Selenium *record and playback* vs Cabelenium: uma análise comparativa para automação de testes funcionais

Rodolpho Traboussy

Orientador: Ricardo Terra

Departamento de Ciência da Computação
Universidade Federal de Lavras (UFLA)

rtraboussy@estudante.ufla.br

Abstract. *It is common for professionals to have difficulties in testing functionalities of software systems in companies with high scale of productivity and demand. However, this article is centered on the premise that the testing steps are essential for the success of a project, ensuring quality and user satisfaction through the full and correct functioning of the software system. This article compares Selenium, a popular and stable framework and adopted in several software factories with Cabelenium, a new feature-oriented testing framework. A comparative evaluation between these two frameworks – with a focus on developing and maintaining tests – highlight the strengths and weaknesses of each framework. Briefly, Cabelenium has shown itself better in code, writing, development time, performance and maintainability but Selenium is still better in terms of learning.*

Resumo. *É comum profissionais terem dificuldades em testar funcionalidades de sistemas de software em empresas com alta escala de produtividade e demanda. No entanto, este artigo é centrado na premissa de que as etapas de testes são fundamentais para o sucesso de um projeto, garantindo qualidade e satisfação do usuário pelo pleno e correto funcionamento do sistema de software. Diante desse cenário, este artigo compara o Selenium, um framework popular, estável e adotado em diversas fábricas de software com o Cabelenium, um novo framework de testes orientado a funcionalidades. Uma avaliação comparativa entre esses dois frameworks – com foco na construção e manutenção de testes – destacou pontos positivos e negativos de cada framework. De forma sucinta, o Cabelenium se mostrou melhor em código, escrita, tempo de desenvolvimento, desempenho e manutenibilidade, porém o Selenium continua melhor em termos de aprendizagem.*

1. Introdução

Garantir a qualidade de um sistema de *software* é de suma importância, porém um grande desafio para fábricas de software pela dificuldade em ser alcançada (FELDERER; GAROUSI, 2020; BERNARDO, 2011. p. 221). Isso ocorre devido à alta complexidade dos produtos e em virtude das diversas dificuldades encontradas no processo de desenvolvimento, como falhas de verificação, erros de código ou até implementação incorreta (ROMANINI; SOTTO, 2019).

Assim, no intuito de se obter um sistema com credibilidade e confiabilidade, realizam-se diversos processos de testes até a entrega do produto final. Ainda mais importante, foram desenvolvidas – ao longo dos anos – ferramentas para automatizar e simplificar o desenvolvimento de tais testes, principalmente por novas empresas no mercado (SARAVANAN; PRASAD, 2016).

O Selenium[1], atualmente, se apresenta como um dos *frameworks* mais utilizados para automatizar testes funcionais. Além de possuir suporte para uma ampla gama de linguagens de programação (e.g., Java, Perl, C#, Ruby e Python), o Selenium facilita a criação dos casos de testes por meio de *scripts* que podem ser construídos de forma simples, já que praticamente todo o controle com o navegador (iteração, monitoramento, etc.) é realizada de forma transparente pelo *framework* (RAMYA; SINDHURA; SAGAR, 2017).

Ainda, o Selenium – ao longo dos anos – vem reduzindo cada vez mais o esforço na criação dos testes. Por exemplo, o seu recurso *record and playback* permite gravar os passos executados e depois reproduzí-los automaticamente. Por isso, alguns autores o consideram como uma estrutura de testes flexível, rápida e dinâmica para páginas web (RAMYA; SINDHURA; SAGAR, 2017).

No entanto, como esperado, o uso não adequado do Selenium com *record and playback* pode acarretar problemas. Por exemplo, gerará muito código desnecessário, o qual deve ser revisado (informalmente, "enxugado" seria o melhor termo). Como um outro exemplo, mesmo sem o uso do recurso de criação automática, os testes podem apresentar baixa manutenibilidade se abstrações não forem realizadas (GOJARE; JOSHI; GAIGAWARE, 2015; ATEŞOĞULLARI; MISHRA, 2020). Assuma que todo o sistema utilize um padrão para os identificadores dos elementos e tem-se 100 testes. Se o padrão do sistema mudar, todos os 100 testes devem ser modificados. Porém, caso existisse uma função que aplicasse esse padrão, bastaria a alteração em um único ponto (SILVA; ALVES; BRUNO, 2011).

Outro problema recorrente que acontece no *record and playback* está relacionado ao código produzido não possui total confiança de que irá funcionar logo após a gravação, pois o recurso não grava comandos que fazem o *script* de teste aguardar resposta do sistema web. Qualquer atraso no sistema em processar alguma operação resulta em falha no teste, sendo necessário ajustar o código manualmente.

No intuito de forçar o uso mais correto do Selenium, o Cabelenium foi proposto durante uma consultoria de automação de testes na Cabtec Tecnologia e Sistemas (TERRA, 2020). Esse *framework* é desenvolvido em cima do Selenium e herda todas as suas funcionalidades. A ideia inicial é manter o mesmo poder computacional do Selenium, mas adicionando uma camada de abstração, com o intuito de formar um código mais direcionado à uma prática de escrita de teste que reduza a chance de aparecimento de problemas, principalmente relacionados à manutenibilidade e à evolução dos testes. Porém, algumas funcionalidades foram incluídas de forma nativa para facilitar a reexecução parcial dos testes e a verificação de consistência em banco de dados.

Este artigo, portanto, visa realizar um comparativo entre esses dois *frameworks* ressaltando seus prós e contras, haja vista que o tempo de um *tester* criar, corrigir e evo-

[1] https://www.selenium.dev

luir testes é crucial. Como resultado, foi observado que o Cabelenium superou o *record and playback* do Selenium em vários aspectos, como código, escrita, tempo de desenvolvimento, desempenho e manutenibilidade. Entretanto, o recurso do Selenium continua sendo uma ferramenta de mais fácil aprendizagem e de usabilidade mais simplificada, oferecendo uma plataforma mais prática de desenvolvimento, especialmente para novos usuários.

O restante deste artigo é organizado como a seguir. A Seção 2 apresenta um contextualização sobre testes de *software*. A Seção 3 introduz a ferramenta Cabelenium, seu funcionamento e características. A Seção 4 conduz uma avaliação comparativa entre o Selenium e o Cabelenium com foco em aprendizagem, código, escrita, tempo de desenvolvimento, desempenho e manutenibilidade. Por fim, a Seção 5 apresenta os trabalhos relacionados e a Seção 6 conclui.

2. Teste de Software

A elaboração de um sistema de software não é algo simples e toda construção de sistema de software está sujeita a diversos problemas que fazem com que o produto final seja diferente daquele que era esperado (DELAMARO; MALDONADO; JINO, 2013). Por isso, as empresas buscam identificar os erros e defeitos em códigos que afetam a funcionalidade da ferramenta antes da entrega final, pois, além de passar uma imagem negativa para o cliente, essas falhas podem causar prejuízos ou não atender alguma funcionalidade que o software deveria possuir (DELAMARO; MALDONADO; JINO, 2013).

O teste de software é uma área que carece de investigação e análise tendo em vista a sua importância para o mercado, indústria e academia (FELDERER; GAROUSI, 2020). Segundo a IEEE (2008), o processo de teste determina se o produto desenvolvido para uma dada atividade está de acordo com os requisitos daquela atividade e se o sistema de software satisfaz as intenções de uso e as necessidades do usuário. Esta seção apresenta uma breve contextualização de alguns tipos de testes relevantes para este artigo.

2.1. Teste funcional

Também conhecido como teste de caixa-preta ou teste comportamental, o teste funcional é um tipo de teste que foca nos requisitos funcionais do sistema, desconsiderando detalhes da implementação e dando atenção às funções que o software contemplará (PRESSMAN; MAXIM, 2014). Por sua vez, os testes funcionais são complexos devido ao fato de exigirem profundo conhecimento da regra de negócio de uma aplicação, para contemplar todas as variações de cenários existentes (PFLEEGER, 2004. p. 560; BARTIÉ, 2002). Diferentemente da maioria dos testes, que são realizados nas etapas iniciais do processo de teste, os funcionais tendem a ser aplicados durante as últimas etapas (PRESSMAN; MAXIM, 2014).

O Código 1 apresenta um exemplo de teste funcional que verifica se o vínculo de um funcionário a um departamento no sistema está funcionando corretamente. Dentro deste exemplo, é criado um departamento e ao criar um funcionário já o vincula ao departamento recém-criado.

Código 1. Exemplo de teste funcional

```
1. Dado um usuário com o perfil de administrador.
2. Acessa a tela "Departamento".
3. Usuário preenche os campos nome e descrição.
4. Ao clicar em salvar, é informado que o departamento "X" foi salvo.
5. Acessa a tela "Funcionário".
6. Usuário preenche os campos nome, cargo e salário.
7. Usuário preenche o campo departamento com o "X".
8. Ao clicar em salvar, é informado que o funcionário foi salvo.
```

2.2. Teste automatizado

Os testes automatizados são desenvolvidos com a proposta de construir um software ou *script* externo que têm como principal objetivo exercitar o sistema, testando as funcionalidades e verificando se estão de acordo com as especificações dos requisitos do sistema e os objetivos esperados (BERNARDO; KON, 2008; ROMANINI; SOTTO, 2019). Segundo Walker (2020), o teste automatizado realiza os procedimentos que seriam feitos por um humano de forma manual e os executa automaticamente por meio do uso de algum equipamento ou aplicativo.

O Código 2 ilustra um teste automatizado de uma transferência entre duas contas. Vale ressaltar que o objetivo do código é testar se o método `adicionaDiasEmUmaData` está funcionando corretamente, realizando a criação dessas contas com respectivo saldo (linhas 3 e 4), depois é transferida uma quantia da conta de origem para a conta de destino (linha 6) e, por fim, o sistema verifica o saldo da conta de destino e o saldo remanescente na conta de origem (linhas 8 e 9).

Código 2. Exemplo de teste automatizado com JUnit

```
@Test
public void testTransferencia() {
  Conta origem = new Conta("CB-001", 10000);
  Conta destino = new Conta("CB-002", 7500);

  origem.transferir(destino, 1500);

  assertEquals(9000, destino.getSaldo());
  assertEquals(8500, origem.getSaldo());
}
```

2.3. Teste funcional automatizado

Assim como os testes manuais, os testes funcionais automatizados têm como objetivo melhorar a qualidade do produto desenvolvido através da validação e verificação (MOLINARI, 2008). Ainda que um nível elevado de automação não substitua um processo manual bem organizado e racional de qualidade, a automação de testes funcionais permite um melhor uso dos recursos humanos disponíveis (PRESSMAN; MAXIM, 2014; MOLINARI, 2008). Segundo Holmes e Kellogg (2006), automatizar testes funcionais é um dos problemas tradicionais enfrentados por projetos de desenvolvimento de sistemas de software. Ainda segundo o referido autor, ao longo dos anos foram desenvolvidas diversas ferramentas para automatizar as funcionalidades de um programa. Diante disso, o processo de automação de testes funcionais permitiu a criação de *scripts* para as ações do usuário, como em ferramentas com recurso *record and playback*, que permitem a criação de *scripts* que interagem com as ações executadas no navegador e depois as reproduzem automaticamente (CHRISTOPHE et al., 2014).

Uma das ferramentas desenvolvidas para realizar essa automação é o Selenium, um *framework* de testes que foca principalmente em testar aplicações web que comunica diretamente com o navegador e usa seu suporte nativo para automatizar o processo de execução dos casos de teste (RAMYA; SINDHURA; SAGAR, 2017), além de contar com o recurso *record and playback*. A automação de testes funcionais é muito útil quando os casos de teste devem necessariamente ser repetidos várias vezes (RAMYA; SINDHURA; SAGAR, 2017), como é o caso dos testes de regressão que reexecutam alguns subconjuntos de testes que já foram conduzidos para garantir que mudanças não propagaram efeitos colaterais e erros adicionais no código implementado (PRESSMAN; MAXIM, 2014).

O Código 3 apresenta um teste funcional automatizado desenvolvido com Selenium que tem por objetivo automatizar o teste funcional visto na Seção 2.1 (Código 1). Por fins de legibilidade, assuma-se que o usuário já está logado e na tela principal do sistema.

Código 3. Exemplo de teste funcional automatizado com Selenium

```
@Test
public void testVincular(){
  ...
  DRIVER.findElement(By.id("menuDepto")).click();
  DRIVER.findElement(By.id("deptoForm:nome")).sendKeys("TI");
  DRIVER.findElement(By.id("deptoForm:descricao")).sendKeys("Nerds");
  DRIVER.findElement(By.id("btnSalvar")).click();
  assertEquals("Departamento salvo.", DRIVER.findElement(By.id("msg")).getText());

  DRIVER.findElement(By.id("menuFunc")).click();
  DRIVER.findElement(By.id("funcForm:nome")).sendKeys("José da Silva");
  DRIVER.findElement(By.id("deptoForm:cargo")).sendKeys("Tester");
  DRIVER.findElement(By.id("deptoForm:salario")).sendKeys("8130,00");

  Select select = new Select(DRIVER.findElement(By.id("comboDepto"));
  select.selectByVisibleText("TI");

  DRIVER.findElement(By.id("btnSalvar")).click();
  assertEquals("Funcionário salvo.", DRIVER.findElement(By.id("msg")).getText());
}
```

3. CABELENIUM

Inspirado e baseado no Selenium, o *framework* Cabelenium foi desenvolvido para remover a flexibilidade que pode favorecer o aparecimento de problemas, principalmente relacionados à manutenibilidade e à evolução dos testes (TERRA, 2020). Mais importante, o Cabelenium promove um desenvolvimento orientado à funcionalidade (termo que é explicado na Seção 3.2) e provê suporte para consistência em banco de dados (BD).

3.1. Exemplo Motivador

Esta seção ilustra um exemplo motivador de como o Cabelenium pode ser utilizado em um sistema WMS (do inglês *Warehouse Management System*) que fundamentalmente gerencia armazéns.

O Código 4 ilustra um teste funcional em que o produto `TV` é cadastrado (linha 11) e 15 etiquetas são impressas (linha 12). Assim, são recebidas 15 unidades de `TV` do PJ01 (linha 14) e armazenadas no armazém `COBERTO` (linha 15). Nesse ponto, verifica-se se tem-se 15 unidades de `TV` disponíveis no estoque (linha 16). Por fim, planeja-se uma expedição de 14 unidades de `TV` para o PJ02 (linha 18). Logo após finalizar o planejamento, verifica-se se tem-se uma única unidade de `TV` disponível no estoque e 14 reservadas (linha 19).

Código 4. Exemplo de Teste Funcional com Cabelenium

```
@Test
@Clean({
    "Produto:TV",
    "ImprimirEtiqueta:TV",
    "Recebimento:REC01",
    "IniciarRecebimento:REC01",
    "Armazenar:REC01",
    "Expedicao:EXP01",
})
public void planejarExpedicao() {
    this.asYouDesire(ProdutoTestCase.class, "novo").single("TV.json").run();
    this.novaImpressaoEtiquetas("TV", "COBERTO", 15);

    this.novoRecebimentoCompleto("REC01", "[1;TV;15]", "PJ01");
    this.novoArmazenamentoCompleto("REC01");
    this.assertResultInDBMS(new SQL("estoque.sql").addParam("codigo", "TV"), 15, 0);

    this.novaExpedicao("EXP01", "[1;TV;14]", "COBERTO", "PJ02");
    this.assertResultInDBMS(new SQL("estoque.sql").addParam("codigo", "TV"), 1, 14);
}
```

3.2. Orientação à funcionalidade

O Cabelenium estipula como boa prática que, para cada funcionalidade do sistema, cria-se uma classe de teste em que:

- *Atributos*: representam os campos da tela e o atributo que identifica unicamente o registro principal da funcionalidade é anotado com `@Id`.

- *Métodos*: representam *métodos úteis* ou *métodos de caso de teste*. Por um lado, *métodos úteis* são aqueles que fomentam o reúso, por exemplo, `novo` que cria um novo registro preenchendo os campos da tela com os valores dos atributos e verificando se foi corretamente persistido e o método `edita` que busca um registro específico e o deixa em modo de edição. Por outro lado, *métodos de caso de teste* são aqueles que de fato proveem testes funcionais e são anotados com `@Test`.

Na linha 11 do Código 4, é invocado o método `novo` da classe `ProdutoTestCase` passando o arquivo JSON descrito no Código 5.

Código 5. JSON de um Produto

```
{
    "codigo"                    : "TV",
    "nome"                      : "Televisão",
    "cnpj"                      : 39789102000173,
    "tipo"                      : "Inteiro"
}
```

Conforme pode ser observado no Código 6, o Cabelenium inicializa os atributos `codigo`, `nome`, `cnpj` e `tipo` e invoca o método útil `novo`.[2] A partir de métodos de maior abstração – porém usando as funcionalidades do Selenium – entra-se na função novo da funcionalidade (linhas 11-12), preenche os campos (linhas 14–18) e salva o registro (linha 20) confirmando mensagem em tela (linha 21) e na base de dados (linha 22).

[2]Atributos que possuem um valor padrão devem ser inicializados e anotados com `@Default`, por exemplo, o campo de unidade na linha 7 do Código 6. Porém, caso um valor seja informado para o atributo, o valor padrão é ignorado.

Código 6. Fragmentos da classe `ProdutoTestCase`

```
1  public class ProdutoTestCase extends PlugWebRunnerTestCase {
2      ...
3      @Id private String codigo;
4      private Long nome;
5      private Long cnpj;
6      private String tipo;
7      @Default private String unidade = "un";
8      ...
9
10     public void novo() {
11         this.openMenu(Menu.PRODUTO);
12         this.click(Botao.NOVO);
13
14         this.selectMany("checkTipoProduto", this.tipo);
15         this.writeById("inputNome", this.nome);
16         this.writeById("inputCodigo", this.codigo);
17         this.selectOption("comboUnidade", this.unidade);
18         this.selecionarPJ("btSel_fornecedor", this.cnpj);
19
20         this.click(Botao.SALVAR);
21         this.assertMessagePanelContainsText("Registro incluído com sucesso.");
22         this.assertResultInDBMS(new SQL("produto001.sql").addParam("codigo",
               this.codigo), this.nome, this.cnpj, this.tipo, this.unidade);
23     }
24
25     public void edita(String codigo) {
26         this.openMenu(Menu.PRODUTO);
27         this.writeById("inputCodigo", codigo);
28         this.click(Botao.PESQUISAR);
29         this.assertTextAtSearchResultTable(0, "Código", codigo);
30         this.clickOnDefaultSearchResultTable(0);
31         this.assertInputValueById("inputCodigo", codigo);
32     }
33     ...
34 }
```

Mais importante, o método `novo` é genérico o suficiente para ser reutilizado por diversas outras funcionalidades que requerem a criação de um produto, tal como o planejamento de uma expedição ilustrado na Seção 3.1. Além disso, o método `edita` (linhas 25-32) pode ser utilizado por diversos casos de teste em que um produto deva ser alterado, por exemplo. Por fim, vários outros método úteis podem ser desenvolvidos – tais como `excluir`, `inativar`, `transferir`, etc. – dependendo das particularidades de cada funcionalidade.

3.3. Funções utilitárias

A criação de casos de testes devem ser criados – sempre que possível – a partir de invocações de *métodos úteis* na própria classe e de outras classes. Esse prática fomenta o reúso.

No caso de rotinas complexas que sejam largamente utilizadas dentro de um sistema, o Cabelenium sugere que sejam generalizadas e se tornem *funções utilitárias*. O teste funcional do Código 4 utiliza de quatro *funções utilitárias* denominadas `novaImpressaoEtiquetas`, `novoRecebimentoCompleto`, `novoArmazenamentoCompleto` e `novaExpedicao`. Por exemplo, a *função utilitária* `novoRecebimentoCompleto` é um atalho para o planejamento de um recebimento e o recebimento de fato de todos os itens previamente planejados, conforme pode ser observado no Código 7.

Código 7. Função utilitária `novoRecebimentoCompleto`

```
public void novoRecebimentoCompleto(String idRecebimento, String produtos, String nomeArmazem) {
    this.novoPlanejamentoRecebimento(idRecebimento, produtos, nomeArmazem, 1);

    Set<Integer> idEtiquetas = new HashSet<Integer>();
    for (ProdutoInfo pi : this.getProdutosInfo(produtos)) {
        idEtiquetas.addAll(Consultas.getEtiquetasPorProduto(pi.getCodigoProduto()));
    }
    this.asYouDesire(IniciarRecebimentoTestCase.class, "novo")
                    .addParam("identificadorExterno", idRecebimento)
                    .addParam("idEtiquetas", idEtiquetas)
                    .run();
}
```

Mais detalhadamente, o Código 7 utiliza outra *função utilitária* que faz o planejamento (linha 2), busca todas as etiquetas de cada produto (linhas 5-7) e então invoca o *método útil* `novo` (que de fato faz o recebimento) da classe `IniciarRecebimentoTestCase` passando o identificador do recebimento e as etiquetas dos produtos que estão sendo recebidos (linhas 8-11).

É importante salientar que a prática de (i) *funções utilitárias* chamando outras *funções utilitárias* ou *métodos úteis* ou (ii) *métodos úteis* chamando outros *métodos úteis* ou mesmo *funções utilitárias* visam fomentar cada vez mais o reúso dentro do projeto de teste.

3.4. Consistência em banco de dados

A garantia de que o que é feito no sistema é corretamente persistido é algo crítico em diversos tipos de sistemas de software. O Cabelenium, por sua vez, provê nativamente formas de verificação em banco de dados.

Basicamente, o método `assertResultInDBMS` recebe como entrada um objeto SQL e um arranjo de arranjo de objetos que deveriam ser a lista de tuplas retornadas pelo Sistema Gerenciador de Banco de Dados (SGBD).

O teste funcional do Código 4 chama duas vezes o *script* `estoque.sql` com o parâmetro `codigo` sendo "TV" e esperando uma única tupla [15,0] como resultado.[3] Como pode ser observado pelo Código 8, o 15 refere-se à quantidade disponível e 0 à quantidade reservada por outros planejamentos de expedições.

Código 8. SQL de verificação de consistência de estoque

```
select QTDE_DISPONIVEL, QTDE_RESERVADA from ESTOQUE e
            inner join PRODUTO p on e.ID_PRODUTO = p.ID
        where
            p.CODIGO = ':codigo'
```

É importante salientar que tanto as *funções utilitárias* quanto os *métodos úteis* devem incluir verificações de consistência genéricas. Isso implica em verificações de consistência sendo sempre realizadas em diferentes casos de teste o que provê uma maior confiabilidade nos testes. Por exemplo, toda vez ao se criar um produto (método útil `novo`, Código 6), é verificado se foi salvo conforme os valores informados (linha 22).

3.5. Casos de teste autocontidos

O Cabelenium preve que todos os casos de testes sejam autocontidos. Isso garante que a execução do caso de teste não gere efeitos colaterais em outros testes.

[3] Como o retorno é uma única tupla, o recurso varargs do Java permite que seja passado fora da definição de um arranjo de objetos. (ORACLE. . . , 2021)

Portanto, todos os casos de testes devem incluir uma anotação `@Clean` com tudo o que foi criado e deve ser excluído. Essa limpeza ocorre após a execução do teste para garantir que o sistema volte ao seu estado pré-teste. Por preciosismo, a limpeza ocorre também antes da execução para garantir não haver vestígios de uma falha não controlada durante uma execução prévia desse mesmo caso de teste ou de outros casos de testes que podem compartilhar mesmos nomes de identificadores únicos.

O teste funcional do Código 4 (linhas 2-9) tem seis *scripts* de testes sendo executados antes e depois de sua execução. Por exemplo, o "`Produto:TV`" indica que o Cabelenium executará a limpeza a partir do arquivo `produto-clean.sql` (Código 9) passando "`TV`" como parâmetro (`: codigo`). Particularmente para a limpeza de produto, são excluídos os registros de estoque (um para cada armazém) e depois é excluído o produto em si. É importante mencionar que, no decorrer do projeto, os *scripts* de limpeza tendem a crescer.

Código 9. SQL de limpeza de produto

```
--Apagando Estoque
delete from ESTOQUE where IDATIVO IN (
    select p.ID from PRODUTO p where p.CODIGO = ':codigo'
);

--Apagando o produto em si
delete from PRODUTO where codigo = ':codigo';
```

Ainda mais importante, a limpeza permite otimizar correções. Tomando ainda como exemplo o teste funcional do Código 4, assuma que o teste falha e o desenvolvedor corrigiu um *bug* na tela de planejamento de expedição. Ao invés de executar todo o teste para uma aferição temporária, ele pode executar apenas o *clean* de planejamento de expedição (linha 8) e reexecutar somente as linhas 18 a 19.

3.6. Aplicabilidade

Como o *framework* é desenvolvido em cima do Selenium, o Cabelenium pode ser aplicado em qualquer sistema web. Embora o método de elaboração de testes é totalmente independente de tecnologia, as funções essenciais (*core*) são normalmente específicas para uma implementação de *front-end* específica, uma vez que cada implementação tem uma forma diferente de renderização de componentes *web*.

Atualmente, o Cabelenium tem um *driver* implementado para a biblioteca do PrimeFaces[4]. Assim, no caso de utilizar, por exemplo, HTML 5[5] ou Angular[6], basta implementar o *driver* para tal biblioteca.

4. Avaliação comparativa

Esta seção conduz uma avaliação entre os *frameworks* Selenium e o Cabelenium ressaltando seus prós e contras. Inicialmente, é apresentado o sistema a ser testado (Seção 4.1) e como a avaliação é realizada (Seção 4.2). Em seguida, são apresentadas as implementações de cinco testes funcionais usando tanto o Selenium (Seção 4.3) quanto o Cabelenium (Seção 4.4). Por fim, é apresentada uma discussão dos resultados (Seção 4.5) e destacadas as ameaças à validade (Seção 4.6).

[4]https://www.primefaces.org/

[5]https://dev.w3.org/html5/html-author/

[6]https://angular.io/

4.1. Sistema Alvo

O WMS GTI Plug é um sistema desenvolvido e mantido pela Cabtec Tecnologia e Sistemas, uma empresa especialista em *outsourcing* de mobilidade, identificação e rastreabilidade de dados. O sistema WMS supramencionado proporciona uma gestão de armazenamento por meio de uma ampla gama de funcionalidades, tais como gestão de produtos e pessoas, controle de expedições e recebimentos, importações de dados, etc.

É importante mencionar que o foco do sistema WMS está no recebimento e expedição. De forma bem sucinta, recebimento está vinculado à compra, i.e., ao receber um produto comprado, o sistema o estoca em um dado armazém. Expedição está vinculada à venda, i.e., ao expedir um produto vendido, o sistema o remove do armazém no qual estava estocado.

4.2. Método

Os seguintes passos foram realizados de forma a prover um comparativo justos entre os *frameworks* avaliados:

1. um *tester* da Cabtec elaborou uma lista com as seguintes cinco atividades importantes dentro de um sistemas de WMS: i) criação de um produto; ii) importação de uma pessoa jurídica; iii) impressão de dez unidades de um produto a ser criado, associado e atribuído a um armazém; iv) recebimento e armazenamento de dez unidades de um produto; e v) expedição de dez unidades de um produto.

2. o primeiro autor deste artigo implementou essas cinco atividades utilizando o Selenium através do recurso *record and playback*, e o Cabelenium. Durante essa implementação, o autor deste artigo atentou em fazer comentários em relação à aprendizagem, ao código, à escrita, ao tempo de desenvolvimento e à manutenibilidade.

3. Todas as anotações vão ser analisadas e discutidas para se discutir sobre os prós e contras de cada *framework* avaliado.

4.3. Implementação Selenium

Atividade 1 [criação de um produto]: Como pode ser observado no Código 17 encontrado no Apêndice A, após a autenticação (linhas 8-14) e acesso ao respectivo menu (linhas 17-19), preencheu-se vários dados do produto (linhas 21-43) e submeteu-se o formulário (linha 45). Foi então realizada uma verificação da mensagem exibida pelo sistema para confirmar que a operação foi realizada com sucesso (linha 46). O Código 10 ilustra um pequeno trecho (linhas 21-43) que refere-se à criação do produto. Como pode ser observado, mais de 20 linhas foram necessárias apenas para essa tarefa.

Código 10. Selenium: Trecho da criação de um produto (Atividade 1)

```
wait.until(ExpectedConditions.visibilityOfElementLocated(                                                    21
By.id("formCenter: produtosTab:inputNome")));                                                                22
driver.findElement(By.id("formCenter:produtosTab:inputNome")).click();                                       23
driver.findElement(By.id("formCenter:produtosTab:inputNome")).sendKeys("Teste DeProduto01");                 24
driver.findElement(By.id("formCenter:produtosTab:inputIdentExterna")).click();                               25
driver.findElement( By.id("formCenter:produtosTab:inputIdentExterna")).sendKeys("produto001");               26
driver.findElement(By.cssSelector("#btSel_fornecedor > span")).click();                                      27
WebElement element1 = driver.findElement(By.cssSelector("#btSel_fornecedor > span"));                        28
Actions builder = new Actions(driver);                                                                       29
builder.moveToElement(element).perform();                                                                    30
WebElement element2 = driver.findElement(By.tagName("body"));                                                31
Actions builder = new Actions(driver);                                                                       32
builder.moveToElement(element, 0, 0).perform();                                                              33
```

```
34
driver.findElement(By.cssSelector("#frmSelPJ\\3A btPesquisarPJ > .ui-button-text")).click();                                    35
WebDriverWait wait2 = new WebDriverWait(driver, 2);                                                                              36
wait2.until(ExpectedConditions.visibilityOfElementLocated(                                                                      37
      By.xpath("/html/body/div[1]/div[1]/div/span/div[2]/div[2]/form/div[3]/div/
      div[4]/div[2]/table/tbody/tr/td[2]")));
assertEquals(driver.findElement(By.cssSelector("#frmSelPJ\\:dtPJ_data >                                                           38
      tr.ui-widget-content.ui-datatable-even.ui-datatable-selectable > td:nth-child(2)")).getText(), "Pessoa
      Juridica Carga Ltda");
driver.findElement( By.xpath("/html/body/div[1]/div[1]/div/span/div[2]/div[2]/form/div[3]/div/                                   39
      div[4]/div[2]/table/tbody/tr/td[2]")).click();
driver.findElement(By.linkText("Informações adicionais")).click();                                                               40
driver.findElement(By.cssSelector(".col-sm-3:nth-child(1)")).click();                                                            41
driver.findElement( By.id("formCenter:produtosTab:inputLimitePalletVirtual")).click();                                           42
driver.findElement( By.id("formCenter:produtosTab:inputLimitePalletVirtual")).sendKeys("50");                                    43
```

Atividade 2 [importação de uma pessoa jurídica]: Como pode ser observado no Código 18 localizado no Apêndice A, após navegação inicial (autenticação e acesso ao respectivo menu), foi informado o arquivo CSV[7] (linhas 8-11) e submetido o formulário (linha 12). Por fim, foi realizada a verificação da mensagem na tela (linha 13) e aguardado alguns segundos para que o sistema informasse se a importação estava concluída (linhas 14-15), uma vez que a tarefa de importação é assíncrona.

Atividade 3 [impressão de dez unidades de um produto a ser criado, associado e atribuído a um armazém]: Para se ter um produto, todos os passos da Atividade 1 foram realizados a priori. Como pode ser observado no Código 19 localizado no Apêndice A, foi então associado um código EPC[8] a esse produto (linhas 7-20) e atribuído à três possíveis ruas em um dado armazém (linhas 30-56). Assim, foi realizada a impressão de 10 etiquetas para o produto (linhas 60-77). Em cada etapa concluída, é realizada uma verificação da mensagem exibida pelo sistema para confirmar que a operação foi realizada com sucesso (linhas 20, 56 e 77). O Código 11 ilustra um trecho (linhas 6-20) sobre a associação a um EPC.

Código 11. Selenium: Trecho da associação a um EPC (Atividade 3)

```
6
//Associando EPC                                                                                                                 7
driver.findElement(By.id("modulemenu_dadosBasicos")).click();                                                                    8
driver.findElement(By.id("menu_associarEpc")).click();                                                                           9
driver.findElement(By.cssSelector("#btNovo > .botao")).click();                                                                  10
wait.until(ExpectedConditions.visibilityOfElementLocated(
      By.cssSelector("#formCenter\\:associarTab\\:btSelecionar")));
driver.findElement( By.cssSelector("#formCenter\\:associarTab\\:btSelecionar")).click();                                         11
driver.findElement(By.id("frmSelProd:inputIdentExterna")).click();                                                               12
driver.findElement (By.id("frmSelProd:inputIdentExterna")).sendKeys("ProdutoAtv03");                                             13
driver.findElement(By.cssSelector("#frmSelProd\\3A btPsqBuscar > .ui-button-text")).click();                                     14
wait.until(ExpectedConditions.visibilityOfElementLocated( By.cssSelector("#frmSelProd\\:dtProduto_data > tr                     15
      > td:nth-child(2)")));
assertEquals(driver.findElement(By.cssSelector("#frmSelProd\\:dtProduto_data > tr:nth-child(1) >                                 16
      td:nth-child(1)")).getText(), "Produto03");
driver.findElement(By.cssSelector("#frmSelProd\\:dtProduto_data > tr > td:nth-child(2)")).click();                               17
driver.findElement(By.cssSelector("#btSalvar > .botao")).click();                                                                18
wait.until(ExpectedConditions.visibilityOfElementLocated( By.cssSelector("#formCenter\\:msgs_formCenter >                       19
      div > ul > li > span")));
assertEquals(driver.findElement(By.id("formCenter:msgs_formCenter")) .getText(), "Registro incluído com                          20
      sucesso.");
```

Atividade 4 [recebimento e armazenamento de dez unidades de um produto]: Para se ter um produto e etiquetas válidas, todos os passos da Atividade 3 foram realizados a priori. Como pode ser observado no Código 20 localizado no Apêndice A, foi planejado um recebimento de uma única etapa em um dado armazém (linhas 9-47). Assim, foi executado o recebimento de fato informando o código de cada etiqueta (a.k.a., GTI#) previamente impressa (linhas 51-98). Ao finalizar o recebimento, os produtos foram armazenados (linhas 101-111). Novamente, em cada etapa concluída, é realizada uma verificação

[7]CSV é um arquivo de texto com formato específico para possibilitar o salvamento dos dados em um formato estruturado de tabela (GOOGLE..., 2021)

[8]Código eletrônico do produto (EPC) é a identificação de produto de última geração; é dividido em número que identificam o fabricante e o tipo do produto (GS1..., 2021).

da mensagem exibida pelo sistema para confirmar que a operação foi realizada com sucesso (linhas 47, 98 e 111).

Atividade 5 [expedição de dez unidades de um produto]: Como pode ser observado no Código 21 localizado no Apêndice A, para se ter produtos armazenados, todos os passos da Atividade 4 foram realizados a priori. O processo de expedição é bem similar ao de recebimento, mas após o planejamento se realiza a separação dos produtos. Foi planejada uma expedição de uma única etapa (linhas 10-41) e os produtos foram devidamente separados (linhas 44-56). Assim, foi iniciada e finalizada a expedição de fato informando o código de cada produto previamente separado (linhas 59-72). Novamente, em cada etapa concluída, é realizada uma verificação da mensagem exibida pelo sistema para confirmar que a operação foi realizada com sucesso (linhas 41, 56 e 72).

4.4. Implementação Cabelenium

O desenvolvimento das atividades utilizando Cabelenium é diferente do Selenium. Como visto no Capítulo 3 – porém resumidamente – para cada entidade são criadas uma função "*search and edit*" que pesquisa e deixa uma entidade em modo de edição e uma função "novo" que cria uma nova entidade. Mais importante, no Cabelenium, as funções são genéricas e reutilizáveis para poderem ser chamadas em outros casos de teste sem a necessidade de se criar diversas implementações similares.

Atividade 1 [criação de um produto]: Como pode ser observado no Código 12, foi criada a função `novo` (linhas 16-31). Assim, basta invocar o método `novo` via método `asYouDesire` passando os parâmetros desejados que um novo produto será criado (linhas 39-40). Pode-se observar que, além da verificação da mensagem exibida pelo sistema para confirmar que a operação foi realizada com sucesso (linha 28), o caso de teste inclui a verificação da inclusão direto no banco de dados (linha 30).[9]

Código 12. Cabelenium: Criação de um produto (Atividade 1)

```
public class CriarProdutoInteiro extends PlugWebRunnerTestCase {                                                    1
    private static Logger LOGGER LoggerFactory.getLogger(CriarProdutoInteiro.class);                                 2
    @Id private String identificadorExterno;                                                                         3
    private String nome;                                                                                             4
    @Default private Integer limitePalletVirtual = 1000;                                                             5
    @Default private String tipoProduto = "Inteiro";                                                                 6
    @Default private String CNPJ = "39789102000173L";                                                                7
                                                                                                                     8
    public void searchAndEdit(final String identificadorExternoProduto) {                                           9
        this.openMenu(Menu.PRODUTO);                                                                                 10
        this.writeById("inputIdentificadorExterno", identificadorExternoProduto);                                    11
        this.click(Botao.PESQUISAR);                                                                                 12
        this.clickOnDefaultSearchResultTable(0);                                                                     13
    }                                                                                                                14
                                                                                                                     15
    public void novo() {                                                                                             16
        this.openMenu(Menu.PRODUTO);                                                                                 17
        this.click(Botao.NOVO);                                                                                      18
        this.selectMany("checkTipoProduto", tipoProduto);                                                            19
        this.writeById("inputNome", this.nome);                                                                      20
        this.writeById("inputIdentExterna", this.identificadorExterno);                                              21
        this.selecionarPJ("btSel_fornecedor", CNPJ);                                                                 22
                                                                                                                     23
        this.clickByLinkText("Informações adicionais");                                                              24
        this.writeById("inputLimitePalletVirtual", this.limitePalletVirtual);                                        25
                                                                                                                     26
        this.click(Botao.SALVAR);                                                                                    27
        this.assertMessagePanelContainsText("Registro incluído com sucesso.");                                       28
        LOGGER.trace("confirma na base de dados");                                                                   29
        this.assertResultInDBMS(new SQL("ProdutoInteiro.sql").addParam("idExterno", identificadorExterno),           30
             this.nome, this.limitePalletVirtual);
    }                                                                                                                31
                                                                                                                     32
    @Test                                                                                                            33
```

[9]É importante mencionar que a autenticação é realizada por um método que sempre é executada automaticamente antes de cada método.

```
    @Clean ({
        "Produto:ProdutoNovo001"
    })
    public void criarProdutoInteiro() {
        LOGGER.trace("Crio um produto inteiro");
        this.asYouDesire("novo").addParam("nome", "ProdutoTesteCase01")
            .addParam("identificadorExterno", "ProdutoNovo001").run();
    }
}
```

Atividade 2 [importação de uma pessoa jurídica]: De forma similar, como pode ser observado no Código 13, foi criada a função `novo` (linhas 4-11). Assim, bastou invocar o método `novo` via método `asYouDesire` passando os os parâmetros desejados que uma nova importação será realizada (linhas 16-17). Novamente, pode-se observar que ocorre a verificação da mensagem exibida pelo sistema para confirmar que a operação foi realizada com sucesso (linha 10). No entanto, optou-se por incluir verificação da inclusão no banco de dados no caso de teste (linha 20) devido a complexidade em se tratar diferentes formatos de arquivos CSV aceitos pelo sistema. Por fim, é importante mencionar que aguardou-se alguns segundos para que verificasse a inclusão dos registros no banco de dados (linha 22) uma vez que a tarefa de importação é assíncrona.

Código 13. Cabelenium: Importação de uma pessoa jurídica (Atividade 2)

```
public class Atividade02 extends PlugWebRunnerTestCase {
    private static Logger LOGGER = LoggerFactory.getLogger(Atividade02.class);
    private String caminhoArquivo;
    public void novo() {
        this.openMenu(Menu.IMPORTACAO_PESSOA);

        this.uploadFile("arquivoImportacao_input", caminhoArquivo);

        this.click(Botao.IMPORTAR);
        this.assertMessagePanelContainsText("Solicitação de importação enviada com sucesso");
    }

    @Test
    public void importarPJ() {
        LOGGER.trace("Informando o arquivo");
        this.asYouDesire("novo")
            .addParam("caminhoArquivo", "importacao/pjModel.csv");

        LOGGER.trace("confirma na base de dados");
        this.assertResultInDBMS(new SQL("importPJ.sql")
            .addParam("CNPJ", "84497910000111"),this.nomeCompleto, this.nomeReduzido);
        this.delay(13);
        this.assertMessagePanelContainsText("Concluída");
    }

}
```

Atividade 3 [impressão de dez unidades de um produto a ser criado, associado e atribuído a um armazém]: Como pode ser observado no Código 14, usou-se as funções `novo` de diversas outras telas como de produto previamente implementada na Atividade 1 (linhas 18-23), associação de EPC (linhas 26-28), atribuição de produto em endereço (linhas 31-35) e impressão de etiquetas (linhas 38-41). Dentro de cada método `novo` supramencionado, incluiu-se verificação de mensagem exibida pelo sistema e de registros em banco de dados.

Código 14. Cabelenium: Impressão de dez unidades de um produto a ser criado, associado e atribuído a um armazém (Atividade 3)

```
public class Atividade03 extends PlugWebRunnerTestCase {
    private static Logger LOGGER = LoggerFactory.getLogger(Atividade03.class);

    @Test
    @Clean ({
        "Produto:ProdutoAtv03",
        "ImprimirEtiqueta: ProdutoAtv03"
    })
    public void criarProdAssociarEPCAtrAtvEndImprmirEtiquetas() {
        private String identificadorExterno;
        private String nome;
        private String tipo = "Inteiro";
        @Default private Integer limitePalletVirtual = 1000;
```

```
        LOGGER.trace("Crio um produto inteiro");
        this.nome = "Produto003";
        this.identificadorExterno = "ProdutoAtv03";
        this.asYouDesire(ProdutoTestCase.class, "novo")
            .addParam("nome", this.nome)
            .addParam("idExterno", this.identificadorExterno)
            .addParam("tipo", this.tipo)
            .addParam("limitePalletVirtual", this.limitePalletVirtual)
            .run();

        LOGGER.trace("Associando EPC");
        this.asYouDesire(AssociarEpcTestCase.class, "novo")
            .addParam("identificadorExterno", this.identificadorExterno)
            .run();

        LOGGER.trace("Atribuindo Ativo-Endereço");
        this.asYouDesire(AssociarAtivoEnderecoTestCase.class, "novo")
            .addParam("identificadoresExterno", new String[] {this.identificadorExterno})
            .addParam("armazem", "Armazém com Porta-Pallet Carga")
            .addParam("ruas", new String[] {"1","2","3"})
            .run();

        LOGGER.trace("Imprimindo 10 Etiquetas");
        this.asYouDesire(ImprimirEtiquetaTestCase.class, "novo")
            .addParam("identificadorExternoProduto", this.identificadorExterno)
            .addParam("quantidadeEtiquetas", 10)
            .run();
    }
}
```

Atividade 4 [recebimento e armazenamento de dez unidades de um produto]: Quando um fluxo ocorre muito em um sistema, pode-se criar funções utilitárias que agilizam a codificação. Como pode ser observado no Código 15, as funções `novoProdutoInteiro` cria um produto inteiro, já o associa um código EPC e já o atribui a um endereço (linha 21), `novaImpressaoEtiquetas` realiza uma impressão de etiquetas (linha 24), `novoRecebimentoCompleto` planeja um recebimento e o executa (linha 27) e `novoArmazenamentoCompleto` armazena todo os produtos de um dado recebimento (linha 29). Novamente, dentro de cada método `novo` supramencionado, incluiu-se verificação de mensagem exibida pelo sistema e de registros em banco de dados.

Código 15. Cabelenium: Recebimento e armazenamento de dez unidades de um produto (Atividade 4)

```
public class Atividade04 extends PlugWebRunnerTestCase {
    private static Logger LOGGER = LoggerFactory.getLogger(Atividade04.class);

    @Test
    @Clean ({
        "Produto:ProdutoAtv04",
        "ImprimirEtiqueta:ProdutoAtv04",
        "Recebimento:ProdutoAtv04-REC",
        "IniciarRecebimento:ProdutoAtv04-REC",
        "Armazenar:ProdutoAtv04-REC"
    })

    public void receberEtiquetasEArmazenar() {
        private String identificadorExterno;
        private String nome;
        @Default private Integer limitePalletVirtual = 1000;

        LOGGER.trace("Crio um produto inteiro, associo EPC e associo ativo endereço");
        this.nome = "Produto004";
        this.identificadorExterno = "ProdutoAtv04";
        this.novoProdutoInteiro(this.nome, this.identificadorExterno, this.limitePalletVirtual, null);

        LOGGER.trace("Imprimir 10 etiquetas");
        this.novaImpressaoEtiquetas(this.identificadorExterno, "Armazém com Porta-Pallet Carga", 10, null);

        LOGGER.trace("Receber 10 etiquetas e armazená-las");
        this.novoRecebimentoCompleto("ProdutoAtv04-REC", "[1;ProdutoAtv04;10]", null);

        this.novoArmazenamentoCompleto("ProdutoAtv04-REC");
    }
}
```

Atividade 5 [expedição de dez unidades de um produto]: Como pode ser observado no Código 16, são novamente utilizadas todas as funções utilitárias utilizadas na Atividade 4, porém adicionando a chamada à função `novaExpedicao` a qual planeja uma expedição e a executa (linha 32). Novamente, é importante saliente que, dentro de cada

método `novo` supramencionado, incluiu-se verificação de mensagem exibida pelo sistema e de registros em banco de dados.

Código 16. Cabelenium: Expedição de dez unidades de um produto (Atividade 5)

```
public class Atividade05 extends PlugWebRunnerTestCase {
    private static Logger LOGGER = LoggerFactory.getLogger(Atividade04.class);

    @Test
    @Clean ({
        "Produto:ProdutoAtv05",
        "ImprimirEtiqueta:ProdutoAtv05",
        "Recebimento:ProdutoAtv05-REC",
        "IniciarRecebimento:ProdutoAtv05-REC",
        "Armazenar:ProdutoAtv05-REC",
        "Expedicao:ProdutoAtv05-EXP",
        "IniciarExpedicao:ProdutoAtv05-EXP"
    })
    public void expedir10Etiquetas() {
        private String identificadorExterno;
        private String nome;
        @Default private Integer limitePalletVirtual = 1000;

        LOGGER.trace("Crio um produto inteiro, associo EPC e associo ativo endereço");
        this.nome = "Produto005";
        this.identificadorExterno = "ProdutoAtv05";
        this.novoProdutoInteiro(this.nome, this.identificadorExterno, this.limitePalletVirtual, null);

        LOGGER.trace("Imprimir 10 etiquetas");
        this.novaImpressaoEtiquetas(this.identificadorExterno, "Armazém com Porta-Pallet Carga", 10, null);

        LOGGER.trace("Receber 10 etiquetas e armazená-las");
        this.novoRecebimentoCompleto("ProdutoAtv05-REC", "[1;ProdutoAtv05;10]", null);
        this.novoArmazenamentoCompleto("ProdutoAtv05-REC");

        LOGGER.trace("Expedir 10 etiquetas");
        this.novaExpedicao("ProdutoAtv05-EXP", "[1;ProdutoAtv05;10", "Armazém com Porta-Pallet Carga");
    }
}
```

4.5. Análise e discussão

As Seções 4.3 e 4.4 demostram as implementações de cinco atividades tanto no Selenium quanto no Cabelenium. O principal objetivo é vivenciar na prática o uso de dois *frameworks* para que uma comparação pudesse ser realizada. Portanto, a Tabela 1 apresenta a análise de seis aspectos.

O primeiro aspecto foi a aprendizagem. Muitos *frameworks* para automação de testes funcionais oferecem ferramentas diferentes para auxiliar na aprendizagem do processo de construção dos testes, como o *record and playback* que aumenta consideravelmente a curva de aprendizado sobre o Selenium e foi utilizada para construir os casos de teste deste artigo. No entanto, o recurso *record and playback* não é tão legível quanto à implementação baseada em código. É importante destacar que o *record and playback* apresenta aprendizagem distinta do Selenium com implementação baseada em código que, consequentemente, se difere do Cabelenium que necessita de conhecimentos lógicos de programação e linguagem Java.

Em termos de código, como pode ser observado na Tabela 1, o Selenium se comporta de forma muito repetitiva por não apresentar nenhuma reutilização de código já implementado e, com o uso do recurso *record and playback*, o sistema grava comandos desnecessários nos *scripts* de teste. Já o Cabelenium possui funções autocontidas que simplificam o código.

Outro ponto importante analisado é que ambas ferramentas possuem escrita simples e clara. Porém, o Cabelenium se sobresai em termos de praticidade devido ao reaproveitamento de código que gera uma escrita contínua, sem necessidade de refazer código produzidos anteriormente.

Tabela 1. Relação de prós e contras: organização do código

	Selenium *record and playback*	**Cabelenium**
Aprendizagem	Curva de aprendizagem menor. Muito simples e iterativo, com comandos e recursos fáceis de usar.	Curva de aprendizagem maior. É necessário conhecer a estrutura e ter noção razoável de programação e linguagem Java.
Código	Extenso e repetitivo. Não é autocontido nem reutilizável, há necessidade de se refazer os mesmos comandos várias vezes, e pausar a execução para aguardar o sistema terminar de executar uma função para conferir o valor de saída.	Código simples e claro. Não há necessidade de refazer funções, pois são reutilizáveis e autocontidas. Não há necessidade de se pausar a execução para aguardar resposta do sistema, o *framework* já possui isso implícito.
Escrita	Escrita simples, porém muito repetitiva, é necessário refazer o mesmo trecho de código diversas vezes durante os testes.	Escrita simples e mais prática. Não há quaisquer necessidades de se reescrever o mesmo trecho de código.
Tempo de Desenvolvimento	Tempo relativamente maior, é necessário fazer várias iterações até os passos dos testes funcionarem corretamente e não há reaproveitamento de código.	Tempo de desenvolvimento menor, não há necessidade de repetições e pode-se utilizar funções utilitárias para acelerar o processo.
Desempenho	Bom desempenho, porém gera muito código desnecessário com a utilização do *record and playback* o que pode gerar um *overhead*.	Como trabalha no topo do Selenium, o Cabelenium tem um pequeno *overhead*.
Manutenibilidade	Muito baixa, pois o uso do recurso *record and playback* gera código de baixa legibilidade.	Baixa, é necessário conhecer a plataforma antes de começar a desenvolver, porém mais legível do que a utilização do *record and playback*.

Fonte: do autor.

O quarto aspecto foi o tempo de desenvolvimento. Foi possível perceber novamente o Cabelenium se sobressaindo pela ausência de repetições na hora da implementação dos testes, e também pelo fato de já trazer funções utilitárias, onde os *scripts* mais repetitivos pode ser utilizados com mais facilidade. Por outro lado, o Selenium utilizando *record and playback* pode deixar o tempo de desenvolvimento pequeno no primeiro momento, porém os comandos gravados pela ferramenta não possuem segurança de que irão funcionar se executados logo em seguida, sendo necessário realizar várias iterações e modificações no código gravado para que funcione corretamente.

Ainda sobre tempo de desenvolvimento, é importante ressaltar que, apesar do tempo não ter sido medido utilizando mecanismos exatos, o autor deste artigo desenvolveu os testes em aproximadamente 6 horas utilizando Selenium. Para o Cabelenium, foram necessárias cerca de 3 horas para implementar os mesmos testes.

O desempenho é um fator muito notório em ambas as ferramentas. Muitos autores apresentam o Selenium como uma das ferramentas de melhor desempenho no mercado, porém há uma clara diferença quando se utiliza o recurso *record and playback* que grava muitos comandos desnecessários, o que pode provocar um *overhead*, fator que também está presente no Cabelenium que trabalha no topo do Selenium. Apesar dessa desvantagem, o Cabelenium pode contorná-la em teoria pelo ganho que possui no reaproveitamento de código.

Por último, a manutenibilidade foi um aspecto que deixou a desejar em ambas as ferramentas. No caso do Selenium utilizando o *record and playback*, a gravação dos passos gera código de baixa legibilidade pois o recurso grava qualquer iteração realizada no navegador deixando o código muito "sujo". Já no caso do Cabelenium, é necessário estudar a plataforma antes de desenvolver para entender como o *framework* funciona, o que faz ele sobressair em relação ao *record and playback* do Selenium. Porém, o aprendizado da plataforma do Cabelenium não é tão simples quanto à do Selenium pelo fato de o testador precisar de conhecimentos intermediários da linguagem Java.

Outro ponto importante a se observar é o fato do crescimento dos casos de teste ser muito maior no Selenium do que no Cabelenium. Por exemplo, na Atividade 1 foi implementado um código de 47 linhas, ao passo que no Cabelenium resultou em um código de apenas 29 linhas. Quando implementada a Atividade 03, o tamanho do caso de teste no Selenium aumenta consideravelmente pelo fato de o mesmo não fomentar reúso, ou seja, todo o código que foi feito na Atividade 01 deve ser replicado para a Atividade 3 com a adição do código responsável pela tarefa da Atividade 3.

Já no Cabelenium isso não ocorre. Só há uma chamada à função previamente implementada na Atividade 1 e em sequência é implementado o código referente a tarefa da Atividade 3. Conclui-se então que o código implementado no Selenium aumenta consideravelmente em número de linhas à medida em que se avança nas atividades, ou seja, na Atividade 5 é necessário reimplementar todo o código das Atividades 1, 3 e 4 ao passo que, no Cabelenium, o código fica menor em número de linhas por só precisar de uma chamada para cada função construída nas atividades previamente mencionadas.

Em relação às limitações, o Cabelenium funciona em cima do Selenium e não limita a utilização do Selenium. No entanto, o Cabelenium provê uma camada de abstração e sugere-se que o testador opte por utilizar preferencialmente as funções do Cabelenium, como um direcionamento à uma boa prática de desenvolvimento de testes. É importante destacar, contudo, que as funções básicas do Cabelenium são implementadas por instruções providas pelo Selenium.

Outra questão a se discutir está na forma de execução dos testes que é diferente nas duas ferramentas. No caso do Selenium, os testes podem ser executados diretamente na interface do aplicativo, onde a ferramenta irá abrir uma janela do navegador e interagir com ela em tempo real durante a execução do teste. O Cabelenium também realiza essa iteração com o navegador com a diferença de que não possui uma interface própria, isto é,

utiliza a interface provida pelo *framework* de testes de unidade. No entanto, o Cabelenium tem como vantagem a produção de um relatório notificando as ações conduzidas, funções utilizadas, validações no banco de dados e diversas outras informações sobre o teste que está sendo conduzido.

Por fim, é importante salientar que o Selenium foi utilizado e analisado neste artigo fazendo proveito do recurso *record and playback*. Portanto, nota-se uma diferença no desempenho da ferramenta quando utilizada com esse recurso do que quando é utilizada por meio de implementação de código de forma tradicional. É possível que muitos dos problemas encontrados no Selenium com o uso do *record and playback* não aconteçam quando ele é utilizado montando o código de forma manual, por exemplo, o número de iterações necessárias para os passo funcionarem corretamente é bem menor quando se projeta o *script* de teste manualmente. Enfim, é cabível de discussão o fato de que a comparação entre o Cabelenium e o Selenium com o uso do *record and playback* não produz os mesmos resultados do que se comparar o Cabelenium com o Selenium desenvolvido com implementação de código manual.

4.6. Ameaças à validade

É possível identificar pelo menos quatro ameaças à validade para este estudo. Em primeiro lugar, como utilizou-se de um único sistema, os resultados não podem ser extrapolados para qualquer sistema. No entanto, acredita-se que, pela natureza do GTI Plug, os resultados podem ser sim extrapolados para sistemas web com mesmo dinâmica de uso.

Em segundo lugar, o desenvolvimento dos casos de teste foram realizados pelo primeiro autor deste artigo o qual poderia apresentar certo viés. No entanto, no início do estudo, o primeiro autor não tinha prática nem sobre Selenium nem sobre Cabelenium, o que retrata um excelente perfil para fins de comparativos.

Em terceiro lugar, cabe pontuar que não foi utilizado nenhuma forma de medição exata do tempo de desenvolvimento dos testes, portanto os valores são aproximados. No entanto, como o autor desenvolvia os testes sempre em *slots* de tempo de 4 horas, a discrepância nos tempos não deve ser alta.

Por fim, ao implementar os testes usando Cabelenium, o primeiro autor deste artigo fez o uso de funções otimizadas (por exemplo, uma única chamada de método que planeja, separa e expede itens de uma dada expedição). Esses métodos trazem muito mais agilidade no processo de implementação dos testes uma vez que dispensa a reformulação de métodos que são muito utilizados pelo sistema e facilita a implementação dentro da plataforma do Cabelenium. No entanto, acredita-se que isso deva sim ser contemplado, já que a essência do Cabelenium é o reúso de testes já previamente realizados.

5. Trabalhos Relacionados

Rollwagen (2020) propôs um estudo comparativo entre ferramentas para automação de testes de software BadBoy[10], Selenium IDE e Sikuli[11]. A pesquisa foi realizada utilizando um sistema web desenvolvido como ambiente de testes permitindo aplicar as ferramentas e testar suas funcionalidades, identificando suas vantagens e limitações por

[10]www.badboy.com.au
[11]http://www.sikuli.org/

meio da análise de diferentes aspectos. O estudo ressaltou o Selenium IDE como sendo a ferramenta de melhor desempenho nos testes levando em conta fatores como tempo, funcionalidades e tratamento de código, o que o fez se destacar perante o BadBoy e o Sikuli. Apesar de os autores utilizarem um cenário de testes diferente, todo o processo é realizado utilizando-se uma abordagem similar à abordagem proposta neste artigo, explorando as funcionalidades das ferramentas.

Gojare, Joshi e Gaigawae (2015) apresentam um *framework* para automação de testes em aplicações web baseado no Selenium. Nesse estudo, o testador não precisa estudar a ferramenta Selenium em detalhes para desenvolver os casos de teste e apresenta um projeto modular mais útil para manter e reparar o conjunto de testes para uma nova versão da aplicação testada. Como resultado, os autores coletam resultados que mostram uma redução no tempo para construção dos casos de teste e um aumento no percentual de aceitação desses casos de teste. O *framework* Cabelenium proposto neste trabalho compartilha fundamentalmente o mesmo objetivo desse trabalho ao também utilizar um projeto modular mais útil e plataformas de automação de testes.

Atesogullari e Mishira (2020) realizam um estudo mais amplo comparando 21 ferramentas de automação de testes para ajudar profissionais a obter mais informações nessa área. O estudo se baseia na análise sistemática e comparativa das funcionalidades de cada uma das ferramentas e reforçam o fato de o Selenium se apresentar como uma das mais úteis no mercado, embora haja muitas outras ferramentas. Por exemplo, foram identificadas funcionalidades que são únicas em determinadas ferramentas uma vez que cada uma se encaixa nas necessidades do mercado de perspectivas diferentes. Segundo os autores, o Selenium se destaca por apresentar automações para diversos projetos e soluções para diferentes propósitos. Apesar de o cenário diferenciado, o estudo utiliza uma abordagem semelhante a proposta neste artigo para avaliar as ferramentas, inclusive análises comparativas em aplicações web que é o tipo de aplicação analisada neste artigo.

Kumar e Saxena (2015) apresentam uma abordagem para se construir um *framework* para automação de testes no Selenium utilizando Java. O estudo faz uso da ferramenta TestNG [12] para auxiliar no desenvolvimento do *framework* que provê um rico conjunto de instruções utilizadas para testes de unidades em aplicações web. Segundo os autores, um *framework* para automação de testes utilizando o Selenium se destaca por fornecer uma boa gama de aplicações para diferentes tipos de aplicações web, onde o testador pode conhecer a ferramenta de forma simplificada através do estudo direcionado das funcionalidades do Selenium. O estudo realça a dificuldade de se selecionar uma boa ferramenta para a automação de testes e analisa o Selenium sob uma perspectiva semelhante a deste artigo.

Islam e Quadri (2020) descrevem uma abordagem para se desenvolver um *framework* para automação de testes utilizando o Selenium. O estudo ressalta a importância da ferramenta para manter a padronização das atividades realizadas e para manipular as operações nos *scripts* de teste, impactando tanto positivamente quanto negativamente na ferramenta em si. O artigo utiliza as mesmas ferramentas e linguagens que este artigo para construir o *framework* de automação de testes. Como resultado, os autores alegam a dificuldade de se escolher um boa ferramenta para automação de testes e quais bene-

[12] https://testng.org/doc/

fícios elas podem trazer no processo de desenvolvimento. Além disso, o estudo realça flexibilidades do Selenium em se integrar com ambientes de desenvolvimento variáveis, navegadores e métodos de *design* para *scripts* de teste.

Vila, Novakova e Todorova (2017) realizaram um estudo sobre o Selenium da perspectiva de oportunidades e desafios enfrentados ao se produzir um *framework* para testes automatizados utilizando a ferramenta. Os resultados coletados foram que as vantagens do Selenium ainda prevalecem em questões como flexibilidade, desempenho e linhas de código, apesar de o desenvolvimento do *framework* incluir múltiplas subtarefas relacionadas à implementação de classes e métodos que são usadas para automatizar funcionalidades comuns. O artigo evidencia a importância do uso de *frameworks* para a automação de testes e como eles incrementam o uso do Selenium para executar tais testes, o que motiva estudos comparativos entre ferramentas de automação de testes puras e *frameworks* desenvolvidos a partir dessas ferramentas, como o conduzido neste artigo.

Singh e Sharma (2015) apresentaram um estudo comparativo entre duas ferramentas para automação de testes: Selenium e Sahi Pro[13]. O estudo teve enfoque no desempenho das duas ferramentas em um mesmo cenário e apresentaram resultados de que o Selenium é muito superior em questão de desempenho. É também conduzida uma avaliação das ferramentas sobre características semelhantes como tempo de execução e ferramentas para gravação dos testes. Esse trabalho utilizou como principal alvo de análise o desempenho das ferramentas, porém aborda outras características como eficiência e curva de aprendizado, que também foram alvo de análise neste artigo.

6. Conclusão

O Selenium é uma ferramenta *open-source* popular para a geração de testes automatizados voltados a aplicações web e contempla uma série de funcionalidades que facilitam sua compreensão e manuseio. Ainda mais importante, trata-se de uma ferramenta que provê suporte a automação de casos de teste em diferentes linguagens e navegadores. Entretanto, o uso não adequado dos seus recursos pode acarretar problemas. Por exemplo, o desenvolvimento de testes com o Selenium – sem a criação de abstrações de mais alto nível – deprecia o reúso e a manutenibilidade. Como um outro exemplo, o seu recurso *record and playback* grava qualquer movimento realizado na interface, gerando muito código desnecessário e testes que podem se quebrar facilmente com pequenas alterações na interface. Logo, se não utilizado adequadamente, gera outros problemas como legibilidade e estabilidade. Ainda, um mesmo procedimento feito por um usuário pode estar duplicado em inúmeros testes, enquanto que, no Cabelenium, seria natural modularizar tal procedimento em uma função utilitária.

Diante desse cenário, o objetivo do Cabelenium é: (i) prover um projeto modular mais rígido (i.e., mais direcionado a uma prática de escrita de testes) em cima do Selenium de forma a mitigar problemas relacionados à manutenibilidade e à evolução dos testes e (ii) incluir funcionalidades para facilitar a reexecução parcial dos testes e a verificação de consistência em banco de dados. Outro ponto importante sobre o Cabelenium se dá no fato de que ele foi projetado para funcionar em cima do Selenium, porém seu método, ou seja, como ele é utilizado pode ser reprojetado para funcionar em cima de qualquer outro *framework*. Portanto, este artigo conduziu uma avaliação comparativa do

[13]https://www.sahipro.com/

Selenium com o Cabelenium sob aspectos como aprendizagem, código, escrita, tempo de desenvolvimento, desempenho e manutenibilidade. Com base em cinco caso de testes em um sistema real de grande porte, o primeiro autor deste artigo os implementou utilizando tanto o Selenium quanto o Cabelenium.

Vale ressaltar que o Cabelenium supera o *record and playback* do Selenium em diversos pontos, principalmente com relação à facilidade de implementação e reusabilidade do código desenvolvido. Entretanto, o Selenium continua sendo uma ferramenta de mais fácil aprendizagem e de simples usabilidade, oferecendo uma forma muito prática para desenvolvimento dos testes através de seus recursos, especialmente para novos usuários. Cabe pontuar que os resultados comparativo obtidos neste artigo seriam diferentes se utilizado o desenvolvimento por meio do Selenium puro, ao invés do recurso *record and playback*.

Trabalhos futuros incluem: (i) comparativo usando outros sistemas; (ii) comparativo usando outras linguagens; (iii) comparativo com outros *frameworks*; (iv) análise de outros aspectos como modificações e evolução; e (v) condução de análises comparativas com mais participantes e utilizando de estudo cruzado.

Referências

ATEŞOĞULLARI, D.; MISHRA, A. Automation testing tools: A comparative view. **International Journal on Information Technologies & Security**, v. 12, n. 4, p. 1–14, 2020.

BARTIÉ, A. **Garantia de Qualidade de Software: adquirindo maturidade organizacional**. 3. ed. Rio de Janeiro: Elsevier, 2002. 291 p.

BERNARDO, P. C. **Padrões de testes automatizados**. Dissertação (Mestrado) — Universidade de São Paulo, Programa de Pós-Graduação em Ciência da Computação, 2011. p. 221.

BERNARDO, P. C.; KON, F. A importância dos testes automatizados. **Engenharia de Software Magazine**, v. 1, n. 3, p. 54–57, 2008.

CHRISTOPHE, L. et al. Prevalence and maintenance of automated functional tests for web applications. In: IEEE. **2014 International Conference on Software Maintenance and Evolution (ICSME)**. [S.l.], 2014. p. 141–150.

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. **Introdução ao teste de software**. [S.l.]: Elsevier, 2013. p. 433.

FELDERER, M.; GAROUSI, V. Together we are stronger: Evidence-based reflections on industry-academia collaboration in software testing. In: **International Conference on Software Quality (ICSQ)**. [S.l.]: Springer, 2020. p. 3–12.

GOJARE, S.; JOSHI, R.; GAIGAWARE, D. Analysis and design of Selenium Webdriver automation testing framework. **Procedia Computer Science**, v. 50, n. 1, p. 341–346, 2015.

GOOGLE Ads. 2021. <https://support.google.com/google-ads/answer/9004364?hl=pt-BR>. Accessed: 2021-11-10.

GS1 Brasil. 2021. <https://www.gs1br.org/faq/o-que-e-epc>. Accessed: 2021-11-22.

HOLMES, A.; KELLOGG, M. Automating functional tests using Selenium. In: IEEE. **Agile Conference (AGILE'06)**. [S.l.], 2006. p. 6.

IEEE STD 829-2008. **Standard for Software and System Test Documentation**. [S.l.], 2008. 1-150 p. Disponível em: <https://ieeexplore.ieee.org/abstract/document/6976080/references\#references>.

ISLAM, M. N.; QUADRI, S. M. K. Framework for automation of cloud-application testing using Selenium (FACTS). **Advances in Science, Technology and Engineering Systems Journal**, v. 5, n. 1, p. 226–232, 2020.

KUMAR, A.; SAXENA, S. Data driven testing framework using Selenium Webdriver. **International Journal of Computer Applications**, v. 118, n. 18, p. 1–6, 2015.

MOLINARI, L. **Testes de Software: Produzindo sistemas melhores e mais confiáveis**. 4. ed. São Paulo: Erica, 2008.

ORACLE Java Documentation. 2021. <https://docs.oracle.com/javase/8/docs/technotes/guides/language/varargs.html>. Accessed: 2021-11-01.

PFLEEGER, S. L. **Engenharia de Software: Teoria e Prática**. 2. ed. São Paulo: Pearson Prentice Hall Brasil, 2004. p. 560.

PRESSMAN, R. S.; MAXIM, B. R. **Software Engineering: a practitioner's approach**. 8. ed. [S.l.]: McGrawHill Education, 2014. 941 p.

RAMYA, P.; SINDHURA, V.; SAGAR, P. V. Testing using selenium web driver. In: **Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)**. [S.l.: s.n.], 2017. p. 1–7.

ROLLWAGEN, A. F. e. a. Comparativo entre ferramentas de automação de testes de software para sistemas web. **Salão do Conhecimento**, v. 6, n. 6, p. 1–11, 2020.

ROMANINI, I. Z.; SOTTO, E. C. S. Selenium web driver na evolução dos testes manuais. **Revista Interface Tecnológica**, v. 16, n. 2, p. 112–123, 2019.

SARAVANAN, K.; PRASAD, E. P. C. Open source software test automation tools: A competitive necessity. **Scholedge International Journal of Management Development**, v. 3, n. 6, p. 103–110, 2016.

SILVA, P. C. B.; ALVES, T. S.; BRUNO, E. A. Automação de testes funcionais. **Revista de Ciência Exatas e Tecnologia, Anhanguera Educacional Ltda.**, v. 6, n. 6, p. 113–133, 2011.

SINGH, J.; SHARMA, M. Performance evaluation and comparison of Sahi Pro and Selenium Webdriver. **International Journal of Computer Applications**, v. 129, n. 8, p. 23–26, 2015.

TERRA, R. Tratativa de falhas e automação de testes funcionais: O caso do Wms GTI Plug. In: **XI Brazilian Conference on Software: Theory and Practice (CBSoft), Industry Track**. [S.l.: s.n.], 2020. p. 1–4.

VILA, E.; NOVAKOVA, G.; TODOROVA, D. Automation testing framework for web applications with Selenium WebDriver: Opportunities and threats. In: **International Conference on Advances in Image Processing (ICAIP)**. [S.l.: s.n.], 2017. p. 144–150.

WALKER, J. T. **Enhancing the Test and Evaluation Process: Implementing Agile Development, Test Automation, and Model-Based Systems Engineering Concepts**. Tese (Doutorado) — Colorado State University, 2020.

Apêndices

A. Códigos do Selenium *record and playback*

Este apêndice ilustra os testes de unidade completos – Códigos 17 a 21 – respectivamente relacionados às Atividades 1 a 5 desenvolvidas com o recurso *record and playback* do Selenium.

Código 17. Selenium: Criação de um produto (Atividade 1)

```
@Test
public void criarProduto() {
    driver.get("https://hml.gtiplug.com.br/wms/login.jsp");
    driver.manage().window().maximize();
    WebDriverWait wait = new WebDriverWait(driver, 5);

    //Efetuando Login
    driver.findElement(By.id("j_username")).click();
    driver.findElement(By.id("j_username")).sendKeys("cabtec@testeautomatizado");
    driver.findElement(By.id("j_password")).click();
    driver.findElement(By.id("j_password")).sendKeys("cabtec2018");
    driver.findElement(By.id("btnLogin")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated(By.cssSelector("body > div.wrapper.ng-scope >
        div.content-wrapper > section.content-header > h1")));
    assertEquals(driver.findElement(By.cssSelector("body > div.wrapper.ng-scope > div.content-wrapper >
        section.content-header > h1")).getText(), "DashboardTela inicial");

    //Criando produto
    driver.findElement(By.id("modulemenu_dadosBasicos")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated( By.id("menu_produtos")));
    driver.findElement(By.id("menu_produtos")).click();
    driver.findElement(By.cssSelector("#btNovo > .botao")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated(
    By.id("formCenter: produtosTab:inputNome")));
    driver.findElement(By.id("formCenter:produtosTab:inputNome")).click();
    driver.findElement(By.id("formCenter:produtosTab:inputNome")).sendKeys("Teste DeProduto01");
    driver.findElement(By.id("formCenter:produtosTab:inputIdentExterna")).click();
    driver.findElement( By.id("formCenter:produtosTab:inputIdentExterna")).sendKeys("produto001");
    driver.findElement(By.cssSelector("#btSel_fornecedor > span")).click();
    WebElement element1 = driver.findElement(By.cssSelector("#btSel_fornecedor > span"));
    Actions builder = new Actions(driver);
    builder.moveToElement(element).perform();
    WebElement element2 = driver.findElement(By.tagName("body"));
    Actions builder = new Actions(driver);
    builder.moveToElement(element, 0, 0).perform();

    driver.findElement(By.cssSelector("#frmSelPJ\\3A btPesquisarPJ > .ui-button-text")).click();
    WebDriverWait wait2 = new WebDriverWait(driver, 2);
    wait2.until(ExpectedConditions.visibilityOfElementLocated(
        By.xpath("/html/body/div[1]/div[1]/div/span/div[2]/div[2]/form/div[3]/div/
        div[4]/div[2]/table/tbody/tr/td[2]")));
    assertEquals(driver.findElement(By.cssSelector("#frmSelPJ\\:dtPJ_data >
        tr.ui-widget-content.ui-datatable-even.ui-datatable-selectable > td:nth-child(2)")).getText(), "Pessoa
        Juridica Carga Ltda");
    driver.findElement( By.xpath("/html/body/div[1]/div[1]/div/span/div[2]/div[2]/form/div[3]/div/
        div[4]/div[2]/table/tbody/tr/td[2]")).click();
    driver.findElement(By.linkText("Informações adicionais")).click();
    driver.findElement(By.cssSelector(".col-sm-3:nth-child(1)")).click();
    driver.findElement( By.id("formCenter:produtosTab:inputLimitePalletVirtual")).click();
    driver.findElement( By.id("formCenter:produtosTab:inputLimitePalletVirtual")).sendKeys("50");
    driver.findElement(By.cssSelector(".fa-save")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated( By.id("formCenter:msgs_formCenter")));
    assertEquals(driver.findElement(By.id("formCenter:msgs_formCenter")).getText(), "Registro incluído com
        sucesso.");
}
```

Código 18. Selenium: Importação de uma pessoa jurídica (Atividade 2)

```
@Test
public void ImportarPJ() throws InterruptedException {
    WebDriverWait wait = new WebDriverWait(driver, 5);
    driver.get("https://hml.gtiplug.com.br/wms/");
    driver.manage().window().maximize();

    //Importando Pessoa Jurídica
    driver.findElement(By.id("modulemenu_importacoes")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated( By.id("menu_importarPessoa")));
    driver.findElement(By.id("menu_importarPessoa")).click();
    driver.findElement(By.id("formCenter:arquivoImportacao_input"))
        .sendKeys("C:\\Users\\turok\\Downloads\\pjModel.csv");
    driver.findElement(By.id("btImportar")).click();
    assertEquals(driver.findElement(By.id("formCenter:msgs_formCenter")).getText(), "Solicitação de importação
        enviada com sucesso. Acompanhe o andamento na tabela abaixo que se atualizará automaticamente a cada
        10 segundos.");
    Thread.sleep(13000);
    assertEquals(driver.findElement(By.cssSelector("#formCenter\\:dtResult_data >
        tr.ui-widget-content.ui-datatable-odd > td:nth-child(3)")).getText(), "Concluída");
}
```

Código 19. Selenium: Impressão de dez unidades de um produto a ser criado, associado e atribuído a um armazém (Atividade 3)

```
@Test
public void CriarProdutoAssociarEPCAtrAtvImprEt() throws InterruptedException {

    /*Criando um Produto */

    //Associando EPC
    driver.findElement(By.id("modulemenu_dadosBasicos")).click();
    driver.findElement(By.id("menu_associarEpc")).click();
    driver.findElement(By.cssSelector("#btNovo > .botao")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated(
          By.cssSelector("#formCenter\\:associarTab\\:btSelecionar")));
    driver.findElement( By.cssSelector("#formCenter\\:associarTab\\:btSelecionar")).click();
    driver.findElement(By.id("frmSelProd:inputIdentExterna")).click();
    driver.findElement (By.id("frmSelProd:inputIdentExterna")).sendKeys("ProdutoAtv03");
    driver.findElement(By.cssSelector("#frmSelProd\\3A btPsqBuscar > .ui-button-text")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated( By.cssSelector("#frmSelProd\\:dtProduto_data > tr
          > td:nth-child(2)")));
    assertEquals(driver.findElement(By.cssSelector("#frmSelProd\\:dtProduto_data > tr:nth-child(1) >
          td:nth-child(1)")).getText(), "Produto03");
    driver.findElement(By.cssSelector("#frmSelProd\\:dtProduto_data > tr > td:nth-child(2)")).click();
    driver.findElement(By.cssSelector("#btSalvar > .botao")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated( By.cssSelector("#formCenter\\:msgs_formCenter >
          div > ul > li > span")));
    assertEquals(driver.findElement(By.id("formCenter:msgs_formCenter")) .getText(), "Registro incluído com
          sucesso.");

    WebElement element1 = driver.findElement(By.cssSelector("#btSalvar > .botao"));
    Actions builder = new Actions(driver);
    builder.moveToElement(element).perform();
    WebElement element2 = driver.findElement(By.tagName("body"));
    Actions builder = new Actions(driver);
    builder.moveToElement(element, 0, 0).perform();

    //Associando Ativo-Endereço
    driver.findElement(By.id("modulemenu_configuracao")).click();
    driver.findElement(By.id("menu_associarAtivoEnd")).click();
    driver.findElement(By.cssSelector("#btNovo > .botao")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated( By.id("formCenter:btPsqProdDg")));
    assertEquals(driver.findElement( By.id("formCenter:j_id_2e:comboArmazem_label")).getText(), "Armazém com
          Porta-Pallet Carga");
    driver.findElement(By.id("formCenter:btPsqProdDg")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated( By.id("frmSelProd:inputIdentExterna")));
    driver.findElement(By.id("frmSelProd:inputIdentExterna")).click();
    driver.findElement(By.id("frmSelProd:inputIdentExterna")) .sendKeys("ProdutoAtv03");
    driver.findElement(By.cssSelector("#frmSelProd\\3A btPsqBuscar > .ui-button-text")).click();

    WebElement element3 = driver.findElement(By.cssSelector("#frmSelProd\\3A btPsqBuscar > .ui-button-text"));
    Actions builder = new Actions(driver);
    builder.moveToElement(element).perform();
    WebElement element4 = driver.findElement(By.tagName("body"));
    Actions builder = new Actions(driver);
    builder.moveToElement(element, 0, 0).perform();

    assertEquals(driver.findElement(By.cssSelector("#frmSelProd\\:dtProduto_data > tr >
          td:nth-child(1)")).getText(), "Produto03");
    driver.findElement(By.cssSelector(".ui-widget-content > td:nth-child(2)")).click();
    Thread.sleep(1000);
    driver.findElement(By.id("formCenter:treeLayout:0:lblNode")).click();
    driver.findElement(By.id("formCenter:treeLayout:2:lblNode")).click();
    driver.findElement(By.id("formCenter:treeLayout:3:lblNode")).click();
    driver.findElement(By.id("btSalvar")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated( By.cssSelector("#formCenter\\:msgs_formCenter >
          div > ul > li > span")));
    assertEquals(driver.findElement(By.id("formCenter:msgs_formCenter")).getText(), "Registro incluído com
          sucesso.");
    Thread.sleep(1000);

    //Imprimindo 10 etiquetas
    driver.findElement(By.id("modulemenu_dadosBasicos")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated( By.id("menu_imprimir")));
    driver.findElement(By.id("menu_imprimir")).click();
    driver.findElement(By.id("formCenter:inputIdentExterna")).click();
    driver.findElement(By.id("formCenter:inputIdentExterna")) .sendKeys("ProdutoAtv03");
    driver.findElement(By.cssSelector("#btPesquisar > .botao")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated( By.cssSelector(".ui-widget-content >
          td:nth-child(2)")));
    assertEquals(driver.findElement(By.cssSelector("#formCenter\\:dtResult_data > tr >
          td:nth-child(1)")).getText(), "Produto03");
    driver.findElement(By.cssSelector(".ui-widget-content > td:nth-child(2)")).click();
    Thread.sleep(1000);
    jse.executeScript("window.scrollBy(0,250)");
    wait.until(ExpectedConditions.visibilityOfElementLocated(
          By.id("formCenter:imprimirCodificar:qtdeEtiquetas_input")));
    driver.findElement( By.id("formCenter:imprimirCodificar:qtdeEtiquetas_input")).click();
    driver.findElement( By.id("formCenter:imprimirCodificar:qtdeEtiquetas_input")).sendKeys("10");
    driver.findElement(By.cssSelector("#formCenter\\3 AimprimirCodificar\\3A btImprimirEtiquetas >
          .ui-button-text")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated( By.cssSelector("#formCenter\\:msgs_formCenter >
          div > ul > li > span")));
    String nomeProduto = driver.findElement(By.id("formCenter:inputProduto")).getAttribute("value");
    assertEquals(driver.findElement(By.id("formCenter:msgs_formCenter")).getText(), "Etiqueta impressa com
          sucesso.");
}
```

Código 20. Selenium: Recebimento e armazenamento de dez unidades de um produto (Atividade 4)

```
@Test
public void ReceberEtiquetaseArmazenar() throws InterruptedException {
    /* Etapa de Login */
    /* Criacao de produto */
    /* Associa EPC */
    /* Atribui produto a um endereço */

    //Agendar Recebimento
    driver.findElement(By.id("modulemenu_operacao")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated( By.id("menu_recebimento")));
    driver.findElement(By.id("menu_recebimento")).click();
    driver.findElement(By.cssSelector("#btNovo")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated(
          By.cssSelector("#formCenter\\:recebimentoTabView\\:inputIdentExterna")));
    driver.findElement(By.cssSelector("#formCenter\\:recebimentoTabView\\: inputIdentExterna")).click();
    driver.findElement(By.cssSelector("#formCenter\\:recebimentoTabView\\:
          inputIdentExterna")).sendKeys(IdentExt);
    driver.findElement(By.id("btSel_remetente")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated( By.id("frmSelPJ:btPesquisarPJ")));
    driver.findElement(By.id("frmSelPJ:btPesquisarPJ")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated( By.cssSelector("#frmSelPJ\\:dtPJ_data > tr >
          td:nth-child(2)")));
    assertEquals(driver.findElement(By.cssSelector("#frmSelPJ\\:dtPJ_data > tr > td:nth-child(2)")).getText(),
          "Pessoa Juridica Carga Ltda");
    driver.findElement(By.cssSelector("#frmSelPJ\\:dtPJ_data > tr > td:nth-child(2)")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated(
          By.id("formCenter:recebimentoTabView:buttonAutorizado:0")));
    driver.findElement(By.cssSelector("#formCenter\\:recebimentoTabView\\: buttonAutorizado >
          div.ui-button.ui-widget.ui-state-default.ui-button-text- only.ui-corner-left")).click();
    driver.findElement(By.id("formCenter:recebimentoTabView:comboQtdEtapas_label")) .click();
    driver.findElement(By.id("formCenter:recebimentoTabView:comboQtdEtapas_1")) .click();
    wait.until(ExpectedConditions.visibilityOfElementLocated(
          By.id("formCenter:recebimentoTabView:dtEtapas:0:comboArmazem_label")));
    driver.findElement(By.id("formCenter:recebimentoTabView:dtEtapas:0:
          dataAgendamento_input")).sendKeys("25/03/2021 15:23");
    driver.findElement(By.id("formCenter:recebimentoTabView:dtEtapas:0: comboArmazem_label")).click();
    driver.findElement(By.id("formCenter:recebimentoTabView:dtEtapas:0: comboArmazem_1")).click();
    driver.findElement(By.id("btSalvar")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated(
          By.id("formCenter:recebimentoTabView:dtEtapas:0:btAdProdutos")));
    assertEquals(driver.findElement(By.id("formCenter:msgs_formCenter")) .getText(), "Registro incluído com
          sucesso.");
    driver.findElement(By.id("formCenter:recebimentoTabView:dtEtapas:0: btAdProdutos")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("btPsqProdDg")));
    driver.findElement(By.id("btPsqProdDg")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated( By.id("frmSelProd:inputIdentExterna")));
    driver.findElement(By.id("frmSelProd:inputIdentExterna")).sendKeys(IdentExt);
    driver.findElement(By.id("frmSelProd:btPsqBuscar")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated( By.cssSelector("#frmSelProd\\:dtProduto_data > tr
          > td:nth-child(2)")));
    driver.findElement(By.cssSelector("#frmSelProd\\:dtProduto_data > tr > td:nth-child(2)")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated(
          By.id("frmAddProd:dtAtivo:0:qtdeInformada_input")));
    driver.findElement(By.id("frmAddProd:dtAtivo:0:qtdeInformada_input")) .sendKeys("10");
    driver.findElement(By.id("frmAddProd:btSalvarListaProd")).click();
    Thread.sleep(1000);
    driver.findElement(By.id("btSalvar")).click();
    Thread.sleep(3000);
    assertEquals(driver.findElement(By.id("formCenter:msgs_formCenter")).getText(), "Registro alterado com
          sucesso.");


    //Coletar ID's
    driver.findElement(By.id("modulemenu_relatorio")).click();
    Thread.sleep(1000);
    WebElement DIVelement = driver.findElement(
          By.xpath("/html/body/div[1]/aside[1]/div/section/ul/li[8]/ul/li[8]/a"));
    jse.executeScript("arguments[0].scrollIntoView(true)", DIVelement);
    driver.findElement(By.id("menu_relEtiqueta")).click();
    driver.findElement(By.cssSelector("#formCenter\\3A btPsqProdDg > .ui-button-text")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated( By.id("frmSelProd:inputIdentExterna")));
    driver.findElement(By.id("frmSelProd:inputIdentExterna")).click();
    driver.findElement(By.id("frmSelProd:inputIdentExterna")).sendKeys(IdentExt);
    driver.findElement(By.cssSelector("#frmSelProd\\3A btPsqBuscar > .ui-button-text")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated( By.cssSelector("td:nth-child(2)")));
    driver.findElement(By.cssSelector("td:nth-child(2)")).click();

    driver.findElement(By.cssSelector("#btPesquisar > .botao")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated( By.cssSelector("#formCenter\\:dtResult_data >
          tr:nth-child(1) > td:nth-child(1)")));
    jse.executeScript("window.scrollBy(0,250)");
    String gti[] = new String[11];
    int i;
    for(i = 1; i<=10; i++) {
        gti[i] = driver.findElement(By.cssSelector("#formCenter\\:dtResult_data > tr:nth-child("+i+") >
              td:nth-child(1)")).getText();
    }


    //Iniciar Recebimento
    driver.findElement(By.id("modulemenu_operacao")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated( By.id("menu_iniciarRecebimento")));
```

```
    driver.findElement(By.id("menu_iniciarRecebimento")).click();
    driver.findElement(By.id("formCenter:j_id_2d:inputIdentificadorExterno")) .sendKeys(IdentExt);
    driver.findElement(By.id("btPesquisar")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated( By.cssSelector("#formCenter\\:dtResult_data > tr
         > td:nth-child(3)")));
    assertEquals(driver.findElement(By.cssSelector("#formCenter\\:dtResult_data > tr >
         td:nth-child(2)")).getText(), "Pessoa Juridica Carga");
    driver.findElement(By.cssSelector("#formCenter\\:dtResult_data > tr > td:nth-child(3)")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated( By.id("portalBtIniciar")));
    driver.findElement(By.id("portalBtIniciar")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated( By.id("btEntradaManual")));
    driver.findElement(By.id("btEntradaManual")).click();
    for(i = 1; i<=10; i++) {
         driver.findElement(By.id("formEntradaManual:tagsEntradaManual_tagsinput")) .sendKeys(gti[i]);
         driver.findElement(By.id("formEntradaManual:tagsEntradaManual_tagsinput")) .sendKeys(Keys.ENTER);
    }

    driver.findElement(By.id("formEntradaManual:btEnviar")).click();
    Thread.sleep(1000);
    driver.findElement(By.id("formEntradaManual:btFechar")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated( By.id("portalBtFinalizar")));
    driver.findElement(By.id("portalBtFinalizar")).click();
    Thread.sleep(1000);
    assertEquals(driver.findElement(By.id("formCenter:msgs_formCenter")).getText(), "finalizado");

    //Armazenar Etiquetas
    driver.findElement(By.id("modulemenu_operacao")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("menu_armazenar")));
    driver.findElement(By.id("menu_armazenar")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated( By.id("formCenter:inputIdentificadorExterno")));
    driver.findElement(By.id("formCenter:inputIdentificadorExterno")) .sendKeys(IdentExt);
    driver.findElement(By.id("btPesquisar")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated( By.cssSelector("#formCenter\\:dtResult_data > tr
         > td:nth-child(3)")));
    driver.findElement(By.cssSelector("#formCenter\\:dtResult_data > tr > td:nth-child(3)")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated( By.id("formCenter:btArmazenarTodos")));
    driver.findElement(By.id("formCenter:btArmazenarTodos")).click();
    assertEquals(driver.findElement(By.id("formCenter:msgs_formCenter")).getText(), "Armazenamento concluído
         com sucesso");
}
```

Código 21. Selenium: Expedição de dez unidades de um produto (Atividade 5)

```
@Test
public void ExpedirEtiquetas() throws InterruptedException {

    /* Etapa de login */
    /* Criação de produto */
    /* Associar EPC */
    /* Atribuir produto a um endereço */

    //Planejar Expedição
    driver.findElement(By.id("modulemenu_operacao")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("menu_expedicao")));
    driver.findElement(By.id("menu_expedicao")).click();
    driver.findElement(By.cssSelector("#btNovo > .botao")).click();
    driver.findElement(By.id("formCenter:expedicaoTabView:inputIdentExterna")) .sendKeys("Expedicao11");
    driver.findElement(By.cssSelector("#btSel_remetente > span")).click();
    driver.findElement(By.cssSelector("#frmSelPJ\\3A btPesquisarPJ > .ui-button-text")).click();
    driver.findElement(By.cssSelector(".ui-widget-content > td:nth-child(2)")).click();
    driver.findElement(By.cssSelector(".ui-state-hover > .ui-icon")).click();
    driver.findElement(By.id("formCenter:expedicaoTabView:combMeioTransporte_4")) .click();
    driver.findElement(By.cssSelector(".ui-state-hover > .ui-icon")).click();
    driver.findElement(By.id("formCenter:expedicaoTabView:comboQtdEtapas_1")).click();
    driver.findElement(By.cssSelector(".ui-state-hover > .ui-button-text")).click();
    driver.findElement(By.cssSelector(".ui-state-hover > .ui-icon")).click();
    driver.findElement(By.id("formCenter:expedicaoTabView:dtEtapas:0: comboArmazem_1")).click();
    driver.findElement(By.cssSelector(".ui-state-focus > .ui-button-text")).click();
    driver.findElement(By.linkText("26")).click();
    driver.findElement(By.id("formCenter:expedicaoTabView:dtEtapas")).click();
    driver.findElement(By.cssSelector("#btSalvar > .botao")).click();
    driver.findElement(By.cssSelector("#formCenter\\3A expedicaoTabView\\3A dtEtapas\\3A 0\\3A btAdProdutos >
         .ui-button-text")).click();
    driver.findElement(By.cssSelector("#btPsqProdDg > .botao")).click();
    driver.findElement(By.id("frmSelProd:inputIdentExterna")).click();
     ver.findElement(By.id("frmSelProd:inputIdentExterna")).sendKeys("ProdutoAtv05");
    driver.findElement(By.cssSelector("#frmSelProd\\3A btPsqBuscar > .ui-button-text")).click();
    driver.findElement(By.cssSelector(".ui-datatable-selectable > td:nth-child(2)")).click();
    driver.findElement(By.id("frmAddProd:dtAtivo:0:qtdeInformada_input")).click();
    driver.findElement(By.cssSelector(".ui-widget-content > .input100:nth-child(7)")).click();
    driver.findElement(By.id("frmAddProd:dtAtivo:0:qtdeInformada_input")) .sendKeys("10");
    driver.findElement(By.id("dgListaProd")).click();
    driver.findElement(By.cssSelector("#frmAddProd\\3A btSalvarListaProd > .ui-button-text")).click();
    Thread.sleep(3000);
    assertEquals(driver.findElement(By.id("formCenter:msgs_formCenter")).getText(), "Registro alterado com
         sucesso.");

    //Separar produtos
    driver.findElement(By.cssSelector("#modulemenu_operacao > .ng-binding")).click();
    driver.findElement(By.id("menu_separar")).click();
    driver.findElement(By.id("formCenter:j_id_2a:inputIdentificadorExterno")).click();
    driver.findElement(By.id("formCenter:j_id_2a:inputIdentificadorExterno")) .sendKeys("Expedicao11");
    driver.findElement(By.cssSelector("#btPesquisar > .botao")).click();
    driver.findElement(By.cssSelector("td:nth-child(4)")).click();
```

```
    Thread.sleep(1000);
    for(i = 1; i<=10; i++) {
         driver.findElement(By.id("formCenter:tagsEntradaManual_tag")) .sendKeys(gti[i]);
         driver.findElement(By.id("formCenter:tagsEntradaManual_tag")) .sendKeys(Keys.ENTER);
    }
    driver.findElement(By.cssSelector("#formCenter\\3A btConfirmarGTI > .ui-button-text")).click();
    assertEquals(driver.findElement(By.id("formCenter:msgs_formCenter")).getText(), "foram separados com
          sucesso.");

    //Expedir 10 etiquetas
    driver.findElement(By.id("modulemenu_operacao")).click();
    driver.findElement(By.id("menu_iniciarExpedicao")).click();
    driver.findElement(By.id("formCenter:j_id_2a:inputIdentificadorExterno")).click();
    driver.findElement(By.id("formCenter:j_id_2a:inputIdentificadorExterno")) .sendKeys("Expedicao11");
    driver.findElement(By.cssSelector("#btPesquisar > .botao")).click();
    driver.findElement(By.cssSelector(".ui-widget-content > td:nth-child(2)")).click();
    driver.findElement(By.id("portalBtIniciar")).click();
    driver.findElement(By.cssSelector(".fa-edit")).click();
    driver.findElement(By.cssSelector("#formEntradaManual\\3A btGerar > .ui-button-text")).click();
    driver.findElement(By.cssSelector("#formEntradaManual\\3A btEnviar > .ui-button-text")).click();
    driver.findElement(By.cssSelector("#formEntradaManual\\3A btFechar > .ui-button-text")).click();
    driver.findElement(By.cssSelector(".fa-stop")).click();
    wait.until(ExpectedConditions.visibilityOfElementLocated( By.id("formCenter:msgs_formCenter")));
    assertEquals(driver.findElement(By.id("formCenter:msgs_formCenter")).getText(), "Expedição concluída com
          sucesso.");
}
```