Padronização e reciclagem de códigos em um *Marketplace* utilizando *Gherkin*: um estudo de caso na *Agro2Business* com testes automatizados *RSpec*, *Capybara* e *Page Objects*

Ana Paula Ferreira Barroso de Castro[1], Gabriel Cardoso dos Santos Herculino[1], Victor Hugo Bustamonte Mendonça[1], Carlos Alberto da Silva

[1]Curso de Bacharelado em Sistemas de Informação – Faculdade de Computação (FACOM)
Universidade Federal de Mato Grosso do Sul (UFMS).
Av. Costa e Silva, s/n. - Bairro Universitário - CEP 79070-900 - Campo Grande - MS.

{ana_barroso, gabriel_cardoso, victor_bustamonte, carlos.silva}@ufms.br

Abstract. *To meet customer expectations and maintain competitiveness, software quality is essential in a Marketplace. This article explores an approach to document, standardize, and recycle code using the Cucumber tool with the Gherkin language in conjunction with RSpec, Capybara, and the Design Pattern: Page Objects. The presented case study describes how this approach was implemented at Agro2Business, and the results demonstrate that the use of Gherkin facilitated code understanding for individuals without software development knowledge and simplified the implementation of automated tests that validate the software's compatibility with the company's business rules.*

Resumo. *Para atender às expectativas dos clientes e manter a competitividade, a qualidade do software é essencial em um Marketplace. Este artigo explora uma abordagem para documentar, padronizar e reciclar códigos, utilizando a ferramenta Cucumber com a linguagem Gherkin em conjunto com RSpec, Capybara e o Design Pattern: Page Objects. O estudo de caso apresentado descreve como essa abordagem foi implantada na empresa Agro2Business, os resultados obtidos demonstram que a utilização do Gherkin facilitou o entendimento do código por pessoas sem conhecimento de desenvolvimento de software e simplificou a implementação de testes automatizados, que validam a compatibilidade do software com as regras de negócio da empresa.*

1. Introdução

A *internet* fez-se cada vez mais presente em nossas vidas nos últimos anos, desse modo, surgiram novas formas de comércio que exploram a facilidade e a comodidade que ela oferece. Um exemplo dessa vantagem é o *Marketplace*, pois permite que compradores e vendedores de diferentes partes do mundo se conectem e realizem negócios de forma rápida e eficiente [Laudon and Traver 2018].

Segundo [Laudon and Traver 2018], os mercados *online* mitigam limitações geográficas e temporais do comércio tradicional, disponibilizando uma variedade maior de produtos e valores mais acessíveis, fazendo com que esse modelo de negócio se popularizasse nos últimos anos, gerando, assim, um aumento das transações e tornando-se alvos frequentes de ataques cibernéticos, incluindo furto de dados pessoais e financeiros.

O sucesso de um *Marketplace* depende em grande parte da qualidade do seu *software*, que precisa ser confiável, eficiente e fácil de ser utilizado. Dessa forma, é essencial a implementação de medidas de segurança robustas no *software*, como um mecanismo para garantir a proteção das informações confidenciais dos clientes e manter a sua integridade [Sommerville 2019].

Desenvolver e manter um *software* de alta qualidade é uma tarefa complexa que requer práticas de documentação, padronização, reciclagem e uma comunicação clara entre os membros da equipe [Sommerville 2019]. Existem diversas ferramentas que podem ser utilizadas com esse intuito, logo,

escolher a ferramenta que vai atender melhor as necessidades da empresa é uma tarefa minuciosa, na qual muitos fatores precisam ser levados em consideração.

Nesse artigo, exploramos uma abordagem para o desenvolvimento de testes automatizados utilizando a linguagem *Gherkin*, e como ela foi implementada na empresa *Agro2Business* [Agro2Business 2023], uma plataforma digital que tem como principal produto um *Marketplace* agropecuário. Demonstramos como o uso do *Gherkin* facilitou a implementação de testes automatizados, que validam a compatibilidade do *software* com as regras de negócio da empresa e buscam garantir sua integridade.

2. Fundamentos Teóricos

Esta seção descreve os fundamentos teóricos utilizados durante a realização deste artigo.

2.1. *Behavior Driver Development*

O *Behavior Driver Development* (BDD) [Fox 2016], ou Desenvolvimento Orientado a Comportamento, é uma técnica de desenvolvimento de *software* que se concentra em analisar os requisitos, descrevendo-os em testes executáveis, que são essenciais para garantir a integridade do *software*.

O BDD permite aprimorar a comunicação com o cliente e o entendimento das regras de negócio, preparando inicialmente o teste de aceitação para o recurso, que será implementado em seguida. Essa técnica é amplamente utilizada na automatização de testes por equipes de desenvolvimento ágil [Adzic 2011].

2.2. Linguagem *Gherkin*

O *Gherkin* [Cucumber 2019a] possui a premissa de ser uma linguagem simples e legível por humanos, possibilitando sua utilização por membros do projeto que não são desenvolvedores, envolvendo toda a equipe em um processo colaborativo, ajudando o *software* desenvolvido a atender os requisitos do cliente de maneira eficiente e eficaz [Ehrenfried 2019].

Os cenários são escritos em um formato declarativo, com mais de sessenta idiomas disponíveis, sendo recomendado utilizar a língua falada pelos usuários e especialistas de domínio. A sintaxe do *Gherkin* é composta por uma série de palavras-chave, em português, “Dado”, “Quando” e “Então” para descrever as etapas que o sistema deve seguir. Essa gramática deriva das Histórias de Usuário do estilo *Given – When – Then* [Ehrenfried 2019].

As palavras-chave mais comuns são:

- Dado: descreve o estado inicial do cenário de teste;
- Quando: descreve a ação que está sendo testada;
- E: adiciona uma nova etapa ao cenário de teste, seguindo a palavra-chave anterior;
- Mas: adiciona uma nova etapa ao cenário de teste, com uma condição de exceção.
- Então: é a verificação do comportamento do sistema em resposta à ação realizada no passo anterior.

2.3. Ferramenta *Cucumber*

O *Cucumber* [Cucumber 2019b] é uma ferramenta de automação de testes, muito utilizada no desenvolvimento de *software*, sua implementação utiliza a linguagem *Gherkin* para descrever os cenários, ele foi implementado para diversas linguagens de programação como o *Ruby* [Ruby 2023], *Java* [Java 2022], *Javascript* [JavaScript 2023] e outras [Wynne and Hellesøy 2012]. Assim sendo, o *Cucumber* pode ser considerado como uma documentação de requisitos funcionais extremamente detalhada e precisa, em uma linguagem bastante próxima da natural [Baraúna 2014].

Ao utilizar o *Cucumber* para escrever testes, é preciso descrever as funcionalidades do sistema ao invés de focar em classes ou métodos específicos do código. O teste pode resultar em “Falha”, “Não Implementado” ou “Passou”. Caso o resultado seja “Não Implementado”, o desenvolvedor precisa definir o que será executado nas linhas correspondentes [Ehrenfried 2019].

2.4. Ferramenta *Capybara*

Capybara [Capybara 2011] é uma biblioteca de automação de testes que simula as interações do usuário com aplicações *web*. A ideia é tentar reproduzir as ações do usuário ao utilizar um navegador *web*, acelerando a escrita dos testes automatizados. Essa ferramenta pode ser utilizada em algumas linguagens de programação como *Ruby*, *Python* [Python 2023], *Java*, entre outras, porém neste artigo iremos aplicar somente na linguagem *Ruby*. [Robbins 2013]

Com *Capybara* é possível realizar visitas em páginas, preenchimento de formulários, clicar em botões, ou verificar resultados esperados. As simulações dessas ações são feitas a partir de um *driver*, que pode ser configurado para utilizar diversos navegadores [Gundecha 2015] como *Firefox* [Mozilla 2023], *Chrome* [Google 2023], *Safari* [Apple 2023]. A utilização da biblioteca *Capybara* pode ser combinada com as ferramentas de automação de testes *Cucumber* e *RSpec* [Sumner 2014].

2.5. Ferramenta *RSpec*

RSpec [Rspec 2023] é um *framework* popular de testes orientado a comportamento (BDD), implementado para a linguagem *Ruby*. Ele é amplamente utilizado para escrever testes, fornecendo uma sintaxe clara e legível que ajuda os desenvolvedores a especificar o que se espera do código de forma expressiva e compreensível [Marston and Dees 2017].

O *RSpec* fornece uma estrutura flexível para a escrita de testes, permitindo que os desenvolvedores modifiquem e personalizem eles de acordo com as suas necessidades, se concentrando no sistema, ao invés de se preocuparem com os detalhes de implementação, resultando em um código mais confiável e fácil de manter [Marston and Dees 2017].

Outro aspecto importante do *RSpec* é a sua integração com outras ferramentas de desenvolvimento, como o *Rails*, o *Capybara* e o *Cucumber*, juntas, essas ferramentas ajudam a garantir a qualidade de uma aplicação em todas as etapas do desenvolvimento [Chelimsky et al. 2010].

2.6. *Page Objects (Design Pattern)*

Page Objects (Design Pattern) é uma abordagem amplamente utilizada no desenvolvimento de testes automatizados para aplicações *web*. Tendo como objetivo principal separar a lógica do teste, dos elementos da página, tornando o código modular e de fácil manutenção [Gundecha 2015].

A estrutura do padrão *Page Objects* é composta pela separação das páginas em classes e seus métodos que encapsulam a lógica de interação com os elementos. Os testes interagem somente com os métodos, não interagindo diretamente com os elementos da página, conforme ilustrado na Figura 1 [Gundecha 2015].

Este padrão é essencialmente útil em testes complexos ou de longa duração, na qual muitas vezes a manutenção pode se tornar um grande desafio [Gundecha 2015].

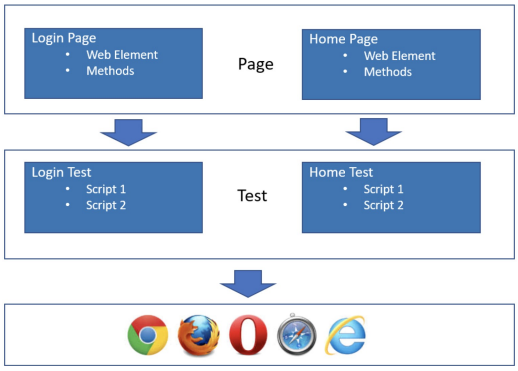

Figura 1. Page Object Model

Fonte: https://www.browserstack.com/guide/page-object-model-in-cucumber

3. Implementação e Cenários

Com o objetivo de validar as funcionalidades do *Marketplace* da *Agro2Business* [Agro2Business 2023], foram estruturados e desenvolvidos alguns cenários de teste utilizando as ferramentas descritas.

3.1. Funcionalidade de Cadastro e Login

Para o usuário possuir uma experiência completa na plataforma, é preciso cadastrar uma conta ou fazer login em uma conta existente.

Cenários propostos para essa funcionalidade:

- **Cadastro de usuário com sucesso:** cadastro com todas as informações preenchidas de forma válida, conforme Figura 2(a).
- **Cadastro incompleto:** cadastro de usuário faltando informações obrigatórias que devem ser preenchidas, conforme Figura 2(a).
- **Login com sucesso:** login definindo informações de um usuário cadastrado, conforme Figura 2(b).
- **Login inválido:** tentativa de login definindo informações inválidas para se realizar um login, conforme Figura 2(b).

Figura 2. Relatório de (a) Cadastro e de (b) Login

Fonte: Elaborado pelos autores utilizando a ferramenta Cucumber

3.2. Funcionalidade de Cadastro de Anúncio e Compra de Produto

A base do *Marketplace* consiste na criação de anúncios de produtos, que são criados pelos usuários fornecedores, e as compras desses produtos que são realizadas pelos usuários compradores. Portanto, é importante validar o correto funcionamento da etapa de cadastro de anúncios e da etapa de comprar um produto.

Cenários propostos para essa funcionalidade:

- **Cadastro de anúncio com campos obrigatórios preenchidos:** cria um anúncio preenchendo somente os campos obrigatórios, conforme Figura 3(a).
- **Cadastro de anúncio com todos os campos preenchidos:** cria um anúncio preenchendo todos os campos (obrigatórios e opcionais), conforme Figura 3(a).
- **Compra de produto com boleto bancário:** adiciona um anúncio ao carrinho e realiza a compra de um produto, conforme Figura 3(b), com a forma de pagamento Boleto Bancário.

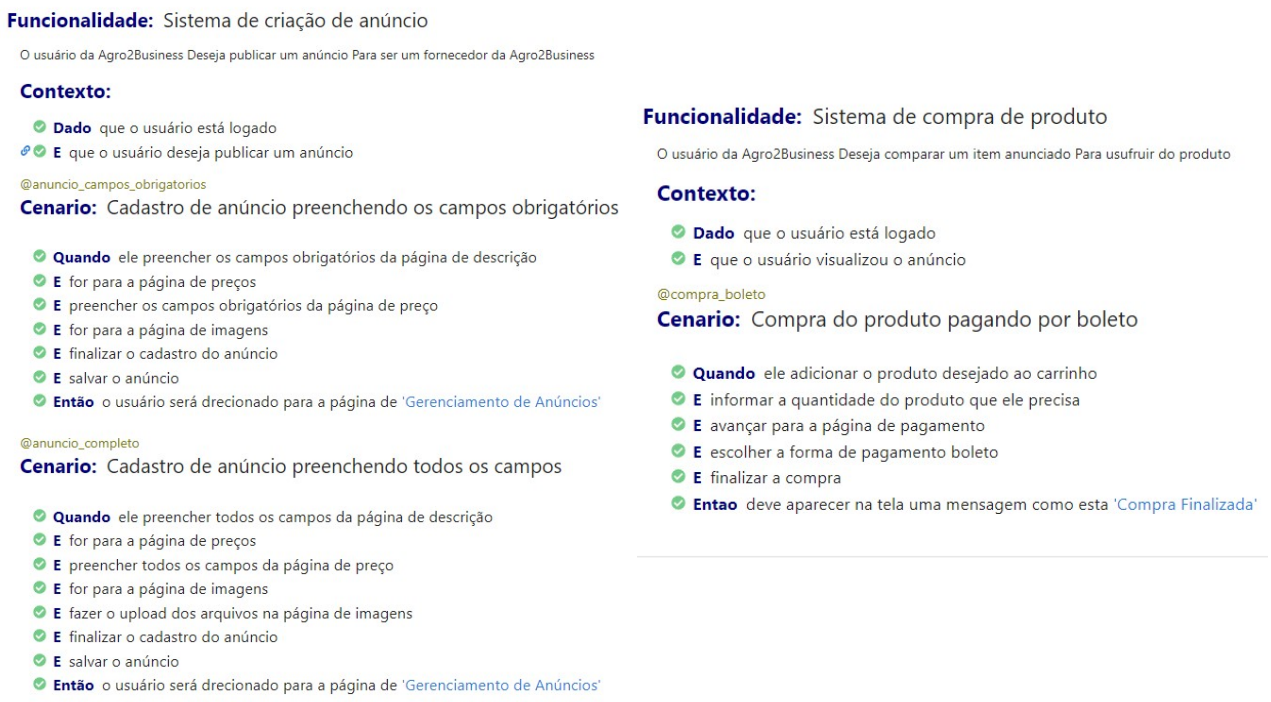

Figura 3. Relatório de (a) criação de anúncio e (b) compra de produto

Fonte: Elaborado pelos autores utilizando a ferramenta Cucumber

3.3. Funcionalidade de Controle de Acessos

Um outro ponto crucial para um *Marketplace* é a segurança das informações dos usuários. Desse modo, é importante garantir o controle de acesso a fim de evitar que informações sejam apresentadas a pessoas não autorizadas.

Cenários propostos para essa funcionalidade:

- **Controle de Acessos Indevidos:** realiza tentativas de acessos indevidos a páginas na visão de Administração e de outros usuários, conforme Figura 4.

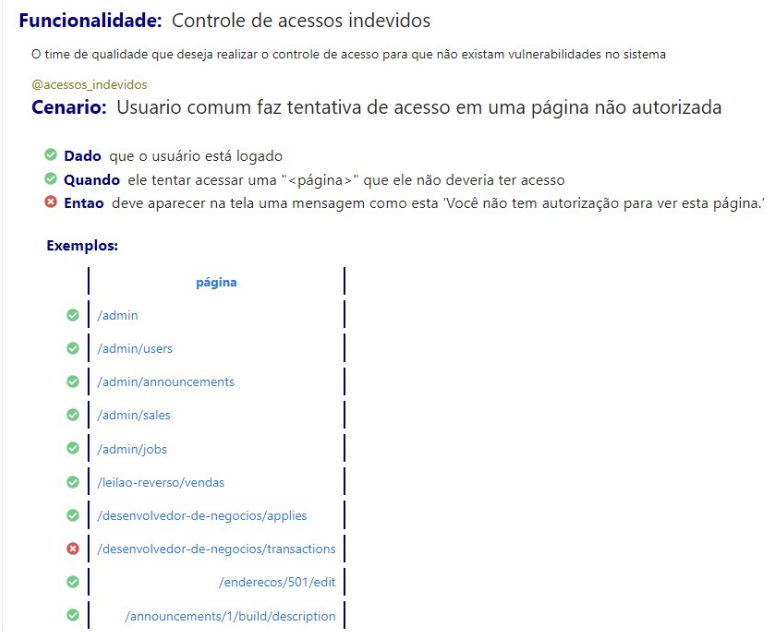

Figura 4. Relatório da Funcionalidade de Controle de Acessos

Fonte: Elaborado pelos autores utillizando a ferramenta Cucumber

4. Resultados Obtidos

Os resultados obtidos foram positivos em relação aos testes automatizados realizados no contexto de cadastro de anúncio, compra de produto, login, cadastro de usuário e segurança.

Dos 22 testes realizados, 21 foram bem-sucedidos, indicando que as funcionalidades do sistema de *Marketplace* da *Agro2Business* que reflete o bom desempenho do sistema, conforme apresentado na Figura 5. Entretanto, um teste de segurança falhou, conduzindo a reavaliação na plataforma, por parte da empresa, nas medidas de proteção adotadas.

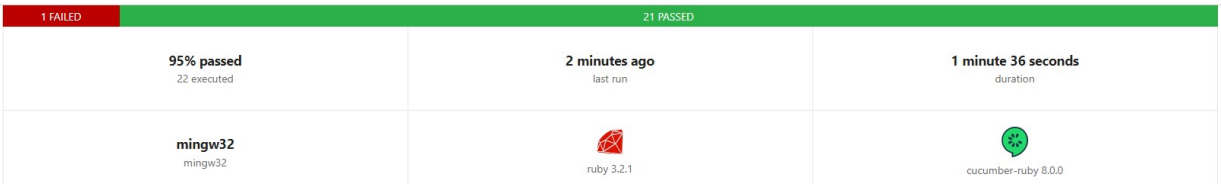

Figura 5. Resultados obtidos nos testes de todas as funcionalidades

Fonte: Elaborado pelos autores utilizando a ferramenta Cucumber

Foram desenvolvidos *pipelines* automatizados utilizando a ferramenta *GitLab CI/CD* [Gitlab 2023], que são acionados sempre que há uma alteração no código (*merge request*), garantindo a execução rápida e eficiente dos testes desenvolvidos com as ferramentas *RSpec*, *Capybara* e *Page Objects*. Isso permitiu uma maior agilidade no processo de desenvolvimento, além de garantir a qualidade do código e reduzir significativamente o tempo de testes manuais. Em resumo, os pipelines automatizados se tornaram uma peça fundamental em nosso fluxo de desenvolvimento, garantindo que as funcionalidades testadas estejam sempre em conformidade com as expectativas do cliente.

5. Trabalhos Relacionados

A automação de testes é uma prática amplamente utilizada em projetos de desenvolvimento de *software*, visando a garantia da qualidade do produto final. Diversos trabalhos têm sido desenvolvidos nessa área, explorando diferentes metodologias e ferramentas.

O artigo [Fernandes and Tomkelski 2020] descreve um estudo de caso realizado em uma empresa de *software* chamada *Softplan*. O objetivo do estudo foi implementar uma solução de automação de testes utilizando *Cucumber* para otimizar este processo e reduzir os custos associados.

No artigo [Fonseca 2021] é apresentado uma abordagem para implementar testes automatizados em aplicações *backend* no *Robot Framework* e são destacadas as vantagens da automação em relação ao método manual, como a redução de erros humanos, a economia de tempo e a melhoria da confiabilidade do sistema.

Estes artigos [Fernandes and Tomkelski 2020, Fonseca 2021] possuem em comum o objetivo de melhorar a qualidade do produto final por meio da automação de testes de *software*. Cada um deles explora diferentes ferramentas e metodologias, o que evidência a importância da diversidade de opções e da escolha da abordagem mais adequada para cada projeto.

Algumas outras ferramentas que utilizam a linguagem *Gherkin* incluem:

Robot Framework: é uma ferramenta de automação de testes de código aberto que permite a escrita de testes utilizando a linguagem Gherkin. O Robot Framework é altamente extensível e suporta a integração com diversas ferramentas e bibliotecas de teste [Robot 2023].

Behat: Uma ferramenta open-source, baseado em PHP [PHP 2023], para testes de comportamento de aplicações *web*, que utiliza a linguagem *Gherkin* para especificação de requisitos e testes [Kudryashov 2016].

SpecFlow: Uma ferramenta open-source, baseado em C# [Microsoft 2023], para testes de comportamento de aplicações .NET, que utiliza a linguagem *Gherkin* para especificação de requisitos e testes [SpecFlow 2021].

***FitNesse*:** Uma ferramenta open-source para testes de aceitação de aplicações *web* baseada em *Java*, que utiliza a linguagem *Gherkin* para especificação de requisitos e testes. No entanto, enquanto o *Cucumber* é mais focado em testes de comportamento, o *FitNesse* é projetado especificamente para testes de aceitação [Fitnesse 2022].

6. Conclusão e Trabalhos Futuros

Implementar a linguagem *Gherkin* para documentar, padronizar e reciclar códigos, por meio das ferramentas mencionadas para desenvolver testes automatizados, foi uma solução aceita por todo o time da *Agro2Business*, e trouxe resultados positivos para a empresa. Os testes automatizados executados a cada alteração no código garantiram a qualidade do *software*, permitindo uma rápida identificação e correção de possíveis falhas, além de evitar a ocorrência de problemas em produção. A implementação também permitiu que a equipe de desenvolvimento pudesse focar em outras tarefas importantes, enquanto os testes eram executados de forma automática.

Nesse artigo validamos o funcionamento e a adaptação em algumas das funcionalidades do *Marketplace* da *Agro2Business*, no entanto, como trabalho futuro, a proposta é que sejam implementados testes automatizados para outras funcionalidades que existem, e que serão desenvolvidas num futuro próximo. Ademais, é possível explorar outras ferramentas e técnicas de testes automatizados, como a simulação de carga e a automação de testes de integração e aceitação. A continuidade da implementação de testes automatizados permitirá que a *Agro2Business* possa garantir a qualidade de todo o software e manter a competitividade no mercado de *Marketplaces*.

Referências

Adzic, G. (2011). *Specification by Example: How successful teams deliver the right software*. Manning Publications Co.

Agro2Business (2023). Marketplace agro2business. `https://agro2business.com/`. Acessado em 03 de Maio de 2023.

Apple (2023). Navegador safari. `https://www.apple.com/br/safari/`. Acessado em 03 de Maio de 2023.

Baraúna, H. (2014). *Cucumber e RSpec: Construa aplicações Ruby e Rails*. Casa do Código, São Paulo, Brazil, 1st edition.

Capybara, T. (2011). Test your app with capybara. `https://teamcapybara.github.io/capybara`. Acessado em 03 de Maio de 2023.

Chelimsky, D., Astels, D., Dennis, Z., and Hellesøy, A. (2010). *The RSpec Book: Behaviour Driven Development with RSpec, Cucumber, and Friends*. Pragmatic Bookshelf.

Cucumber (2019a). Gherkin syntax. `https://cucumber.io/docs/gherkin/`. Acessado em 03 de Maio de 2023.

Cucumber (2019b). Tools techniques that elevate teams to greatness. `https://cucumber.io/`. Acessado em 03 de Maio de 2023.

Ehrenfried, H. V. (2019). Gherkin specification extension - uma linguagem de especificação de requisitos baseada em gherkin. *Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática*.

Fernandes, M. and Tomkelski, S. (2020). Automação de testes de software: estudo de caso da empresa softplan.

Fitnesse (2022). Fitnesse documentation. http://docs.fitnesse.org/FrontPage. Acessado em 29 de Abril de 2023.

Fonseca, M. A. N. (2021). Desenvolvimento de testes automatizados para backend.

Fox, S. (2016). All you need to know about behaviour-driven software. http://behaviour-driven.org/need-know-behaviour-driven-software.html. Acessado em 28 de Março de 2023.

Gitlab (2023). The devsecops platform. https://about.gitlab.com/. Acessado em 03 de Maio de 2023.

Google (2023). Navegador google chrome. https://www.google.com/intl/pt-BR/chrome/. Acessado em 03 de Maio de 2023.

Gundecha, U. (2015). *Selenium Testing Tools Cookbook*. Packt Publishing, Birmingham, UK.

Java (2022). Java. https://www.java.com/pt-BR/. Acessado em 03 de Maio de 2023.

JavaScript (2023). Javascript. https://www.javascript.com/. Acessado em 03 de Maio de 2023.

Kudryashov, K. (2016). Behat documentation. https://behat.org/en/latest/guides.html. Acessado em 29 de Abril de 2023.

Laudon, K. C. and Traver, C. G. (2018). *E-commerce: business, technology, society*. Pearson, 14th ed. edition.

Marston, M. and Dees, I. (2017). *Effective Testing with RSpec 3*. The Pragmatic Programmers, Raleigh, NC.

Microsoft (2023). Documentação do c#. https://learn.microsoft.com/pt-br/dotnet/csharp/. Acessado em 03 de Maio de 2023.

Mozilla (2023). Navegador firefox. https://www.mozilla.org/pt-BR/firefox/new/. Acessado em 03 de Maio de 2023.

PHP (2023). Php. https://www.php.net/. Acessado em 03 de Maio de 2023.

Python (2023). Python. https://www.python.org/. Acessado em 03 de Maio de 2023.

Robbins, M. (2013). *Application Testing with Capybara*. Packt Publishing, Birmingham, UK, 1st edition.

Robot (2023). Robot framework documentation. https://robotframework.org/. Acessado em 29 de Abril de 2023.

Rspec (2023). Desenvolvimento orientado a comportamento para ruby. https://rspec.info/. Acessado em 03 de Maio de 2023.

Ruby (2023). Ruby. https://ruby-doc.org/. Acessado em 03 de Maio de 2023.

Sommerville, I. (2019). *Engenharia de software*. Pearson Universidades, 10 edition.

SpecFlow (2021). Specflow documentation. https://specflow.org/. Acessado em 29 de Abril de 2023.

Sumner, A. (2014). *Everyday Rails Testing with RSpec*. Leanpub.

Wynne, M. and Hellesøy, A. (2012). *The Cucumber Book: Behaviour-driven Development for Testers and Developers*. Pragmatic Bookshelf, Raleigh, NC.