campos do formulário devem ser validados antes de alterados
Estrutura de Dados	
1. Usuário	
2. Senha	

3.1.5.4 CASO DE USO BUSCAR CEP

Nome do Caso de Uso	Busca CEP
Ator Principal	Administrador
Atores Secundários	Usuário, Visitantes
Resumo	Este caso de uso caracteriza a ação de procurar endereços a partir do atributo CEP
Pré-condições	Acessar a aba “Endereços”
Fluxo Principal	
Ações do ator	Ações do sistema
Escrever um CEP	Buscar API
	Procurar no Banco de dados e retornar
Restrições/Validações/Regras de Negócio	1. O usuário deve estar na aba “Endereços”
	2. Os campos a serem inseridos devem conter números de CPF válidos.
Estrutura de Dados	
1. CEP do endereço	

3.1.5.5 CASO DE USO LISTAR DOADORES

Nome do Caso de Uso	Listar Doadores
Ator Principal	Administrador
Atores Secundários	Usuário, Visitantes
Resumo	Este caso de uso caracteriza a ação de listar os usuários que são doadores
Pré-condições	Estar na aba “Doadores”
Fluxo Principal	
Ações do ator	Ações do sistema
Clicar na aba Doadores	Retornar a lista dos usuários que possuem o atributo doadores = TRUE
Restrições/Validações/Regras de Negócio	1. O usuário deve estar cadastrado na base de dados
	2. O atributo doador deve ser = “TRUE”

3.1.5.6 CASO DE USO EFETUAR LOGIN

Nome do Caso de Uso	Efetuar Login
Ator Principal	Administrador
Atores Secundários	Usuário
Resumo	Este caso de uso caracteriza a ação de o usuário se autenticar no sistema
Pré-condições	Estar cadastrado na base de dados

Pós-condições	Persistir com sucesso, erro.
Fluxo Principal	
Ações do ator	Ações do sistema
Inserir Usuário e Senha	Proceder os parâmetros com os registros da base de dados
	Permitir ou Restringir a autenticação
Disponibilização da aba “Usuários”	
Restrições/Validações/Regras de Negócio	1. O usuário deve estar cadastrado na base de dados
	2. Os campos devem ser iguais os do registro
Estrutura de Dados	
1. Usuário	
2. Senha	

3.1.5.7 CASO DE USO EFETUAR LOGOUT

Nome do Caso de Uso	Efetuar Loggout
Ator Principal	Administrador
Atores Secundários	Usuário
Resumo	Este caso de uso caracteriza a ação de o usuário efetuar a saída do sistema
Pré-condições	Estar autenticado no sistema

Pós-condições	Estar desautenticado do sistema
Fluxo Principal	
Ações do ator	Ações do sistema
Clicar em fazer Loggout	Proceder os parâmetros com os registros da base de dados
	Permitir o Loggout
Restrições/Validações/Regras de Negócio	1. O usuário deve estar logado no sistema
Estrutura de Dados	

3.2 IMPLEMENTAÇÃO DA PLATAFORMA DE TESTES

Faria (2015) define aplicações WEB como um serviço independente de hardware, pois quando o autor fala em aplicação WEB, está se referindo a sites ou sistemas onde parte do desenvolvimento fica hospedado em um servidor na nuvem, e o cliente basicamente não necessita ter instalado em seu dispositivo para utilizá-las, além de um navegador (browser). O sistema desenvolvido como ambiente de testes é um portal de notícias e cadastro de usuários. Este apresenta interações com recursos de textos, imagens, vídeos, download, segurança de informação e interação com API.

A implementação foi feita através da IDE NetBeans executada em ambiente Windows. Java foi a linguagem utilizada para o desenvolvimento da camada de modelo do sistema, que dentro das especificações JavaEE, foi implantado o componente JavaServerFaces (JSF) que foi usado para desenvolver a parte Web. É a linguagem Java que proporciona toda a estrutura das páginas criadas, que dentro da arquitetura de classes, constitui a camada de modelo. Esta camada é responsável por criar a parte de *Back-end*, permitindo a interação com os métodos criados futuramente. Toda a estrutura, como menu, campos de entrada de texto, hyperlinks, imagens e botões são gerados pelo framework PrimeFaces. A partir

desta estrutura, somada à estilização CSS e ao Java JSF, torna-se possível apresentar um site organizado e com boa usabilidade. A linguagem XHTML foi usada para a criação de scripts do lado do servidor. Estes scripts são responsáveis pela realização de muitas tarefas, tais como: interação com os métodos de inserir, alterar e excluir dados; conexão com a base de dados; processamento de requisições; validação de dados.

O conteúdo é replicado inicialmente em cinco páginas: página Início, Doadores, Dúvidas, Contato e Usuário, após o usuário concluir a autenticação é disponibilizada uma sexta página que permite alterações no cadastro, dependendo de sua permissão. Para a segurança de informação foi implantado permissões aos usuários, que são conjuntos de funcionalidades atribuídas a um determinado tipo de cadastro. O controle dos registros é permitido totalmente ao Sistema Gerenciador de banco de dados PostgreSQL, que por sua vez está hospedado na máquina do autor do trabalho.

3.3 CASOS DE TESTE DE SOFTWARE

Como todo teste de software é uma projeção de possíveis falhas que o sistema possa apresentar, após sua implementação, o mesmo busca revelar falhas que antes não eram vistas. Antes de uma aplicação de testes, tanto automatizados quanto manuais, o testador possui um objetivo final, o qual, busca alcançar com o teste e os possíveis caminhos que deverá ser percorrido até atingir a finalidade. Segundo Sommerville (2011) a documentação deste teste é classificada como caso de teste, que permite aos testadores simular todas as rotas que deveram ser seguidas e implantadas para atingir o objetivo final do teste.

O caso de teste pode ser escrito em forma de documento texto, onde o testador fará as classificações dos passos a serem seguidos, declarando os valores e nomes de variáveis que possivelmente serão transferidas no teste. O caso de teste não necessariamente segue um padrão, porém as boas práticas de testes de software sugerem que seja construído após a descrição dos casos de uso e então seguir esse modelo para cada caso, aproveitando as mesmas nomenclaturas utilizadas nos casos de uso. Conforme Sommerville (2011, p.27), o teste das

hipóteses pode envolver o rastreio manual do código do programa, bem como exigir novos casos de teste para localização do problema.

CASO DE TESTE ERRO LOGIN
Objetivo do teste: Responsável por garantir que apenas usuários autenticados tenham acesso a aba usuários do sistema. **Pré-condições:** Estar cadastrado na base de dados. **Prioridade**: Alta **Tipo de execução**: Automatizado
Passo 1: Acessar página de login disponível no link: https://localhost:8181/MeuTCC-Web/login.xhtml **Resultados Esperados**: O sistema mostra a página de login.
Passo 2: O ator informa um usuário e senha aleatórios que não constam na base de dados e clica em "efetuar login". **Resultados Esperados**: O sistema deve informar ao usuário que os campos inseridos estão inválidos, retornando para a ferramenta o status de "login failed".

CASO DE TESTE SUCESSO LOGIN
Objetivo do teste: Responsável por garantir que os usuários autenticados tenham acesso a aba usuários do sistema. **Pré-condições:** Estar cadastrado na base de dados. **Prioridade**: Alta

Tipo de execução: Automatizado

Passo 1:
Acessar página de login disponível no link: https://localhost:8181/MeuTCC-Web/login.xhtml

Resultados Esperados:
O sistema mostra a página de login.

Passo 2:
O ator informa um usuário e senha que estão cadastrados na base de dados e clica em "efetuar login".

Resultados Esperados:
O sistema deve informar ao usuário que seu login foi efetuado com sucesso e disponibilizar visivelmente a aba “usuários”.

CASO DE TESTE LOGOUT

Objetivo do teste:
Responsável por garantir que os usuários sejam desautenticados do sistema.

Pré-condições:
Estar Logado.

Prioridade: Baixa

Tipo de execução: Automatizado

Passo 1:
Clicar no botão referente a sessão onde consta o atributo “nome” de quem está autenticado, e clicar em loggout.

Resultados Esperados:
O sistema deve desautenticar o usuário, movendo-o para a tela de login.

CASO DE TESTE MANUTENÇÃO ENDEREÇOS

Objetivo do teste:

Responsável por garantir que os endereços cadastrados no sistema, sejam apresentados de maneira correta, possibilitando ao usuário realizar uma busca pelo nome da Cidade do endereço.

Pré-condições:

Ser Visitante, Usuário ou Administrador do sistema

Prioridade: Média

Tipo de execução: Automatizado

Passo 1:

Acessar página de Endereços disponível no link: https://localhost:8181/MeuTCC-Web/enderecos.xhtml

Resultados Esperados:

O sistema mostra a página de endereços.

Passo 2:

O ator insere um valor inválido no campo “filtro” (nome da Cidade de um endereço que não está cadastrado base de dados).

Resultados Esperados:

O sistema deve apresentar um erro de “não encontrado”.

Passo 2.2:

O ator insere um valor válido no campo “filtro” (nome da Cidade de um endereço registrado na base de dados).

Resultados Esperados:

O sistema utilizando a tecnologia ajax, deve apresentar no mesmo momento em que os primeiros caracteres forem comparados com os registros, a lista dos endereços que possuem o mesmo nome que está sendo digitado.

CASO DE TESTE BUSCAR NOME
Objetivo do teste: Responsável por garantir que os usuários cadastrados no sistema, sejam encontrados de maneira correta, possibilitando realizar uma busca pelo atributo “nome” que estão cadastrados na base de dados. **Pré-condições:** Ser Visitante, Usuário ou Administrador do sistema **Prioridade**: Média **Tipo de execução**: Automatizado
Passo 1: Acessar página de Doadores disponível no link: https://localhost:8181/MeuTCC-Web/doadores.xhtml **Resultados Esperados**: O sistema mostra a página de doadores.
Passo 2: O ator insere um valor válido no campo “Buscar nome” (nome de um usuário registrado na base de dados). **Resultados Esperados**: O sistema utilizando a tecnologia ajax, deve apresentar no mesmo momento em que os primeiros caracteres forem comparados com os registros, a lista dos usuários que possuem o mesmo nome que está sendo digitado.

CASO DE TESTE ORDENAR REGISTROS
Objetivo do teste: Responsável por garantir que os usuários possam ordenar os cadastros do sistema da maneira acessível para a visualização dos registros. **Pré-condições:** Ser Visitante, Usuário ou Administrador do sistema **Prioridade**: Média **Tipo de execução**: Automatizado

Passo 1: No campo “Listagem” inserir o número 2 (o número corresponde a quantidade de registros que o usuário deseja visualizar em sua tela)
Passo 2: O usuário clica no botão “Filtrar” **Resultados Esperados**: Os registros devem ser exibidos na quantidade declarada no campo “Listagem”.
Passo 3: O usuário deve clicar nas teclas de posicionamento (posicionadas na parte inferior da tabela de registros) , para listar os registros, independentemente da ordem. **Resultados Esperados**: Os registros devem ser exibidos na ordem prescrita.

CASO DE TESTE LISTAR DOADORES
Objetivo do teste: Responsável por apresentar uma lista dos Usuários cadastrados no sistema, cujo possuem a condição de Doadores de sangue. **Pré-condições:** Ser Usuário do sistema **Prioridade**: Alta **Tipo de execução**: Automatizado
Passo 1: Acessar página de Doadores disponível no link : https://localhost:8181/MeuTCC-Web/doadores.xhtml **Resultados Esperados**: Os registros devem ser listados, mas apenas dos usuários que possuem o atributo “doador = TRUE”.

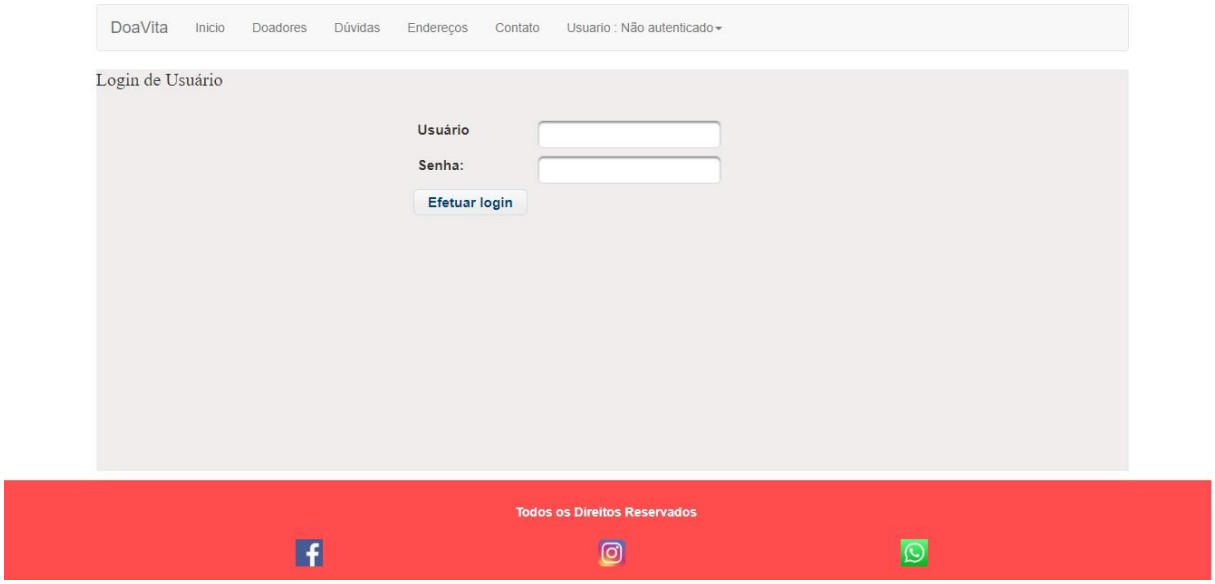
CASO DE TESTE IMPRIMIR
Objetivo do teste: Garantir que o sistema irá concluir com sucesso o processo de impressão, pré-programado a partir de um método que interage com um dispositivo. **Pré-condições:** Ser Visitante, Usuário ou ADM do sistema **Prioridade**: Média **Tipo de execução**: Automatizado
Passo 1: Acessar página de Dúvidas disponível no link: https://localhost:8181/MeuTCC-Web/duvidas.xhtml **Resultados Esperados**: A página “Dúvidas” deve ser carregada com sucesso
Passo 2: Clicar no botão “Imprimir Calendário” **Resultados Esperados**: A configuração de impressão do Windows deve ser aberta
Passo 3: Clicar em “imprimir” **Resultados Esperados**: A impressão deve obter sucesso, após sair da impressora uma folha e nela impressa o calendário do sistema.

CASO DE TESTE CRUD USUÁRIOS
Objetivo do teste: Garantir que os registros do sistema possam ser persistidos, alterados e excluídos com sucesso. **Pré-condições:** Ser Usuário e Administrador do sistema. **Prioridade**: Alta **Tipo de execução**: Automatizado
Passo 1: Logar no sistema. **Resultados Esperados**: O login deve ser efetuado com sucesso
Passo 2: Clicar em “Cadastros” , “usuários” **Resultados Esperados**: As listagens de usuários com os botões de edição devem aparecer.
Passo 3: Clicar em “+ Novo” **Resultados Esperados**: O formulário de cadastros deve ser aberto ao usuário
Passo 4: Clicar em “Alterar” **Resultados Esperados**: O formulário de edição de cadastros deve ser aberto ao usuário
Passo 5: Clicar em “Excluir” **Resultados Esperados**: O usuário deve ser excluído da base de dados

4 RESULTADOS

Neste capítulo são apresentados os resultados obtidos através da aplicação da metodologia proposta no Capítulo anterior. Os testes foram aplicados na plataforma de teste. Os casos de teste que envolvem os componentes de login são aplicados na página login.xhtml. A Figura 6 apresenta a página de login.

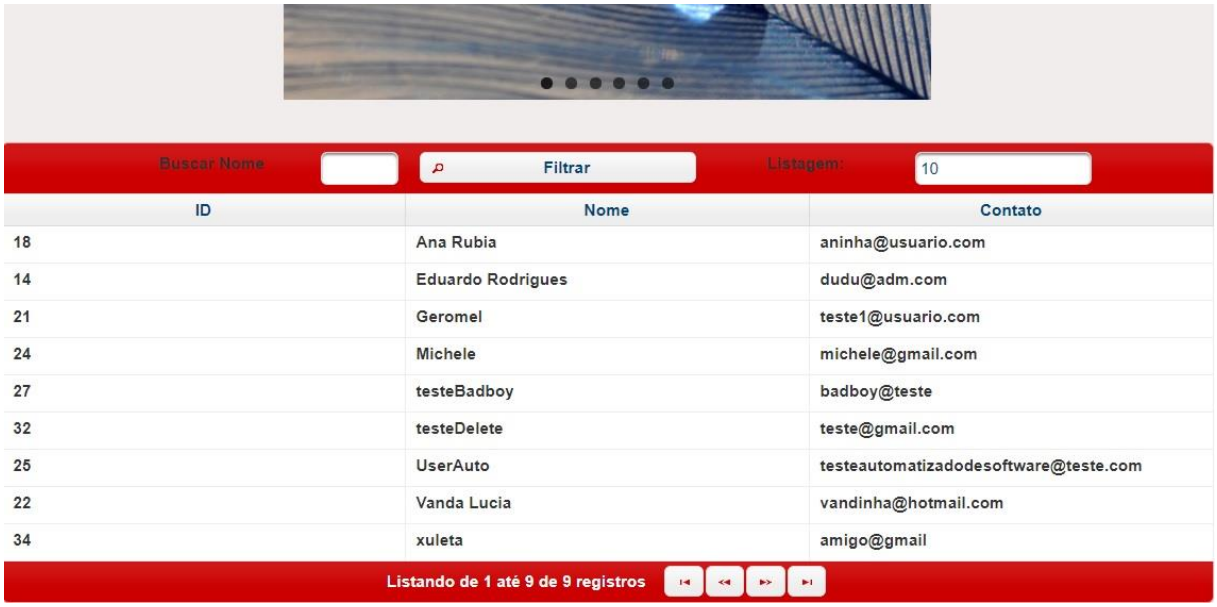
Figura 6: Tela de Login



Fonte: (DO AUTOR, 2018)

Os casos de teste que envolvem listagem, busca, filtro e paginação, são aplicados em formulários xhtml. A Figura 7 apresenta o formulário xhtml referente aos doadores.

Figura 7: Formulário JSF



Fonte: (DO AUTOR, 2018)

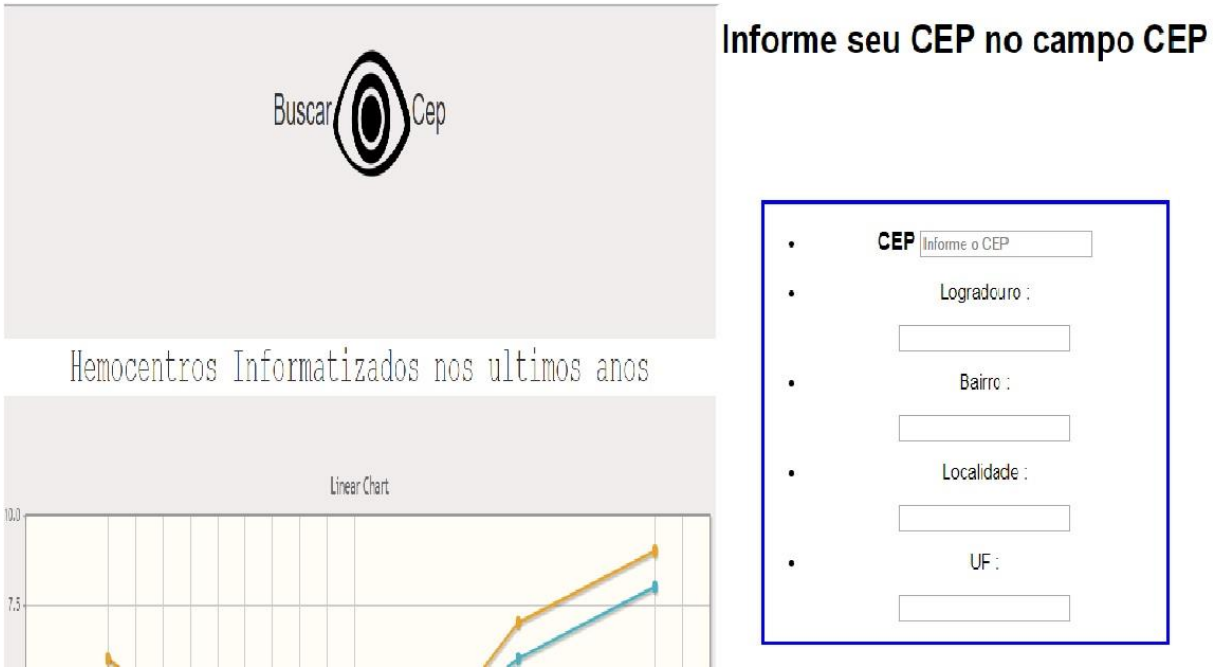
O caso de teste que referencia um teste de conexão é aplicado na página duvidas.xhtml. A Figura 8 apresenta o botão *Download* implementado na página duvidas.xhtml.

Figura 8: Botão JSF

Fonte: (DO AUTOR, 2018)

O caso de teste busca CEP foi aplicado na página enderecos.xhtml. A Figura 9 apresenta o botão e o formulário referente à busca do cep na página endereços.

Figura 9: Pagina Endereços, botão BuscaCep



Fonte: (DO AUTOR, 2018)

Os resultados das aplicações dos testes foram divididos nas sessões de: funcionalidades (Seção 4.1), tratamento de código (Seção 4.2), tempos de execução (Seção 4.3) e documentação (Seção 4.4). Para a elaboração dos testes foi utilizado o seguinte ambiente de trabalho:

Hardware: Processador Intel® Core™ I5-6200U CPU @ 2.30GHz 2.40GHz e memória de 8,00 GB. Software: Sistema Operacional Windows 10; Navegadores: Google Chrome 66.0.3, Mozila Firefox 43.0.1, Microsoft Edge 41.16 e navegador acoplado a ferramenta Badboy (Internet Explorer). A demonstração dos resultados obtidos nas seções 4.1, 4.2 e 4.3 estão atrelados a uma tabela referente à cada seção. Cada resultado descrito nas linhas das tabelas, referência uma ferramenta. Aos resultados obtidos em cada linha, foi atribuído um status. Os status servem para referenciar o resultado final do teste, neste caso foi atribuído as seguintes legendas apresentadas na tabela 1, que foi criada com base nos padrões de documentações de testes de software.

Tabela 1: Legenda para os resultados dos testes

SUCESS	Teste concluído sem erro
OK	Teste concluído, mas com algum tipo de erro
FAILED	Teste não concluído

Fonte: (DO AUTOR, 2018)

A última seção avaliada (seção 4.4) é referente aos manuais e toda a documentação abrangente oferecida pelos desenvolvedores de cada uma das ferramentas automatizadas. Através da aba “ajuda” que se encontra em todas as ferramentas descritas no presente trabalho, tornou-se viável a análise referente a gama de conteúdo que auxilia um testador ao usar uma das fermentas.

Com objetivo de registrar a aplicação dos testes, foram gravados vídeos introdutórios de cada ferramenta, utilizando todos os casos de testes citados no presente trabalho e a plataforma de testes desenvolvida. Todos os vídeos estão em disponíveis no canal Eduardo Rodrigues na plataforma de compartilhamento de vídeos Youtube.

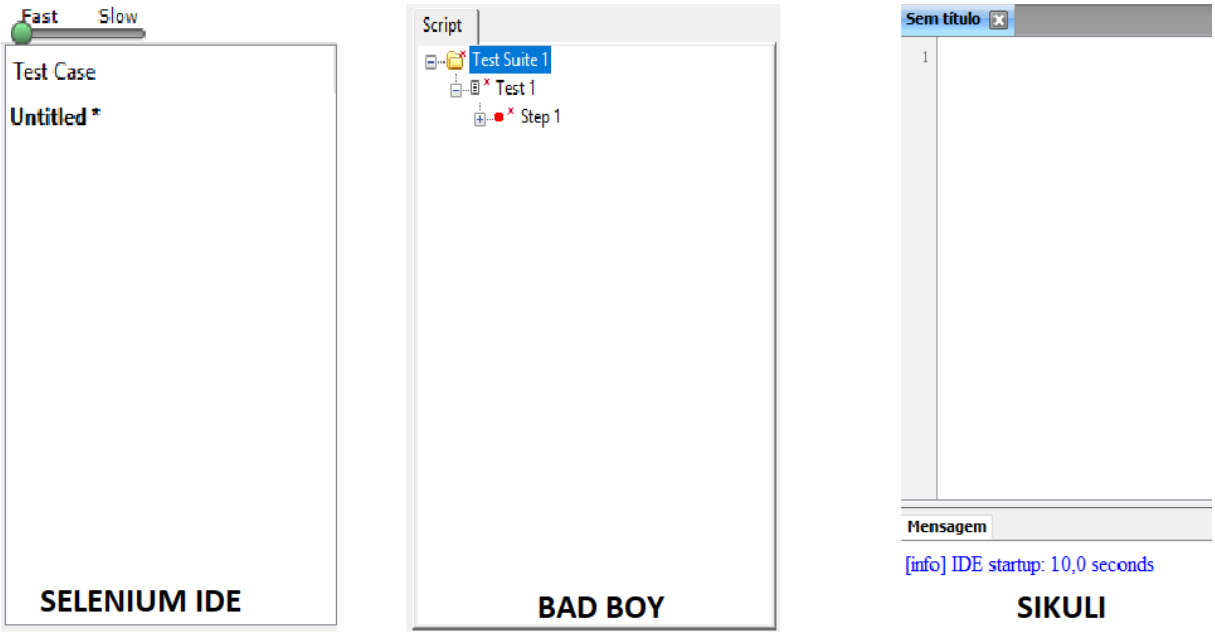
4.1 FUNCIONALIDADES

Nessa seção são avaliadas as funcionalidades fornecidas pelas ferramentas. As funcionalidades de gravação de script, disponibilizadas pelas ferramentas de teste: BadBoy e Selenium IDE, seguem o mesmo padrão de *Record and play*, com os seguintes botões em comum: Play (Dispara o script na ordem em que está salvo no *test suíte*), Play All (Dispara todo o script salvo no *test suíte,* independente da ordem em que foi especificado), Record (Inicia ou pausa a gravação da tela) e o botão Time (programa um tempo para a execução do script de teste). Diferente da ferramenta Sikuli que usa reconhecimento de imagem para encontrar os elementos

da interface gráfica de um sistema, além de utilizar uma biblioteca personalizada com métodos confeccionados para automatizar a sumarização dos resultados obtidos através dos testes e suas evidências.

A funcionalidade que a interface da ferramenta apresenta, cuja as três têm em comum é o *test suíte,* onde ficam os passos que o script de teste foi gravado. Na figura 10, podemos visualizar essa funcionalidade em comum entre as ferramentas Selenium, Badboy e Sikuli (respectivamente).

Figura 10: Cases de Testes das ferramentas



Fonte: (DO AUTOR, 2018)

Seguindo os casos de uso: Buscar CEP (Seção 3.1.5.4), Listar Doadores (Seção 3.1.5.5), Efetuar Login (Seção 3.1.5.6) e Efetuar Loggout (Seção 3.1.5.7) foram aplicados os testes especificados no caso de teste erro login, sucesso login, teste loggout e listar doadores. A tabela 2 apresenta os resultados de cada uma das ferramentas em relação aos casos de testes citados anteriormente.

Tabela 2: Casos Login e Listagem

	Erro Login	Sucesso Login	Teste Loggout	Listar Doadores
BadBoy	SUCESS	SUCESS	SUCESS	FAILED
Selenium	SUCESS	SUCESS	SUCESS	SUCESS
Sikuli	OK	OK	SUCESS	OK

Fonte: (DO AUTOR, 2018)

Após a aplicação dos casos de teste, as três ferramentas concluíram, sem nem uma falha, apenas o caso de teste Loggout. A ferramenta Selenium IDE concluiu todos os casos de teste sem nem uma falha. A ferramenta Sikuli finalizou todos os casos de teste, porém com algum tipo de erro, cujos foram resolvidos utilizando de suas próprias funcionalidades. A ferramenta BadBoy obteve sucesso nos casos de teste ErroLogin, SucessoLogin e Teste Loggout, porém falhou ao automatizar o caso ListarDoadores. Os erros listados na tabela 2 estão detalhados na sequência:

Na execução com a ferramenta Sikuli, no caso de teste “ERRO LOGIN”, ao procurar um elemento com a *tag* “<select>” que corresponde a página “fazer login”, a automação pela parte gráfica da ferramenta foi insuficiente para suprir essa necessidade. A solução do erro foi dada através da funcionalidade “find”, que após ser implementada como um passo adicional no teste, possibilitou a finalização do mesmo. No caso de teste “SUCESSO LOGIN”, ao inserir um valor no campo usuário, o cache do navegador apresenta as sugestões de auto completar, isso oculta o campo da senha, ocasionando o erro no script de teste, por não achar o elemento. A solução foi dada pela própria ferramenta ao inserir uma funcionalidade de “*findAll*”, onde o elemento é referenciado antes do cache do navegador ocultar o campo desejado. No caso de teste “LISTAR DOADORES”, ocorreu um erro no parâmetro da função “*type*”, cuja espera um valor “*String*” e recebe por padrão um

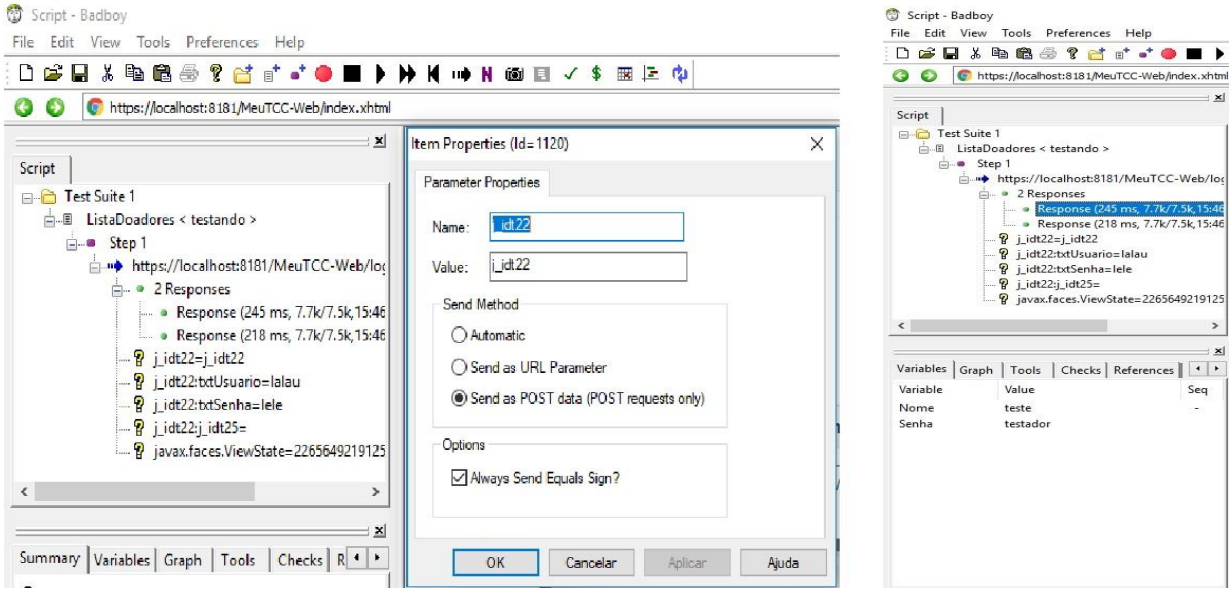
inteiro. A solução foi alterar o código do teste escrito na ferramenta, passando um parâmetro de texto.

A ferramenta BadBoy, ao executar o caso de teste “LISTAR DOADORES” apresentou erro ao inserir um valor no campo de filtragem, referente ao nome do doador. A ferramenta não reconheceu a funcionalidade que foi implementada com a tecnologia Ajax, e não ofereceu recursos funcionais para tratar esse erro. O teste não foi completado.

4.2 TRATAMENTO DE CÓDIGO

Nesta seção são avaliados os atributos gráficos de cada ferramenta, referente ao tratamento de um bloco automatizado manipulado diretamente no código. As ferramentas Selenium IDE e Badboy, possuem esse recurso localizado no canto superior esquerdo da tela, onde ficam seus "*Tests Suits*", na ferramenta Sikuli o mesmo se encontra na parte central. Nas três ferramentas o script que é gerado ao longo dos testes vai sendo replicado por ordem de criação. Para alterar determinada parte do script basta expandi-lo, então, cada uma das ferramentas gera o seu próprio tratamento de caso. Podemos visualizar nas figuras 11, 12 e 13 os atributos gráficos de cada ferramenta, que referenciam o código do teste.

Figura 11: Código Teste BadBoy

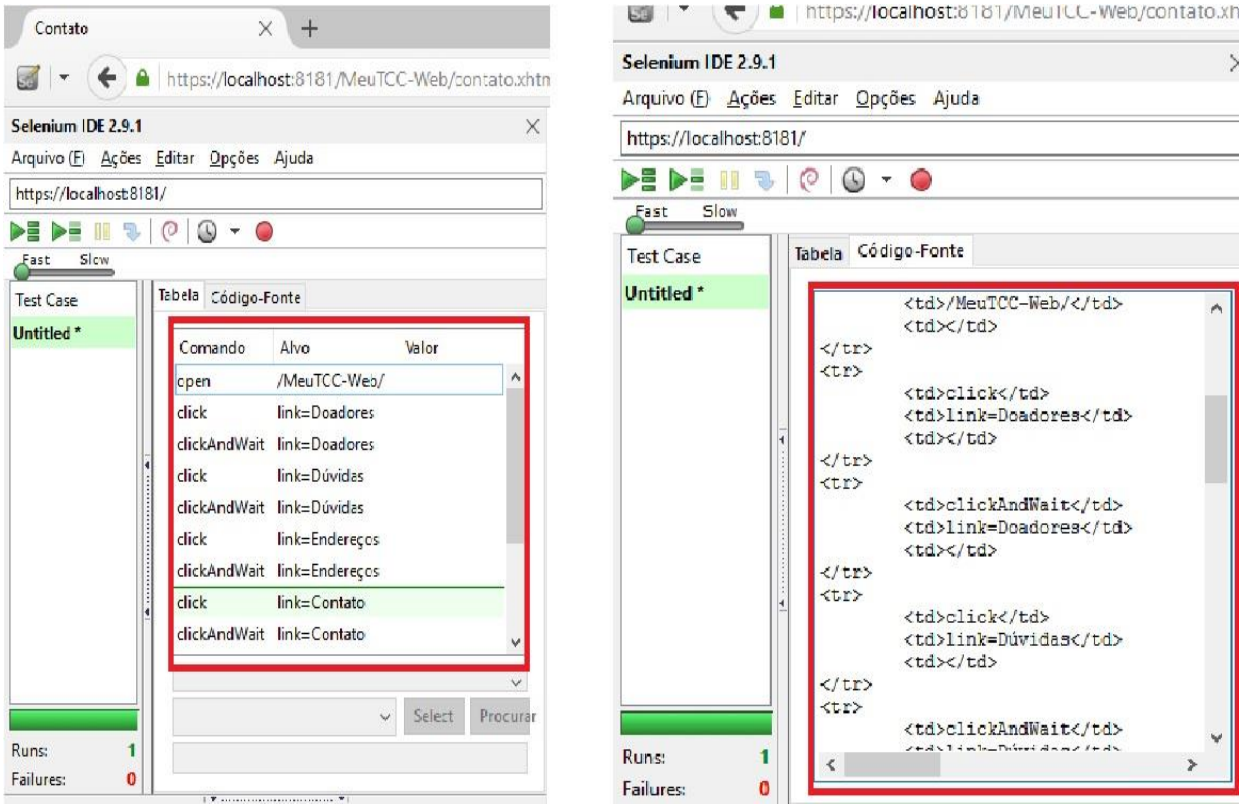


Fonte: (DO AUTOR, 2018)

A figura 11 mostra os métodos de tratamento do código gerado ao finalizar cada passo de um caso de teste, pela ferramenta BadBoy. Através da guia "*Item Properties*" é disponibilizado ao usuário a alteração ou inserção de uma nova propriedade, declarando um nome e um valor, incluindo o método de envio por GET, POST ou Automático. Na guia "*Variables*" é possível criar variáveis e declarar valores a elas, após, podem ser usadas no caso de teste como um valor referenciado. A ferramenta Badboy conta com a funcionalidade JScript, que insere no *Test Suite* um código JavaScript pronto, ou se o usuário preferir, é possível escrever seu próprio código JScript e adicionar a execução do teste.

A figura 12 apresenta as abas de tratamentos do código de teste gerado pela ferramenta Selenium IDE. Na aba "Tabela" é possível alterar, deletar ou inserir um comando referente ao teste gravado. A ferramenta Selenium IDE dispõe seus comandos baseados em JavaScript puro. Na aba código fonte é apresentado o código do teste em HTML, tornando possível também inserções, alterações ou exclusões.

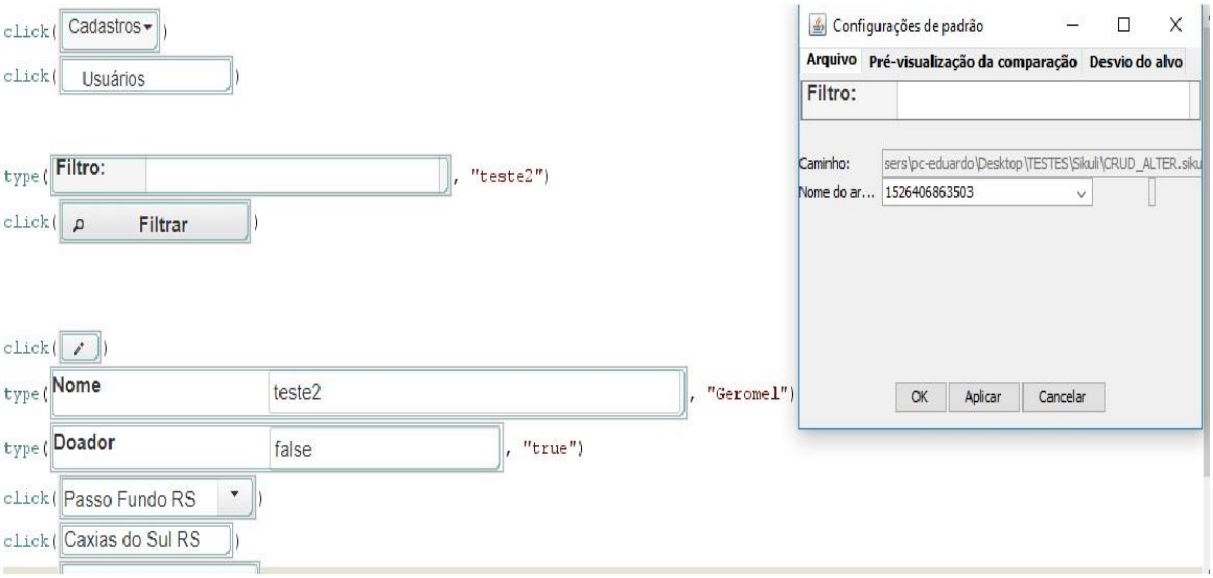
Figura 12: Código Teste Selenium IDE



Fonte: (DO AUTOR, 2018)

A figura 13 ilustra os tratamentos de código da ferramenta Sikuli, seu próprio *test suíte* oferece ao usuário métodos de alteração do código gerado na automação de teste. Além de oferecer um filtro de arquivos que aceita um código escrito somente na linguagem python.

Figura 13: Código Teste Sikuli



Fonte: (DO AUTOR, 2018)

Para avaliar os métodos de tratamento de código de cada ferramenta foram aplicados os testes especificados nos casos de teste Imprimir, Ordenar Registros e Buscar Nome. A tabela 3 apresenta os respectivos resultados de cada uma das ferramentas em relação aos casos de testes citados anteriormente.

Tabela 3: Casos Ordenar, Buscar e Imprimir

	Ordenar Registros	Buscar Nome	Imprimir
Badboy	SUCESS	SUCESS	FAILED
Selenium	SUCESS	SUCESS	SUCESS
Sikuli	SUCESS	SUCESS	FAILED

Fonte: (DO AUTOR, 2018)

Após a aplicação dos casos de teste, as três ferramentas concluíram, sem nem uma falha, os casos de teste: Ordenar Registros e Buscar Nome. A ferramenta Selenium IDE concluiu com êxito todos os casos de teste declarados nesta sessão. As ferramentas BadBoy e Sikuli falharam na execução do caso Imprimir. A seguir são descritos os erros:

Utilizando as ferramentas Sikuli e BadBoy, no caso de teste “IMPRIMIR”, não foi disponibilizado um código de tratamento para funcionalidades JSF que realizam a conexão com dispositivos externos, cujos implementam funções de downloads programadas na aplicação.

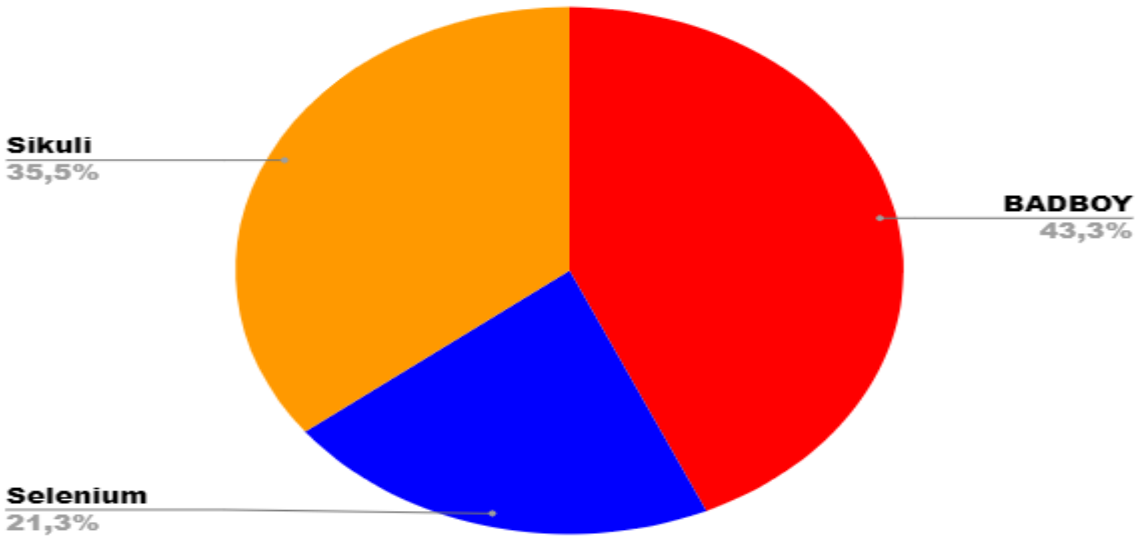
4.3 TEMPOS DE EXECUÇÃO

Nessa seção é avaliado o desempenho de cada ferramenta ao aplicar os casos de teste CRUD Usuários e Manutenção de Endereços. Foram realizados comparativos entre os tempos obtidos em cada ferramenta ao realizarem os testes citados anteriormente. Também são comparados os tamanhos de cada arquivo de teste, quando o mesmo é finalizado. O teste ao ser salvo por uma ferramenta se transforma em um arquivo. Cada ferramenta testada neste trabalho gera um arquivo de teste de acordo com suas funcionalidades. Os tamanhos de arquivos de teste variam de cada ferramenta, pois o mesmo teste pode ter sido acrescentado passos adicionais, em relação a outro programa. Um arquivo de teste ao ser inicializado pela ferramenta que lhe construiu, comtempla todas as funcionalidades salvas anteriormente no seu desenvolvimento. Todas as ferramentas testadas no presente trabalho possuem características diferentes ao criarem um arquivo de teste, porém elas têm em comum um "*log*" de cronometragem, referente ao tempo de execução de um teste, através desta funcionalidade tornou-se possível compará-los entre si e descrever conclusões sobre os resultados obtidos.

Nas figuras 14 e 15 são apresentadas as médias dos resultados obtidos nas tabelas 4 , 5 e 6, referentes ao caso de teste CRUD Usuários. A figura 14 apresenta a soma dos tempos de cada passo do caso de teste, cujo foi dividido em três passos. Um gráfico no modo porcentagem foi utilizado para exibir a soma de tempo dos passos de cada ferramenta ao finalizar o caso de teste CRUD usuários. A

ferramenta BadBoy utilizou 43% do tempo total em que o teste foi executado pelo programa. A ferramenta Sikuli utilizou 35,5% e a ferramenta Selenium 21,3%.

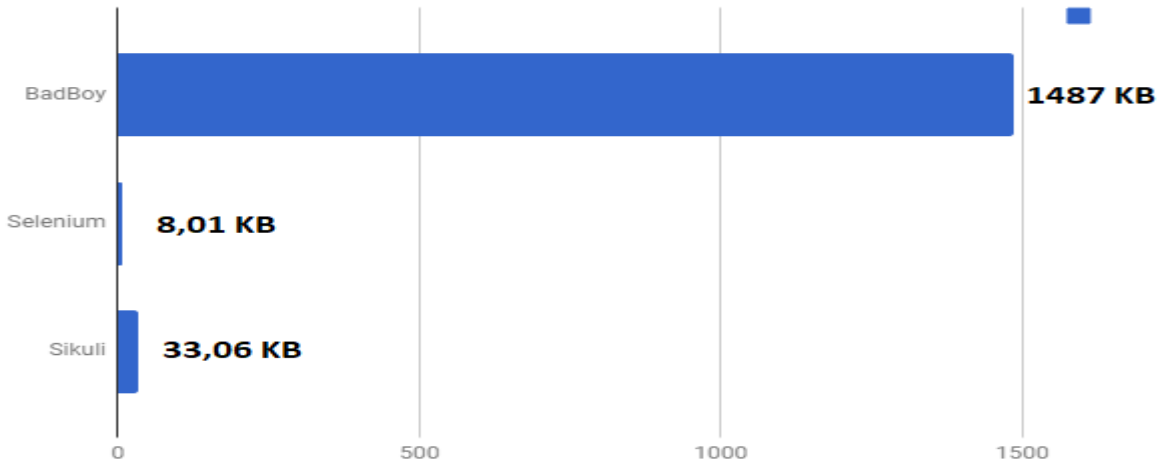
Figura 14: Tempos de execução em ms do teste CRUD



Fonte: (DO AUTOR, 2018)

A figura 15 apresenta os resultados do caso de teste CRUD Usuários referentes ao tamanho do arquivo gerado quando o teste foi finalizado. A ferramenta BadBoy concluiu o caso de teste CRUD com um arquivo de 1487 KB, a ferramenta Sikuli com 33,06 KB e Selenium com 8,01 KB.

Figura 15: Tamanhos dos arquivos do teste CRUD em KB



Fonte: (DO AUTOR, 2018)

As tabelas 4, 5 e 6 apresentam os resultados de cada ferramenta em relação aos casos de testes citados. Os campos analisados das tabelas são denominados como: Status (demonstra como foi concluído o teste), Tempo (tempos de execução da ferramenta em relação ao caso de teste, apresentados em Milissegundo) e Tamanho (apresenta o tamanho do arquivo do teste após ser finalizado, medido em Kilobytes). No caso de teste CRUD Usuários o mesmo foi dividido em três partes, referentes aos passos do teste.

Tabela 4: CRUD + Manutenção de endereços BadBoy

BadBoy	CRUD Usuários	Manutenção Endereços
Status	OK	SUCESS
Tempo	Inserir: 3902 (ms) Alterar: 4020 (ms) Excluir: 1502 (ms)	1133 (ms)
Tamanho	Inserir: 598,00 KB Alterar: 688,00 KB Excluir: 201,00 KB	116,00 KB

Fonte: (DO AUTOR, 2018)

Na ferramenta BadBoy, ao executar o caso “CRUD USUÁRIOS”, a ferramenta precisou carregar a página inteira antes de aplicar a sua automação. Isso influenciou diretamente na agilidade do teste. A ferramenta Selenium finalizou com sucesso os casos de teste, além de obter os melhores tempos no caso CRUD, produziu arquivos mais leves que as demais ferramentas.

Tabela 5: CRUD + Manutenção de endereços Selenium IDE

Selenium	CRUD Usuários	Manutenção Endereços
Status	SUCESS	SUCESS
Tempo	Inserir:1900 (ms) Alterar:2124 (ms) Excluir: 603 (ms)	533 (ms)
Tamanho	Inserir:3,00 KB Alterar:3,91 KB Excluir: 1,10KB	1,00 KB

Fonte: (DO AUTOR, 2018)

A ferramenta Sikuli, ao executar o caso de teste “MANUTENÇÃO ENDEREÇOS”, apresentou erro na funcionalidade “*dragDrop*”, o segundo parâmetro não reconheceu o atributo como uma imagem. A solução foi editar o método persistindo um campo de imagem diferente.

Tabela 6: CRUD + Manutenção de endereços Sikuli

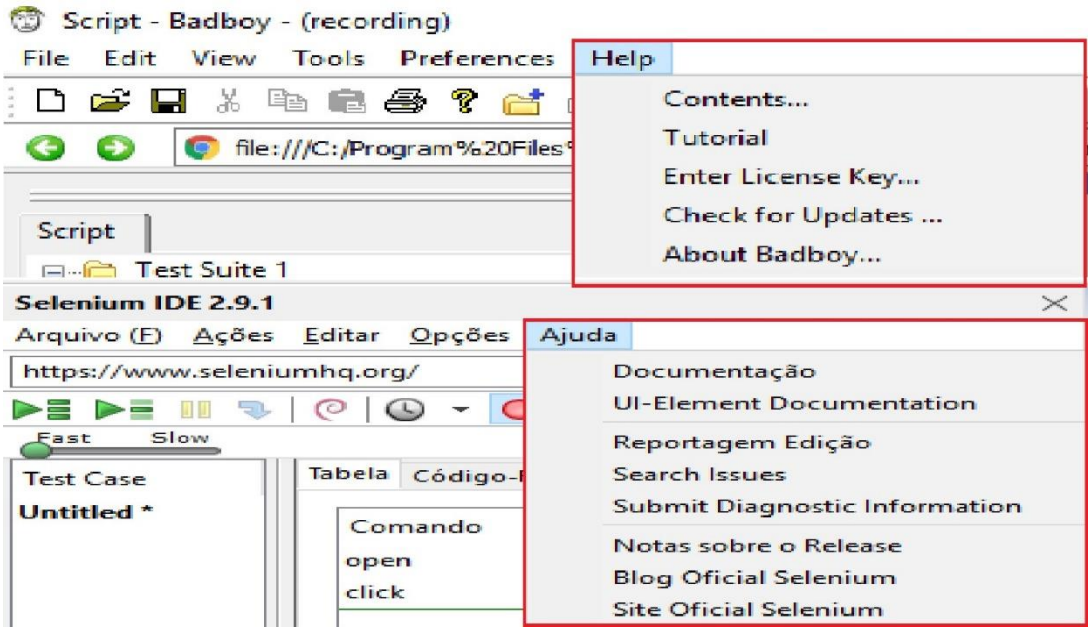
Sikuli	CRUD Usuários	Manutenção Endereços
Status	SUCESS	OK
Tempo	Inserir:3121 (ms) Alterar:3500 (ms) Excluir: 1102 (ms)	1578 (ms)
Tamanho	Inserir:9,72 KB Alterar:7,64 KB Excluir: 15,7 KB	3,28 KB

Fonte: (DO AUTOR, 2018)

4.4 DOCUMENTAÇÃO

Nesta etapa são avaliadas as documentações de cada uma das ferramentas analisadas no presente trabalho. Todas as ferramentas testadas possuem uma aba no seu ambiente gráfico que referencia os documentos de auxílio ao usuário. A ferramenta BadBoy apresenta um pequeno sistema para exibir a sua documentação, com uma interface gráfica para gerenciar os documentos. Essa interface auxilia o usuário a compreender documentos que estão sendo oferecidos pelo BadBoy, com abas de sumarização, índice e pesquisa, ainda é possível alimentar essa interface com documentos escritos e anotações do próprio testador. A documentação da ferramenta BadBoy inicia com um manual de instrução das funções básicas, finalizando a seção com uma introdução sobre automação de testes. As demais sessões de tutoriais da ferramenta BadBoy, são manuais de métodos, testes de carga e tratamento de scripts. Toda a documentação da ferramenta, disponibilizada dentro da mesma, é escrita em inglês. Como mostra a figura 16:

Figura 16: Aba Help das ferramentas BadBoy e Selenium IDE

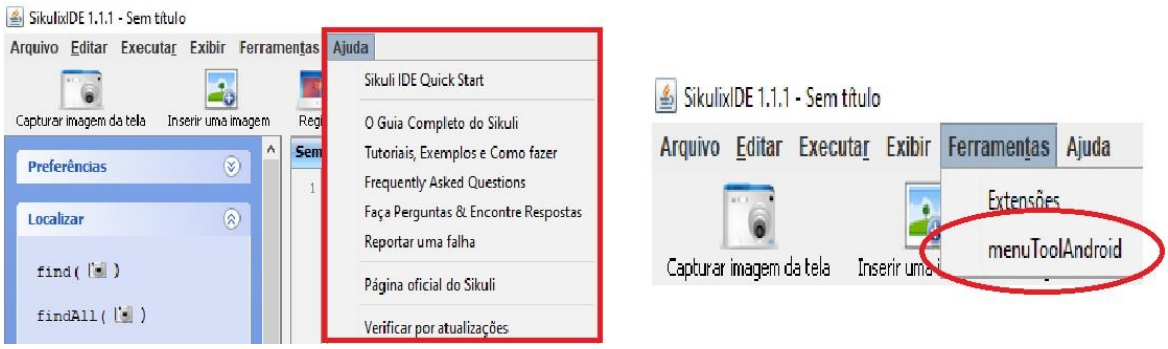


Fonte: (DO AUTOR, 2018)

A ferramenta Selenium IDE conta com uma documentação atrelada ao seu próprio site, porém é referenciada na aba “ajuda”. Uma interface gráfica construída em HTML é alimentada com manuais de usabilidade da ferramenta Selenium IDE.

Inicialmente uma breve introdução é disponibilizada ao usuário, porém essa introdução pode ser alterada ao trocar o valor da página referente a linguagem desejada. A aba de linguagens contém referências ao uso da ferramenta em Java, JavaScript, C#, Python, Ruby, PHP e Perl. A documentação da ferramenta Selenium ainda oferece manuais para a criação de casos de teste, alertas de *popUp*, tratamento de variáveis, erros de sintaxe e até mesmo depurador de bugs de código interno da aplicação. Toda a documentação oferecida pela ferramenta Selenium IDE está escrita em inglês. A documentação da ferramenta Sikuli é escrita no seu próprio Blog, porém em sua interface gráfica é disponibilizada através da aba “ajuda” o respectivo caminho (*link*) para o Blog. Na figura 17 é demonstrado o painel de ajuda, referente ao Sikuli:

Figura 17: Abas de arquivos de documentação do Sikuli



Fonte: (DO AUTOR, 2018)

A documentação do Sikuli se baseia em categorias de produtividade, compilação de testes e sugestões de uso da ferramenta em jogos. Todos os itens descritos no manual da ferramenta Sikuli estão disponibilizados através da aba “guia completo sikuli”, onde são apresentados links para opiniões de desenvolvedores, documentação de versões anteriores e métodos de conexões para a linguagem Java. Toda a documentação da ferramenta Sikuli está escrita em inglês.

5 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Atualmente é cada vez mais raro que uma aplicação web de grande porte seja desenvolvida sem ser testada, tanto pelas métricas manuais quanto automatizadas. O teste em geral permite ao desenvolvedor entregar um produto mais confiável e qualificado ao cliente. Porém quanto maior uma aplicação, mais tempo será necessário para ser testada. A aplicação de testes automatizados se torna muito importante nesse quesito. Uma vez que com o auxílio de uma ferramenta para automatizar um teste, o script gerado é salvo e disponibilizado para todas as ocorrências, iguais ou semelhantes que necessitam dos mesmos parâmetros em outro teste.

Assim, o presente trabalho teve como objetivo o estudo comparativo entre as ferramentas de automação de testes de software Badboy, Sikuli e Selenium IDE. A partir de um sistema WEB proposto como ambiente de testes, foi possível identificar aspectos relevantes em cada uma das ferramentas. Vale ressaltar que todas as ferramentas comparadas nesse estudo cumpriram com o objetivo de implementar os casos de testes no sistema WEB desenvolvido.

O BadBoy mesmo possuindo um navegador próprio, o que teoricamente deveria solucionar muitos bugs referentes a automação em sistemas WEB, mostrou-se menos eficaz em testes de performance, comparado as outras ferramentas. Diferente do Sikuli e Selenium IDE, ele disponibiliza seus comandos exibidos por ícones, o que dificultou muito a depuração do script e a visualização de erros do teste. Além do tempo de espera de carregamento da página do seu próprio navegador interferir diretamente na gravação do teste. O teste após finalizado pela ferramenta BadBoy se transforma em um arquivo com a extensão *record,* isso acabou gerando arquivos muito maiores em comparação as demais ferramentas. Porém a documentação da ferramenta BadBoy demonstrou ser a mais completa em relação aos manuais de usabilidade oferecidos aos seus usuários.

O Sikuli mesmo contando com uma interface discreta comparado as outras ferramentas, mostrou ser uma excelente escolha para aplicação de automações de pequeno porte. Forneceu todos os recursos necessários para a implantação dos casos de teste das seções 4.1, 4.2 e 4.3, porém, apresentou erros ao tratar componentes que estavam ocultos na tela. O reconhecimento de imagem

disponibilizado na automação da ferramenta Sikuli, concluiu todos os casos de testes referenciados no presente trabalho, mesmo quando a automação não reconheceu determinado componente do formulario foi possível tratá-lo diretamente no script do teste.

O Selenium IDE foi a ferramenta com o melhor desempenho nos testes de tempo, funcionalidades e tratamento com o código. Sem dúvida além de ser descrita como uma IDE, a ferramenta é recomendada para qualquer automação de teste, ela se destaca do BadBoy e Sikuli por conter enumeras opções de tratamento de HTML e JavaScript, o que é fundamental em testes de aplicações WEB. Além de conter uma interface simples e intuitiva o Selenium IDE gerou os arquivos de testes em códigos HTML, que ao serem abertos pela ferramenta são traduzidos como um teste. A ferramenta Selenium, por ser atrelada a um navegador, conseguiu reconhecer todos os elementos JSF implementados na plataforma de teste do presente trabalho.

Cada ferramenta, de acordo com seus próprios métodos, realizaram todas as automações descritas nos casos de testes, porém, cada uma possui uma área de teste mais recomendada sobre seu uso. A ferramenta de teste Sikuli apresar de conter apenas o método de reconhecimento de imagens, é recomendada para automações em sistemas de pequeno porte, além da sua velocidade de automação e reconhecimento gráfico. A ferramenta BadBoy, já possui um método ultrapassado de reconhecimento de HTML, pois a WEB convencional é desenvolvida com mais atributos agregados ao HTML, isso quebra as automações do Badboy, que é recomendado para páginas específicas que contenham testes pequenos de inserção de dados. Na ferramenta Selenium IDE notou-se uma grande variedade de recomendações do seu uso, em sistemas de pequeno e médio porte, a sua automação poderosa contém todos os recursos necessários para suprir um sistema de médio porte.

Como planejamentos para trabalhos futuros, pode-se estender o estudo da ferramenta Sikuli, desenvolvendo uma interface acoplada a um navegador, onde suas funcionalidades iniciais estejam presentes e acrescentadas a um módulo de tratamento para linguagem HTML e Javascript, usando como base os testes de código da ferramenta Selenium IDE identificados neste trabalho.

6 REFERÊNCIAS

BARBOSA, Ellen. MALDONADO, José. VINCENZI Auri. **Introdução ao Teste de Software**. Universidade de São Paulo, São Paulo, 2014 . Disponível em: <http://www.labes.icmc.usp.br/site/sites/default/files/NotaDidatica65.pdf>. Acesso em: 01 nov. 2017.

CAMPOS, J. **Mapeamento de Processos: Uma Estratégia Vencedora**. São Paulo, 2009. Disponivel em : <http://www.brisot.com.br/custom/34/uploads/cadastro/4/Artigo%20-%20MAPEAMENTO%20DE%20PROCESSOS.pdf >. Acesso em 03 novembro 2017.

CANDELORO, Cristiane. **Por que é importante testar um software?** . 2015. Disponível em: <https://imasters.com.br/desenvolvimento/software/por-que-e-importante-testar-um-software/?trace=1519021197>. Acesso em: 15 out. 2017.

CAVALCANTI, Carolina . **Interatividade em ambientes WEB - dando um toque humano a cursos on-line** . 2006 . Disponivel em : <http://noticias.universia.com.br/destaque/noticia/2006/01/16/451199/nteratividade-em-ambientes-web-dando-um-toque-humano-cursos-on-line.html> Aceso em 10 de outubro de 2017

COSTA, Guilherme . **SIKULI – O QUE É ? COMO UTILIZAR ?**. 2014. Disponível em: <https://guilerme18.wordpress.com/2014/02/13/sikuli-o-que-e-como-utilizar/>. Acesso em: 04 out. 2017.

COUTINHO ,Pedro H. Módulo de : **Teste de software**. ESAB – ESCOLA SUPERIOR ABERTA DO BRASIL LTDA , 2011

DIJKSTRA, E. W.; DAHL, O. J.; HOARE, C. A. R. **Structured Programming**. Londres: Academic Press, 1972.

DUTRA, A. **Abordagem ao Desenvolvimento de software Cleanroom**. mai, 2014. Disponivel em: < http://alexpagernet.blogspot.com.br/2014/05/abordagem-ao-desenvolvimento-de.html> . Acesso em 03 novembro 2017.

ELIZA, Renata . **Ferramentas de suporte ao Teste de Software**. 2013 . Disponível em: <http://www.devmedia.com.br/ferramentas-de-suporte-ao-teste-de-software/28642> Acesso em: 11 setembro. 2017.

FAGAN, M. E. **Advances in Software Inspections**. IEEE Trans. on Software Eng, v. SE-12, n. 7, 1986, p. 744-751.

FARIA, Thiago. **Java EE 7 com JSF, PrimeFaces e CDI**. 2.ed. São Paulo: Editora USP, 2015

GUGIK, Gabriel. **Código Aberto e Software Livre não significam a mesma coisa**. 2009. Disponível em: <https://www.tecmundo.com.br/linux/1739-codigo-aberto-e-software-livre-nao-significam-a-mesma-coisa-.htm> Acesso em 02 setembro 2017.

INTHURN, Cândida. **Qualidade e teste de software**. Florianópolis: Visual Books, 2001.

JUSTEN, Willian. **Entendendo Testes de Software**. 2015. Disponível em: <https://willianjusten.com.br/entendendo-testes-de-software/#unit>. Acesso em: 06 nov. 2017.

RIBEIRO, Leandro. **O que é UML e Diagramas de Casos de Uso**. 2016. Disponível em:< https://www.devmedia.com.br/o-que-e-uml-e-diagramas-de-caso-de-uso-introducao-pratica-a-uml/23408>. Acesso em: 25 abril. 2018.

LIMA, Júlio. **Automação de testes com Badboy**. 2014. Disponível em:<http://www.qualister.com.br/blog/automacao-de-testes-com-badboy-parte-1-introducao>. Acesso em: 20 out. 2017.

NASRALLAH, A. **Tributário nos bastidores**. São Paulo, agosto, 2016. Disponível em: <http://tributarionosbastidores.com.br/2016/08/spre/> . Acesso em 02 setembro 2017.

NOGUEIRA, Elias. **Introdução ao Selenium IDE**. 2014. Disponível em: <http://www.qualister.com.br/blog/introducao-ao-selenium-ide> Acesso em 04 outubro .2017.

OTTONI, Pasteur. **Testes de Caixa Branca e Caixa-preta**. 2010. Disponível em: <http://www.tesestec.com.br/pasteurjr/TCPB2.PDF>. Acesso em: 02 out. 2017.

PIERAZO , Cynthia., RODRIGUES, Leticia., SOARES, Hélio. **Análise Comparativa de Ferramentas de Teste para Aplicações em Banco de Dados** . 2013. Disponível em: <http://www.computacao.unitri.edu.br/erac/index.php/e-rac/article/download/121/155>. Acesso em: 01 de outubro . 2017.

PRESSMAN, Roger S. **Engenharia de software**. São Paulo: Pearson Makron Books, 1995

PRESSMAN ,Roger S. **Engenharia de Software** - Uma Abordagem Profissional. 7.ed. Amgh Editora, 2011

PROWELL ,Stacy J. **Cleanroom software engineering: technology and process**. Addison Wesley, 1999

RODRIGUES, R. **Tutorial de Utilização do Selenium**. Belo Horizonte, 2014. Disponivel em : < https://pt.scribd.com/document/258407118/Manual-de-Utilizacao-Do-Selenium-IDE > Acesso em 20 agosto 2017.

RODRIGUES, E. **Testes de Software com Ferramentas de automação**, 2018. Disponível em:<https://www.youtube.com/playlist?list=PL5RwNm7G4GGgPOst57jPDE3UdfOVbcly3>. Acesso em: 02 jul. 2018.

SANTOS , Ismaily., BEZERRA, Carla., MONTEIRO, Gustavo., ARAÚJO, Ítalo., OLIVEIRA, Talisson. **Uma Avaliação de Ferramentas para Testes em Sistemas de Informação Móveis baseada no Método DMADV** . 2016. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/sbsi/2013/0051.pdf>. Acesso em: 06 de outubro . 2017.

SOMMERVILLE, Ian. **Engenharia de software**. 6.ed. Editora Addison Wesley, 2003

SOMMERVILLE, Ian. **Engenharia de software**. 8.ed. Editora Addison Wesley, 2007

SOMMERVILLE, Ian. **Engenharia de software**. 9.ed. São Paulo: Editora PEB, 2011

ZANONI, Cynthia. **Testes de Software & Ferramentas de Testes**. São Paulo. 2014. p 23. Disponível em: <https://pt.slideshare.net/cynthiazanoni/testes-em-aplicaes-web> Acesso em 19 setembro 2017.