Testes automatizados de front-end com utilização de BDD e Cypress

Bruna de S. Meger[1], Kassya C. R. de Andrade[1]

[1]Faculdade de Computação e Informática (FCI)
Universidade Presbiteriana Mackenzie – São Paulo, SP – Brasil

10205187@mackenzista.com.br,
kassyachristina.andrade@mackenzie.br

Resumo. *A aplicação de testes de software é fundamental para verificar falhas e erros que possam comprometer o desenvolvimento de um projeto de software. Baseado nisso, esta pesquisa tem como escopo a compreensão do método de automação de testes BDD, conhecido como Desenvolvimento Orientado ao Comportamento, suas vantagens quando utilizado junto da ferramenta Cypress e uma demonstração a partir de um front-end desenvolvido para receber os testes automatizados. A partir da análise de livros e artigos, foi descrito como desenvolver e quais são as vantagens dos testes automatizados para um sistema front-end. Foram desenvolvidos diversos testes com a utilização da ferramenta Cucumber, um framework facilitador de escrita e configuração de testes utilizando o método BDD. Após análise da velocidade ao rodar os testes, há uma comparação com o tempo que levaria realizando-os manualmente, demonstrando as possíveis melhorias de tempo que podem beneficiar as equipes de tecnologia quando utilizam as ferramentas Cypress e Cucumber.*

Palavras-chave: *Teste de software; Desenvolvimento orientado a comportamento; Cypress; Cucumber; Equipes de tecnologia.*

Abstract. *The application of software testing is essential to check failures and errors that could compromise the development of a software project. Based on this, this research aims to understand the BDD test automation method, known as Behavior Driven Development, its advantages when used together with the Cypress tool and a demonstration of a front-end developed to receive automated tests. From the analysis of books and articles, it was described how to develop and what are the advantages of automated tests for a front-end system. Several tests were developed using the Cucumber tool, a framework that facilitates writing and configuring tests using the BDD method. After analyzing the speed when running the tests, there is a comparison with the time it would take to perform them manually, demonstrating the possible time improvements that can benefit technology teams when using the Cypress and Cucumber tools.*

Keywords: *Software testing; Behavior Driven Development; Cypress; Cucumber; Technology teams.*

1. Introdução

Ao introduzir uma motivação da construção de testes em *softwares*, a autora Graham et al. (2019) afirma que a maioria das pessoas que utilizam algum *software* no dia a dia, desde celulares até um caixa de banco, em algum momento já se deparou com algum recurso que apresentava algum erro ou que possuía *delay*. Como uma maneira de auxiliar as equipes de tecnologia que desenvolvem estes *softwares*, a automação dos testes surgiu para diminuir o número de falhas e erros no sistema, além de permitir uma entrega contínua de novas funcionalidades, ganhando tempo, desempenho e dinheiro se comparada com testes não automatizados. De acordo com o relatório *The cost of poor software quality in the US: a 2022 report* (O custo da má qualidade de *software* nos EUA: um relatório de 2022), houve uma estimativa de perda de US$ 2,41 trilhões só em 2022 devido à má qualidade de testes de *software*, comprovando que utilizar testes automatizados, que auxiliam no reconhecimento de erros, é um bom investimento. Há vários métodos de automação do processo de testes, 3 destes são conhecidos como TDD (*Test Driven Development*), BDD (*Behavior Driven Development*) e DDD (*Domain Driven Development*). O método a ser abordado neste artigo é o BDD, em que o teste é guiado pelo comportamento do negócio e o desenvolvimento no código não possui informações técnicas.

O BDD evoluiu do método TDD, portanto possui todos os benefícios do Desenvolvimento Orientado a Testes, porém com ainda mais vantagens. O autor Saleem Siddiqui (2021) explica em seu livro *Learning Test-Driven Development*, que no método TDD escreve-se primeiro o teste unitário com falha, e depois escreve-se rapidamente apenas o código suficiente para fazê-lo passar e, em seguida, tirar-se um tempo para aprimorar o código.

Segundo Jhon Ferguson (2015), o BDD foi inventado por Dan North no início dos anos 2000, como uma maneira mais fácil de ensinar e praticar o Desenvolvimento Orientado a Testes (TDD), que segundo o autor é uma técnica que utiliza testes unitários para especificar e verificar as aplicações de um código.

Uma das vantagens do BDD é a possibilidade de envolvimento de diversos profissionais além dos desenvolvedores de *software*. Isto porque há um *software* chamado Cucumber que funciona como uma ferramenta de automação de testes de *software* que utiliza o método BDD, permitindo que os casos de teste sejam escritos em mais de 70 línguas, dentre elas português e inglês. De acordo com o próprio site oficial, a ferramenta Cucumber impulsiona o desempenho de equipes de engenharia empregando o Desenvolvimento Orientado a Comportamento. Ou seja, mesmo pessoas sem experiência com desenvolvimento conseguem utilizá-lo, já que a maior parte do código pode ser feita em português, possibilitando que profissionais de qualidade e de negócio consigam contribuir na realização da automação dos testes.

O Desenvolvimento Orientado ao Comportamento (BDD) é guiado pelo comportamento do negócio, ou seja, começa-se pela descrição do comportamento a ser testado e, em seguida, a descrição da resposta esperada.

O Cypress é uma ferramenta utilizada para configurar, escrever, rodar e debugar os testes *front-end* de forma fácil. Ele tem o foco de realizar testes ponta a ponta, ou seja, testar o fluxo da aplicação desde o início até o fim, simulando uma experiência mais próxima do usuário real. A autora Narayanan Palani (2021) informa que o Cypress foi

lançado em sua primeira versão em 2015, mas que ficou extremamente popular por volta de 2018, e que a ferramenta é extremamente útil para times que utilizam estratégias focadas em prevenir erros, ao invés de se esforçarem para tentar identificá-los.

O Cypress se destaca de outras ferramentas por unificar diversas funcionalidades, bibliotecas e *frameworks* necessários para fazer o teste ponta a ponta, além de possuir uma velocidade de ações altíssima. Ficou popularmente conhecida há menos de 5 anos, portanto o trabalho contribuirá para a Academia apresentando informações e melhorias para a realização de testes automatizados, incentivando a inovação e levando novos olhares de atenção para novas ferramentas.

Essa pesquisa tem como objetivo geral descrever uma maneira de criar uma ponte entre a parte técnica e não técnica do time engenharia de *software*, utilizando o método BDD configurado pela ferramenta Cucumber juntamente ao Cypress, para possibilitar a redução do tempo de testes de equipes de tecnologia.

O objetivo específico deste trabalho engloba:

- Compreender o funcionamento do método de automação de testes, Desenvolvimento Orientado ao Comportamento;
- Compreender o funcionamento da ferramenta Cypress;
- Compreender o funcionamento da ferramenta Cucumber;
- Analisar as vantagens que as ferramentas e o método podem fornecer a uma equipe de desenvolvimento quando utilizados em conjunto;
- Demonstrar as vantagens que as ferramentas e o método podem fornecer quando utilizados em conjunto na prática, a partir de um sistema *front-end* desenvolvido para receber os testes criados, apresentando a interface do Cypress e o passo a passo feito;
- Análises e considerações finais dos tempos de execução dos casos de teste.

No ambiente de trabalho de equipes de tecnologia, há uma busca contínua por novas tecnologias que favoreçam e tragam qualidade e velocidade para o desenvolvimento de *softwares*. A autora deste artigo teve experiências com diversos *softwares* e métodos, em particular o Cypress, Cucumber e BDD apresentados nesta introdução, e, segundo sua percepção, essas ferramentas se mostraram muito eficientes quando utilizadas em conjunto. Além da facilidade de uso, estes *softwares* se mostraram ágeis na execução de testes, portanto trazem diversas vantagens notáveis de economia de tempo e, consequentemente, dinheiro. Motivada por esta experiência e reconhecimento das vantagens, decidi explorar mais a fundo o potencial destas ferramentas e apresentar os dados neste artigo, que visa apresentar de maneira prática de forma que configurar e utilizar tais ferramentas de maneira a obter-se no final diversos testes automatizados rodando em um *front-end* criado.

A justificativa da escolha de investigar as vantagens das ferramentas gratuitas Cypress e Cucumber, utilizadas em conjunto com o método BDD como tema do artigo, reflete as demandas atuais do mercado de desenvolvimento de *software*. Um artigo

publicado pela empresa *Tricentis*, em 2018, informou que ao investigarem 606 falhas de *softwares*, identificaram que 314 companhias e mais de 3.5 bilhões de pessoas foram afetadas, gerando uma perda de US$ 1.715 trilhão de dólares e de 268 anos de trabalho. Com essas informações, este trabalho de conclusão de curso se propõe a demonstrar como a aproximação dos funcionários de desenvolvimento com os funcionários de qualidade de testes, ambos ajudando na construção dos testes, pode beneficiar equipes e diminuir o gasto de tempo e dinheiro que as falhas trazem às empresas.

2. Referencial Teórico

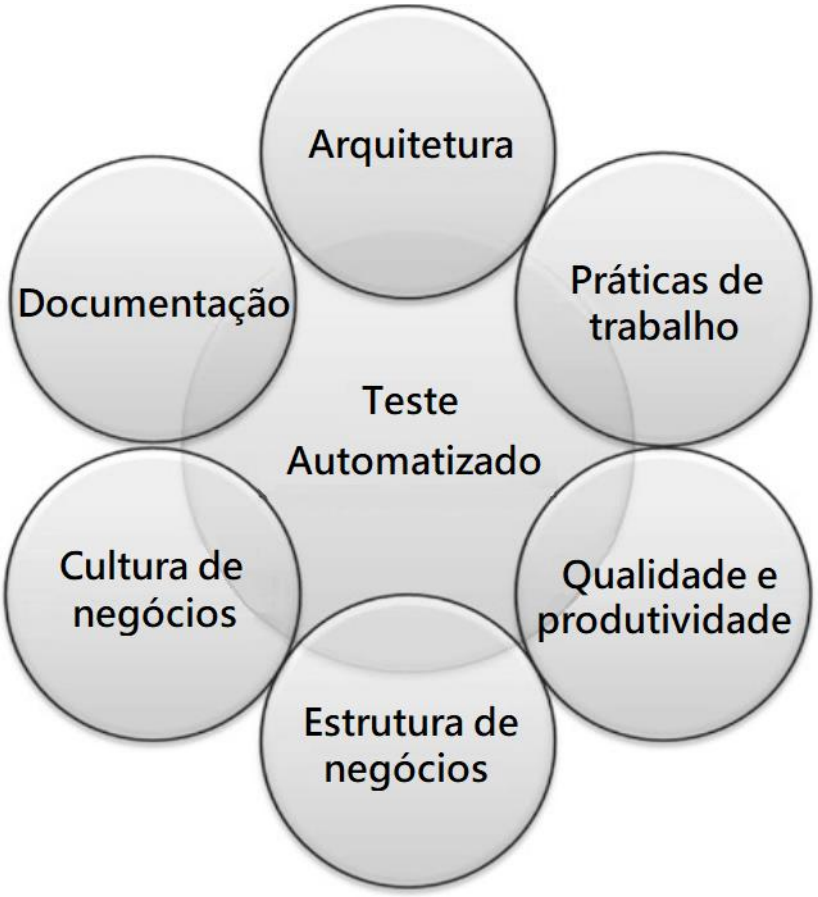
Com o rápido aumento na área de tecnologia de hardware e desenvolvimento de *software* entre os anos 1980 e 1990, os autores Glenford Myers, Corey Sandler e Tom Badgett (2012) enfatizam a necessidade de atualizações tecnológicas nos tópicos de sistemas, aplicações, segurança e tecnologias de comunicação. É esperado que nem todos os projetos de computação nos dias atuais sejam padronizados, organizados e bem-feitos, mas desde que estejam atualizados com tecnologias novas, possuem grandes potenciais. Este referencial teórico tem como foco encontrar um caminho entre discussões de autores para avaliar como mudar de um desenvolvimento de testes desorganizado para um teste de alta qualidade.

A automação de testes possui relação direta com todos os outros ciclos do desenvolvimento de um *software*. Além do que foi abordado anteriormente sobre o aumento da produtividade de um projeto, o autor Arnon Axelrod enfatiza a relação dos testes automatizados com a arquitetura do produto, estrutura organizacional, cultura etc.

> “Para mim, a automação de testes é como um espelho de todas essas coisas. Todos esses aspectos têm um efeito na automação de testes. Mas você também pode aproveitar o reflexo desses efeitos no teste automação para alterar e melhorar qualquer um desses aspectos” (AXELROD, 2018, p.10, tradução nossa).

Esta relação pode ser visualizada na figura 1, que mostra algumas das relações diretas dos testes automatizados com ciclos do desenvolvimento de *software*, muito presentes em equipes de tecnologia.

Figura 1: Relação dos testes automatizados com outros aspectos do desenvolvimento de *software*



Fonte: AXELROD (2018). P. 11. (tradução nossa).

O entrosamento e a comunicação do time são cruciais para o desenvolvimento de qualquer projeto em qualquer área, mas principalmente para a área de tecnologia, em que cada parte depende de que outra tenha sido feita com qualidade e padronização para que haja um melhor entendimento por parte do resto da equipe. As autoras Lisa Crispin e Janet Gregory (2008) afirmam que a comunicação constante entre todas as partes é essencial para o sucesso do projeto, e que todos os membros do time devem trabalhar próximos uns dos outros, independente de ser um trabalho virtual ou presencial.

A abordagem em que toda a equipe pode assumir a responsabilidade pelas tarefas de teste pode favorecer na cobertura e qualidade das mesmas, pois os olhares diferentes de cada um, junto da comunicação entre os membros da equipe, podem levar a uma proveitosa reunião de ideias novas e diferentes. É justamente pensando nas vantagens desta aproximação entre a equipe técnica e não técnica, que as metodologias de desenvolvimento de teste TDD e BDD são utilizadas para realizar a escrita de testes automatizados. O Autor Saleem Siddiqui explica em seu livro *Learning Test-Driven Development* (Aprendendo Desenvolvimento Orientado a Testes) que

> "O código deve inspirar confiança, especialmente o código de nossa autoria. Essa confiança, embora seja um sentimento nebuloso, é baseada em uma expectativa de previsibilidade. [...] É da natureza humana valorizarmos a regularidade e a previsibilidade ainda mais do que o patrimônio líquido. [...] O desenvolvimento orientado a testes aumenta nossa confiança em nosso código

> porque cada novo teste flexibiliza o sistema de maneiras novas e não testadas - literalmente! Com o tempo, o suíte de testes que criamos nos protege contra falhas de regressão. Essa bateria de testes cada vez maior é a razão pela qual, à medida que o tamanho do código aumenta, o mesmo acontece com a sua qualidade e a nossa confiança nele" (SIDDIQUI, 2021, p. 16, 17, tradução nossa).

Sobre as especificações técnicas das metodologias abordadas, o autor Enrique Amoedo enfatiza que

> "A prática de ter um conjunto de testes automatizados não é exclusiva do TDD e do BDD, mas é bastante antiga. O que realmente diferencia abordagens como TDD e BDD é o fato de serem abordagens de *test-first*. Nos testes tradicionais, você escreve seu teste automatizado depois que o código é escrito. [...] No *test-first*, usamos testes com falha como um guia para saber se há necessidade de novo código ou não" (AMOEDO, 2015, p. 7, 8, 10, tradução nossa).

A abordagem de testes *test-first* tem diversas vantagens e algumas são citadas no livro de Enrique Amoedo (2015), por exemplo, explicando que os erros são encontrados e solucionados mais rapidamente pois o ciclo de escrita de testes requer que o código fonte seja escrito apenas com o necessário para implementar tal funcionalidade, portanto caso o teste apresente falhas já neste estágio, fica mais fácil e rápida a correção em comparação a uma correção caso o código fonte estivesse completamente feito. Há também a vantagem de a base de código poder ser aprimorada usando-se mecanismos de refatoração, pois sem testes é muito difícil fazer isso, já que não se pode saber se a mudança de código feita por você mudou a funcionalidade do sistema.

Discute-se então, o porquê de se utilizar o método BDD ao invés do TDD. O autor Enrique Amoedo explica em um trecho de seu livro

> "O maior problema no TDD clássico é que existe uma desconexão entre o que o produto deve fazer e o que o conjunto de testes que a equipe de desenvolvimento constrói está testando. O TDD não diz explicitamente como conectar os dois mundos. Isso leva muitas equipes a utilizar o TDD, mas testando as coisas erradas" (AMOEDO, 2015, p. 14, tradução nossa).

Também sobre a motivação da utilização do método de Desenvolvimento Orientado a Comportamento, a perspectiva do autor Jhon Ferguson é a seguinte

> "Trata-se de construir um *software* que funcione bem e seja fácil de alterar e manter, mas, mais importante, é sobre a construção de *software* que fornece valor real para seus usuários. Queremos construir *software* bom, mas também precisamos construir *software* que valha a pena ser construído" (FERGUSON, 2015, p.3, tradução nossa).

Ainda sobre Jhon Ferguson, o autor enfatiza a importância do levantamento dos cenários dos testes, que agem como a base dos critérios de aceitação

> "Uma parte fundamental dessa prática envolve a definição de cenários, ou exemplos concretos de como um determinado recurso ou história funciona. Esses cenários ajudarão você a validar e ampliar sua compreensão do problema, e também são uma excelente ferramenta de comunicação. Eles agem como a base dos critérios de aceitação, que você integra ao processo de construção na forma de testes de aceitação automatizados" (FERGUSON, 2015, p.33, tradução nossa).

Discutindo sobre as ferramentas que foram utilizadas, os autores Matt Wynne e Aslak Hellesoy (2012) abordam as motivações para se utilizar o Cucumber.

> "O Cucumber ajuda a facilitar a descoberta e o uso de uma linguagem onipresente dentro da equipe, dando aos dois lados da divisão linguística um lugar onde eles podem se encontrar. Os testes do Cucumber interagem diretamente com o código dos desenvolvedores, mas eles são escritos em um meio e linguagem que as pessoas administrativas da empresa possam entender. Ao trabalharem juntos para escrever esses testes – especificando de forma colaborativa – os membros da equipe não apenas decidem qual comportamento eles precisam implementar em seguida, mas também aprendem como descrever esse comportamento em linguagem que todos entendam" (WYNNE, HELLESOY, 2012, p. 5, tradução nossa).

Compreende-se então que utilizando tal ferramenta, conseguimos aproximar a parte técnica da não técnica da equipe, fazendo com que trabalhem juntos nas escritas e desenvolvimento dos casos de teste. Os mesmos autores Matt Wynne e Aslak Hellesoy (2012) explicam o funcionamento da ferramenta Cucumber

> "Cucumber é uma ferramenta de linha de comando. Quando você o executa, ele lê suas especificações em arquivos de texto em linguagem simples chamados *features*, examina-os em busca de cenários para testar e os executa em seu sistema. Cada cenário é uma lista de etapas para o Cucumber trabalhar em cima. Para que o Cucumber possa entender esses arquivos de *features*, eles devem seguir algumas regras básicas de sintaxe. O nome para este conjunto de regras é Gherkin" (WYNNE, HELLESOY, 2012, p. 5, tradução nossa).

Gherkin é uma linguagem que deve ser escrita em um conjunto de regras específicas, e é uma parte fundamental da metodologia de Desenvolvimento Orientado a Comportamento. Os mesmos autores da citação acima explicam que, respeitando a estrutura de palavras-chave *Feature, Scenario, Given, When, and Then*, pode-se utilizar 70 línguas para escrever um cenário. Em português, as palavras utilizadas devem ser Funcionalidade, Cenário, Dado, Quando e Então. Estas palavras-chave, juntamente das palavras das tantas outras línguas aceitas, podem ser verificadas no site oficial da ferramenta Cucumber.

A ferramenta Cypress foi utilizada neste projeto para apresentar uma interface gráfica de visualização para os testes automatizados. O autor Waweru Mwuara consegue resumir o que exatamente é a ferramenta conforme abaixo

> "Cypress é um *framework* de automação de testes ponta a ponta construído e projetado para aplicações *web*. Ele se concentra em eliminar inconsistências nos testes, garantindo que você possa escrever, depurar e executar testes no navegador sem precisar de configuração adicional ou pacotes adicionais. Cypress funciona como um aplicativo independente e pode ser instalado em Sistemas operacionais macOS, Unix/Linux e Windows usando aplicativos Hyphenate ou ferramentas de linha de comando" (MWUARA, 2021, p. 3, tradução nossa).

As pesquisas teóricas realizadas foram de suma importância para a definição da metodologia e parte prática deste artigo, pois a partir delas foi possível compreender o funcionamento do método de testes Desenvolvimento Orientado a Comportamento, e a forma de aplicá-lo mediante a utilização da ferramenta Cucumber. Também foi importante compreender o funcionamento da ferramenta Cypress para que fosse possível a visualização e a configuração dos testes automatizados.

3. Metodologia

Ao tratar da metodologia de pesquisa deste artigo, as etapas executadas foram:

1. O Levantamento bibliográfico foi feito a partir de livros de autores renomados, como Jhon Ferguson e Enrique Amoedo, que descreveram o passo a passo da escrita de testes BDD; Matt Wynne e Aslak Hellesoy, que exploraram todo o funcionamento do Cucumber e Waweru Mwaura descrevendo a forma de utilizar o Cypress para realização de testes ponta a ponta;

2. Pesquisas refinadas sobre as ferramentas Cypress e Cucumber para realizar as configurações e desenvolvimentos necessários;

3. Análise e demonstração dos resultados das pesquisas;

4. Desenvolvimento de um sistema *front-end*;

5. Desenvolvimento dos testes do sistema *front-end*, criado utilizando-se a linguagem Gherkin suportada pela ferramenta Cucumber;

6. Interligação dos casos do teste escritos em Gherkin com a configuração dos mesmos testes escritos em *JavaScript*;

7. Documentação de toda configuração dos testes criados, descrevendo o passo a passo da utilização das ferramentas;

8. Comparação dos tempos de execução dos testes manuais e automatizados;

9. Finalização e últimos ajustes.

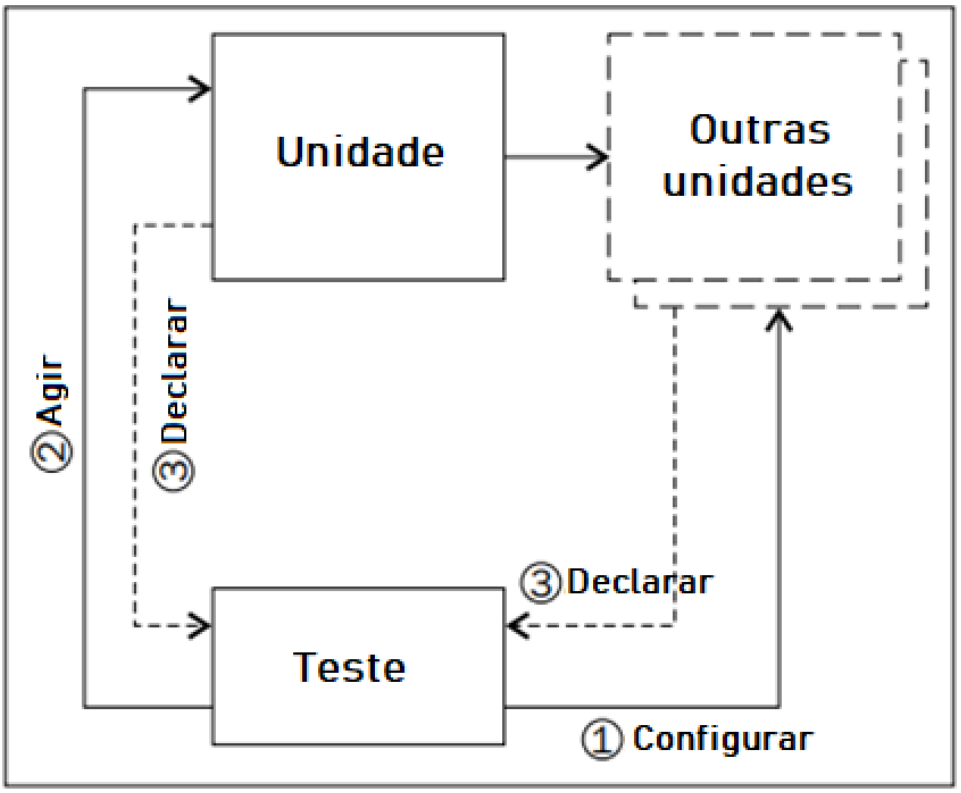
O principal propósito do projeto foi descrever o funcionamento do método de automação BDD, suportado pela ferramenta Cucumber, juntamente da utilização da ferramenta Cypress para configuração e *feedback* rápido dos testes automatizados, determinando quais funcionalidades e vantagens que eles trazem para a equipe de tecnologia que os utiliza em conjunto.

Os conhecimentos teóricos foram adquiridos em livros, como o do autor William Perry (2006) que trata sobre métodos efetivos de testes, e em artigos científicos citados anteriormente. Para a implementação prática, foi construído um sistema *front-end* no formato padrão MVC (Model-View-Controller) com utilização do *framework* ASP.NET MVC, que é ligado a um banco de dados SQL Server Management Studio. Para a realização dos testes, foram utilizadas em conjunto as ferramentas Cucumber para realizar a escrita dos testes em português, e Cypress que orienta e facilita a configuração e execução dos testes para receber o teste automatizado ponta a ponta, anotando e documentando todos os comportamentos dos testes neste artigo.

Foi descrito também todo o passo a passo detalhado do Desenvolvimento Orientado ao comportamento (BDD), escrito com a ferramenta Cucumber para compreensão da metodologia.

Na escrita dos testes automatizados, foi utilizado o tipo de estrutura de testes apresentado pelo autor Enrique Amoedo (2015). O autor explica que esse diagrama apresentado na figura 2 representa a estrutura de um teste unitário, em que o teste realiza uma única operação na unidade por meio de sua *API*. A função do teste é verificar o resultado de determinada funcionalidade, e identificar se o valor de retorno do teste e seus efeitos colaterais são os esperados. Os efeitos colaterais que serão vistos neste artigo serão chamadas e gravações ao banco de dados.

Figura 2: Estrutura de teste unitário



Fonte: AMOEDO (2015) P. 21. (tradução nossa).

Portanto, os passos 1, 2 e 3, apresentados na figura 2, foram seguidos da seguinte maneira:

1. Configurar: Nesta fase, foram configurados os estados e parâmetros de entrada, além de verificado se o banco de dados retorna uma resposta conhecida;

2. Agir: Nesta fase, foram realizadas as ações esperadas de cada teste;

3. Declarar: E nesta fase, verifica-se o valor retornado do teste e seus efeitos colaterais;

Após seguir os passos acima para desenvolver cada teste pensado, os testes foram transformados para o formato da linguagem Gherkin, e assim, colocados no código para que a ferramenta Cucumber pudesse rodar o passo a passo de cada teste e mostrá-lo na ferramenta Cypress.

4. Resultado e Discussão

O desenvolvimento inicial deste projeto foi centrado em compreender o funcionamento da produção dos testes orientados a comportamento. Após entender suas vantagens e métodos de escrita de casos de teste, o esforço seguinte foi alocado na integração do método BDD escrito com a linguagem de programação Gherkin, integrado à ferramenta Cucumber, juntamente da ferramenta de configuração e escrita de testes, Cypress.

A ferramenta inicial utilizada foi o Node.js, um ambiente de execução de JavaScript *back-end*, utilizado para criação de *APIs*, servidores *web*, sistemas de monitoramento, dentre outras opções. O Node utiliza como ferramenta gerenciadora de bibliotecas e pacotes para a linguagem JavaScript o Node Package Manager (NPM), amplamente utilizado em projetos *front-end* para acelerar o desenvolvimento, minimizando a necessidade do desenvolvedor de instalar manualmente bibliotecas e suas dependências. Ao instalar o Node.js, passamos a ter diversos comandos disponíveis ao nosso alcance que facilitam o processo de instalação de dependências e configurações. Neste projeto, foi utilizado alguns comandos para inicializar um novo projeto já com configurações básicas, como por exemplo o arquivo 'package.json' que é criado automaticamente e armazena configurações e propriedades necessárias para o projeto funcionar, comandos para instalar o Cypress automaticamente e comandos para instalar um pacote que permite ao Cypress utilizar a abordagem de testes com Cucumber, um *framework* de testes que permite escrever cenários de teste em 70 tipos de línguas diferentes, como inglês, português etc. utilizando-se a linguagem de programação Gherkin.

Após utilizar estes comandos, foi necessário fazer algumas pequenas alterações nos arquivos criados automaticamente pelos comandos. Como visto nas figuras 3 e 4, estas são as configurações dos dois arquivos importantes.

Figura 3: Captura de tela do arquivo '*cypress.config.js*'

cypress.config.js > ...

```
1   const cucumber = require('cypress-cucumber-preprocessor').default
2   const { defineConfig } = require("cypress");
3
4   module.exports = defineConfig({
5     e2e: {
6       setupNodeEvents(on, config) {
7         on('file:preprocessor', cucumber())
8       },
9       specPattern: "cypress/e2e/*.feature",
10    },
11  });
12
```

Fonte: autoria própria.

Figura 4: Captura de tela do arquivo 'package.json'

```
{} package.json > ...
{
  "name": "bdd-cucumberv2",
  "version": "2.0.0",
  "description": "Cucumber Framework",
  "main": "index.js",
  ▷ Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "cypress"
  ],
  "author": "Bruna Meger",
  "license": "ISC",
  "devDependencies": {
    "cypress": "^13.8.1",
    "cypress-cucumber-preprocessor": "^4.3.1"
  },
  "cypress-cucumber-preprocessor": {
    "nonGlobalStepDefinitions": true,
    "step_definitions": "cypress/e2e/page"
  },
  "run-features": "cucumber-js ./features -r ./steps -f node_modules/cucumber-pretty"
}
```

Fonte: autoria própria.

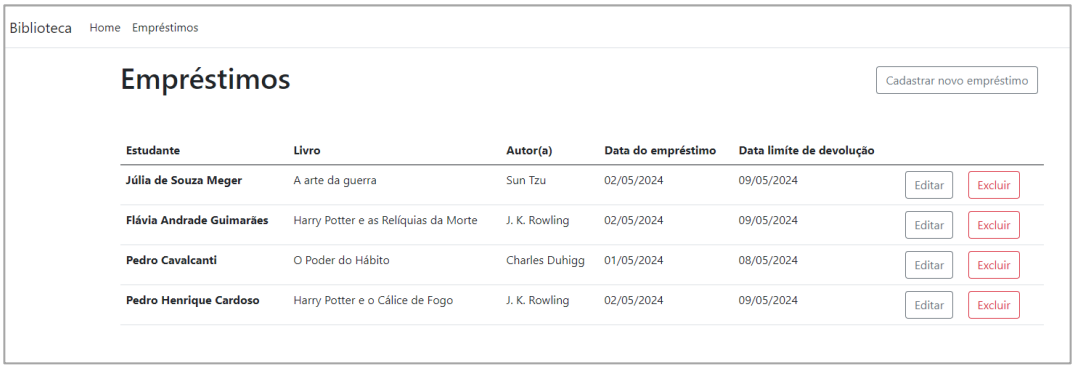
Com as configurações feitas corretamente, o próximo passo foi escrever os casos de teste, realizar a programação em Gherkin e visualizar o funcionamento dos testes na ferramenta Cypress.

Os casos de teste foram criados para uma aplicação *web* desenvolvida por autoria própria. O *front-end* foi desenvolvido no formato padrão MVC (Model-View-Controller) com utilização do *framework* ASP.NET MVC na *IDE* Visual Studio 2022, utilizando as linguagens *C#*, *Html*, *Css* e *JavaScript*. Com o auxílio do *framework* de *front-end bootstrap*, foi criado um layout e interface temático de biblioteca, possuindo uma página principal e uma página de registro de empréstimos de livros. A página de Empréstimo de livros possui uma tabela funcional de registro de dados de aluguel de livros que foi interligada ao banco de dados desenvolvido no SQL Server Management Studio.

Figura 5: Captura de tela da página inicial do *front-end*

Fonte: autoria própria.

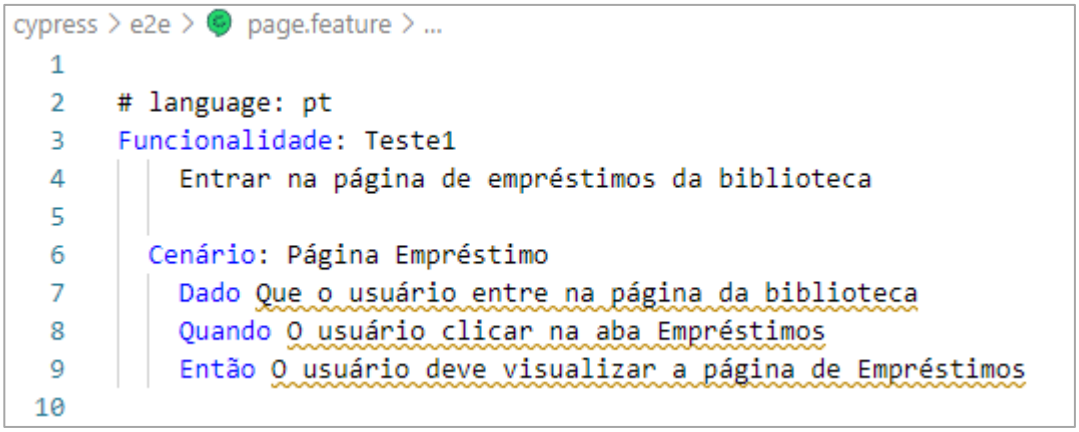
Figura 6: Captura de tela da página de Empréstimos do *front-end*



Fonte: autoria própria.

Ao escrever um caso de teste, é necessário somente adaptá-lo ao formato da linguagem Gherkin, que trabalha em paralelo com a ferramenta Cucumber. O Gherkin utiliza palavras-chave para estruturar as condições, e estas palavras-chave podem ser escritas em 70 línguas simples, que podem ser conferidas em sua documentação oficial. Para o português, as palavras principais utilizadas são ‘Funcionalidade, Cenário, Dado, Quando, Então’. A palavra ‘Funcionalidade’ deve ser utilizada para identificar a funcionalidade testada, podendo ter uma pequena descrição abaixo; a palavra ‘Cenário’ deve apresentar uma descrição do caso de teste; ‘Dado’ deve apresentar a condição inicial do teste; ‘Quando’ deve ser seguido das ações que ocorrem no caso de teste; ‘Então’ deve apresentar o resultado esperado do teste.

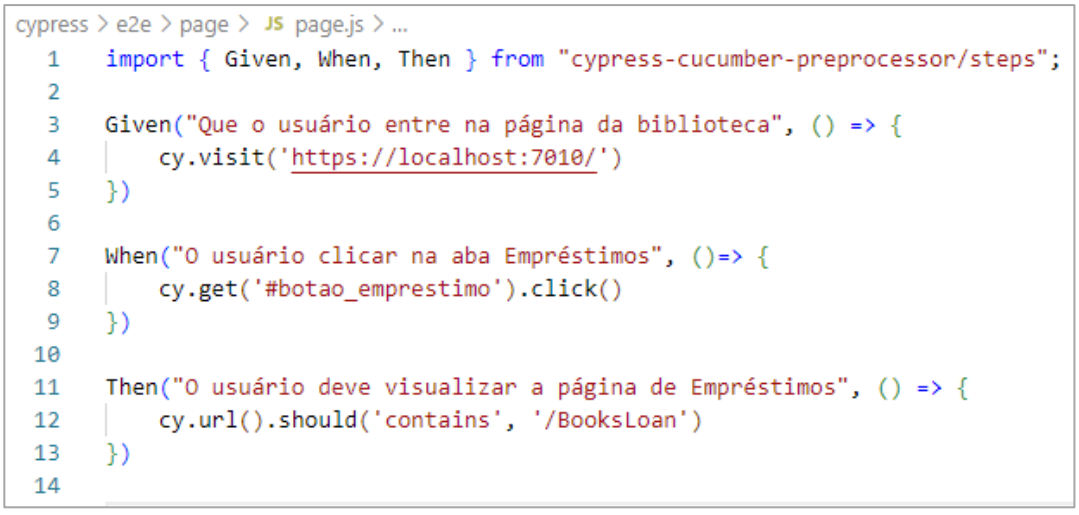
Figura 7: Captura de tela do teste escrito em linguagem Gherkin



Fonte: autoria própria.

Na figura 7, conseguimos visualizar um exemplo de teste escrito para a página de Biblioteca, em que o intuito do teste é verificar se o botão da página Empréstimos realmente nos leva à aba Empréstimos. Podemos verificar que na primeira linha do código, há uma definição da linguagem utilizada, que no caso deste artigo foi o português; esta linha de código é necessária para o Gherkin identificar as palavras-chave na língua escolhida.

Figura 8: Captura de tela do teste escrito em linguagem JavaScript

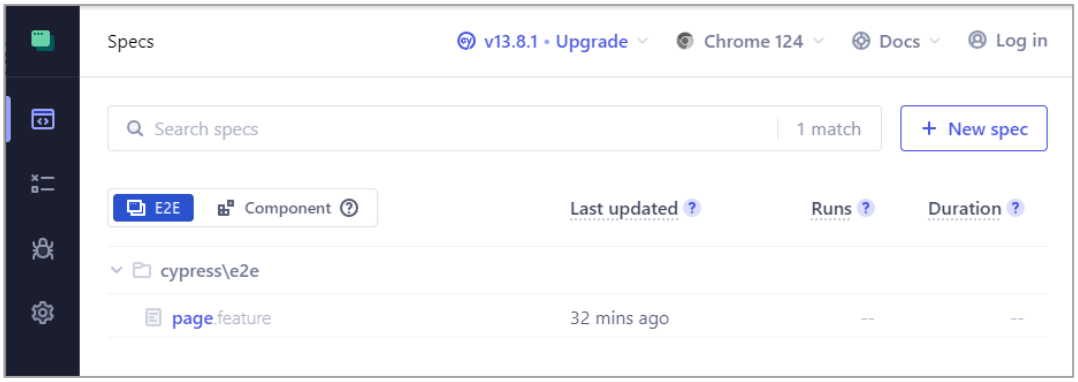


Fonte: autoria própria.

Após escrever os testes em Português, precisamos descrever em *JavaScript* o passo a passo exato do que deve ser feito pelo Cypress, conforme podemos ver na figura 8. Na linha 4, é descrito como o Cypress deve acessar nosso *front-end*. Já na linha 8 é informado onde deve haver um clique, e na linha 12 é informado como o Cypress deve identificar que a página de Empréstimos foi aberta.

Por fim, após escrever o comando *'npx Cypress open'*, é possível visualizar em um browser de sua escolha todos os conjuntos de cenários que foram escritos, podendo-se executá-los facilmente conforme mostra a figura 9.

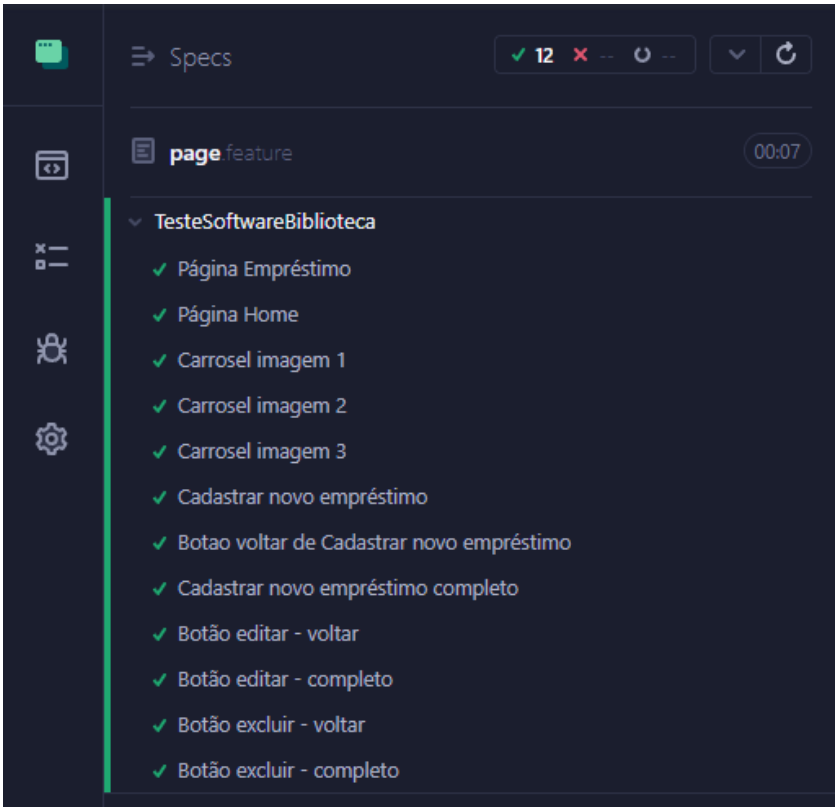
Figura 9: Captura de tela da página inicial do Cypress



Fonte: autoria própria.

Com apenas um clique no conjunto de teste *'page'* é possível visualizar o Cypress executando automaticamente todos os cenários de teste escritos para o conjunto funcionalidade, conforme pode-se conferir na figura 10.

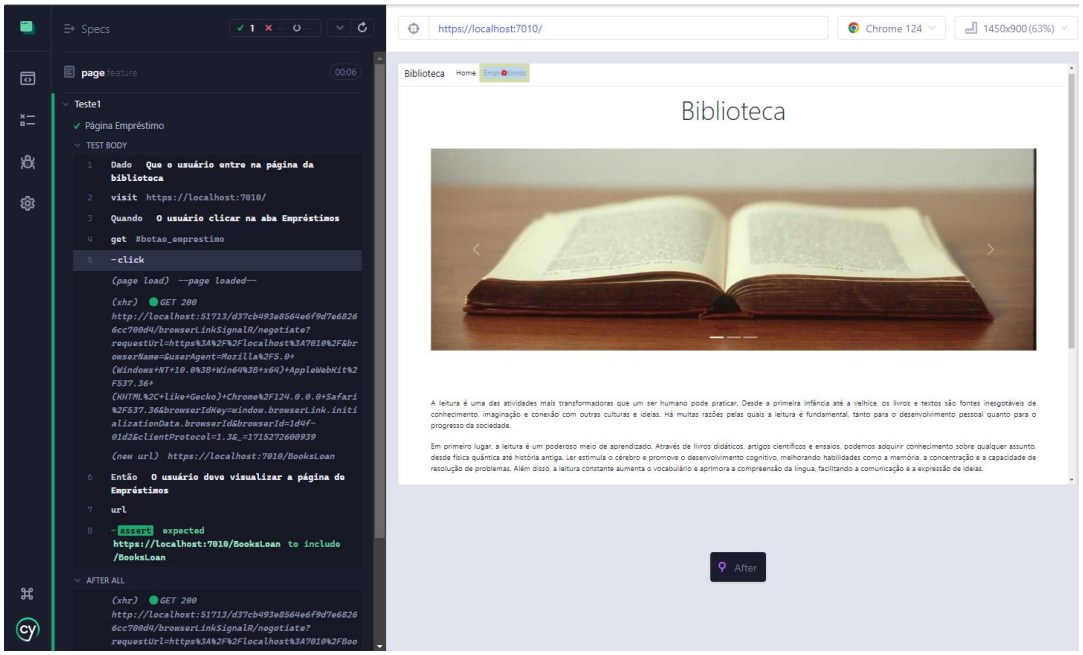
Figura 10: captura de tela do conjunto de teste executados pelo Cypress



Fonte: autoria própria.

Ao clicar em algum dos cenários de teste, como por exemplo 'Página Empréstimo' é possível ver um retorno 200, ou seja, o teste foi positivo e encontrou a página de Empréstimos após o clique como é possível visualizar na figura 11.

Figura 11: Captura de tela do teste executado no Cypress



Fonte: autoria própria.

Após a realização de todo o passo a passo citado acima, foi realizada uma comparação do tempo necessário para realizar todos os testes criados, tanto manualmente quanto automatizado com as ferramentas citadas neste artigo. Os testes manuais foram feitos pela autora do artigo, que já conhecia o sistema, portanto é possível que o tempo registrado fosse ainda maior caso os testes fossem realizados por outra pessoa. A comparação é feita na tabela 1.

Tabela 1: Tempos de execução de cada cenário de teste (em segundos)

Cenários de Teste	Testes Manuais	Teste Automatizados
Página Empréstimo	3 s	0.3 s
Página Home	4 s	0.4 s
Carrossel imagem 1	1 s	0.1 s
Carrossel imagem 2	2 s	0.2 s
Carrossel imagem 3	3 s	0.2 s
Cadastrar novo empréstimo	4 s	0.3 s
Botão voltar de novo empréstimo	5 s	0.4 s
Cadastrar novo empréstimo completo	10 s	0.9 s
Botão editar - voltar	5 s	0.8 s
Botão editar - completo	7 s	1 s
Botão excluir - voltar	5 s	0.8 s
Botão excluir - completo	5 s	1 s
Tempo total	**54 s**	**7 s**

Fonte: autoria própria.

Observa-se que o tempo de execução é muito menor ao realizar os testes automatizados, além de que, quanto mais extenso for o *software*, maior o tempo economizado pelas equipes de tecnologia.

É interessante ressaltar que este conjunto de ferramentas é importante para os testes de regressão, que é um tipo de teste de *software* utilizado para verificar se as últimas alterações feitas no código não geraram novos erros e bugs em partes que já funcionavam corretamente anteriormente. Ou seja, os testes não precisam ser reescritos sempre que o código é mudado, garantindo que mesmo após modificações no código, o sistema continue a funcionar igualmente.

Compreende-se então a facilidade que utilizar o Cypress em conjunto com o Cucumber pode trazer a equipes de tecnologia, principalmente por permitir que pessoas da equipe que não programam possam participar diretamente no desenvolvimento dos casos de teste, escrevendo-os no formato adequado da linguagem Gherkin em português e delegando a parte de configuração de *JavaScript* aos desenvolvedores, de modo a agilizar o trabalho e a melhorar a integração entre os membros da equipe.

5. Conclusão

A utilização das ferramentas Cypress e Cucumber combinadas com a metodologia Desenvolvimento Orientado a Comportamento (BDD) provou ser uma abordagem eficiente e vantajosa para testar a automação de sistemas *front-end*. Os estudos realizados e as implementações práticas mostram que a automação não só reduz significativamente o tempo necessário para a realização de testes, mas também aumenta a precisão e a confiabilidade do processo de desenvolvimento de *software*.

A integração dessas ferramentas permite que os membros da equipe, mesmo aqueles sem conhecimento avançado de programação, participem ativamente na criação de testes, escrevendo-os na linguagem Gherkin. Isso facilita a comunicação e a colaboração dentro da equipe, promovendo um ambiente de trabalho mais inclusivo e produtivo.

Os testes automatizados desenvolvidos economizam muito tempo em comparação aos testes manuais. Esse ganho de tempo é benéfico para grandes projetos de *software*, onde a velocidade e a precisão são essenciais para a entrega de um produto de alta qualidade. Além disso, a utilização destas ferramentas ajuda a detectar erros precocemente, reduzindo assim os custos associados a falhas descobertas durante fases de desenvolvimento mais avançadas.

Este trabalho destaca a importância da automação de testes nos cenários atuais de desenvolvimento de *software*, ressaltando como a utilização de Cypress e Cucumber com o método BDD pode otimizar processos, economizar recursos e melhorar a qualidade final dos produtos desenvolvidos. Espera-se que os resultados apresentados neste artigo contribuam para melhorias nas equipes de engenharia de *software*, promovendo inovação, velocidade e qualidade para diversos projetos.

As propostas de continuidade do estudo abrangem a criação de teste automatizados para *softwares* bem mais extensos do que o apresentado neste artigo, e a criação de um treinamento educacional detalhado para capacitar desenvolvedores na utilização das ferramentas Cypress e Cucumber.

6. Referências bibliográficas

AMODEO, E. Learning Behavior-driven Development with JavaScript. [s.l.] Packt Publishing Ltd, 2015.

AXELROD, A. Complete Guide to Test Automation: Techniques, Practices, and Patterns for Building and Maintaining Effective *Software* Projects. Apress, 2018.

CRISPIN, L.; GREGORY, J. Agile testing: A practical guide for testers and agile teams. Addison-Wesley, 2009.

CUCUMBER | Tools & techniques that elevate teams to greatness. Disponível em: <https://cucumber.io>. Acesso em: 19 maio. 2024.

CYPRESS | JavaScript End to End Testing *Framework*. Disponível em: <https://www.cypress.io>. Acesso em: 19 maio. 2024.

FERGUSON, J. BDD in Action: Behavior-Driven Development for the whole *software* lifecycle. Manning Publications Co, 2015.

GRAHAM, D.; VEENENDAAL, E.; BLACK, R. Foundations of *Software* Testing: ISTQB Certification. Cengage Learning EMEA, 2019.

KRASNER, H. THE COST OF POOR *SOFTWARE* QUALITY IN THE US: A 2022 REPORT. [s.l.] CISQ, 15 dez. 2022. Disponível em: <https://www.it-cisq.org/the-cost-of-poor-quality-*software*-in-the-us-a-2022-report/>. Acesso em: 19 maio. 2024.

MWAURA, W. End-to-End Web Testing with Cypress. [s.l.] Packt Publishing Ltd, 2021.

MYERS, G.; SANDLER, C.; BADGETT, T. The Art of *Software* Testing. John Wiley & Sons, Inc, 2012.

PALANI, N. Automated *Software* testing with Cypress. CRC Press, 2021.

PERRY, W. Effective Methods for *Software* Testing. Wiley Publishing, Inc, 2006.

SIDDIQUI, S. Learning Test-Driven Development. [s.l.] “O’Reilly Media, Inc.”, 2021.

TRICENTIS. Tricentis *Software* Fail Watch Finds 3.6 Billion People Affected and $1.7 Trillion Revenue Lost by *Software* Failures Last Year. Disponível em: <https://www.globenewswire.com/news-release/2018/01/24/1304535/0/en/Tricentis-*Software*-Fail-Watch-Finds-3-6-Billion-People-Affected-and-1-7-Trillion-Revenue-Lost-by-*Software*-Failures-Last-Year.html>. Acesso em: 19 maio. 2024.

WYNNE, M.; ASLAK HELLESOY; TOOKE, S. The Cucumber Book. [s.l.] Pragmatic Bookshelf, 2017.