# Automação de testes:
## Uma proposta para o projeto SisEnex

Larissa Pereira de Oliveira

CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

João Pessoa, 2021

Larissa Pereira de Oliveira

# Automação de testes

Monografia apresentada ao curso Ciência da Computação
do Centro de Informática, da Universidade Federal da Paraíba,
como requisito para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Danielle Rousy Dias Ricarte

Dezembro de 2021

**Catalogação na publicação**
**Seção de Catalogação e Classificação**

O48a Oliveira, Larissa Pereira de.
Automação de testes: uma proposta para o projeto SisEnex / Larissa Pereira de Oliveira. - João Pessoa, 2021.
47 f. : il.

Orientação: Danielle Rousy Dias Ricarte.
TCC (Graduação) - UFPB/CI.

1. Software. 2. Automação de testes. 3. Qualidade de software. 4. Ciência da computação. I. Ricarte, Danielle Rousy Dias. II. Título.

UFPB/CI CDU 004.4

Elaborado por ROGERIO FERREIRA MARQUES - CRB-15/690

**UNIVERSIDADE FEDERAL DA PARAÍBA**
**CENTRO DE INFORMÁTICA**
**BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO**
**COORDENAÇÃO DO CURSO**

Centro de Informática
UFPB

Ata da Sessão Pública de Defesa de Trabalho de Conclusão de Curso de **Larissa Pereira de Oliveira**, realizada em **14** de **dezembro** de **2021**.

Aos **quatorze** dias do mês de **dezembro**, do ano de **dois mil e vinte e um**, às **quinze** horas, por meio de videoconferência, reuniram-se os membros da Banca Examinadora constituída para julgar o Trabalho de Conclusão de Curso da Sra. **Larissa Pereira de Oliveira**, matrícula nº**20160163156**, aluna do Curso de Bacharelado em Ciência da Computação da Universidade Federal da Paraíba. A comissão examinadora foi composta pela professora **Danielle Rousy Dias Ricarte** (UFPB), orientadora e presidente da banca, e pelos professores **Eudisley Gomes dos Anjos** (UFPB), examinador interno, e **Adriana Carla Damasceno** (UFPB), examinador interno. Iniciando os trabalhos, a presidente da banca cumprimentou os presentes, comunicou-os da finalidade da reunião e passou a palavra à candidata para que fizesse a exposição oral da monografia intitulada "Automação de testes: Uma proposta para o projeto SisEnex". Concluída a exposição, a candidata foi arguida pela Banca Examinadora que, em seguida, emitiu o seguinte parecer: "***aprovada***", com conceito 9,0 (0,0 a 10,0). Do ocorrido, eu, Leandro Carlos de Souza, Coordenador do Curso de Bacharelado em Ciência da Computação, lavrei a presente ata que vai assinada por mim e pelos membros da banca examinadora. João Pessoa, 14 de dezembro de 2021.

Prof. Leandro Carlos de Souza

Profa.
Danielle Rousy Dias Ricarte (UFPB)

Prof.
Eudisley Gomes dos Anjos (UFPB)

Profa.
Adriana Carla Damasceno (UFPB)

*"Por vezes sentimos que aquilo que fazemos é apenas uma gota no oceano. Mas, o oceano seria menor se lhe faltasse uma gota."(Madre Teresa de Calcutá)*

# AGRADECIMENTOS

Em primeiro lugar, a Deus, por ter me ajudado a ultrapassar todas dificuldades encontradas durante o curso, não me deixando desistir em meio a tantas situações adversas.

Aos meus pais que sempre me apoiaram em todos os momentos, me abastecendo de ânimo e coragem para seguir em frente em busca do meu objetivo.

A professora Danielle Rousy, por ter sido minha orientadora e minha inspiração, agradeço todos os ensinamentos, pois foram essenciais para a conclusão deste trabalho.

Agradeço aos meus colegas e a todos àqueles que me ajudaram tanto diretamente quanto indiretamente, sei que sozinha nada disso seria possível.

# RESUMO

A utilização de softwares faz parte da rotina de milhares de pessoas, proporcionando uma maior qualidade de vida e maior produtividade. Durante o processo de desenvolvimento de software é muito provável que centenas de defeitos sejam encontrados. Percebe-se que a medida que o sistema evolui, o esforço demandado para realizar os testes se torna exorbitante, sem o auxílio de ferramentas de automatização. Sendo assim, o objetivo deste trabalho foi apresentar uma proposta de implementação e execução da automação dos testes, considerando o escopo do projeto SisEnex. Os resultados obtidos demonstraram ganho de tempo e uma redução no esforço no desenvolvimento da plataforma SisEnex adotada durante o ENEX de 2020.

**Palavras-chave:** Automação de Testes. Qualidade de Software. Teste de Software.

# ABSTRACT

The use of software is part of the routine of thousands of people, providing a better quality of life and greater productivity. During the software development process it is very likely that hundreds of defects will be found. It is noticed that as the system evolves, the effort required to perform the tests becomes exorbitant, without the aid of automation tools. Therefore, the objective of this work was to present a proposal for the implementation and execution of test automation, considering the scope of the SisEnex project. The results obtained demonstrated a saving in time and a reduction in the effort in the development of the SisEnex platform adopted during the 2020 ENEX.



**Key-words:** Test Automation. Software Quality. Software Testing.

# LISTA DE FIGURAS

# LISTA DE TABELAS

# LISTA DE ABREVIATURAS

BDD – Behavior Driven Development

CI – Centro de Informática

ENEX – Encontro de Extensão

LUMO – Laboratório de computação Móvel e Ubíqua

SISENEX – Sistema de Avaliação do Enex

UFPB – Universidade Federal da Paraíba

## Sumário

# 1 INTRODUÇÃO

No mundo atual, a utilização de softwares faz parte da rotina de centenas de milhares de pessoas, proporcionando uma maior qualidade de vida e maior produtividade. Dessa forma, a demanda pela qualidade do software só aumenta a cada década, sendo, atualmente, um conceito indispensável quando se fala em proporcionar uma boa experiência para o usuário.

Para que a qualidade do software seja garantida, é necessário assegurar a qualidade diante do processo de desenvolvimento da aplicação. Além disso, a realização dos testes se torna essencial para alcançar esse fim.

Bartié (2002, p. 3) afirma que no princípio do desenvolvimento de software, a prática da execução de teste era vista como uma simples tarefa de navegação e retificação no código com problema. Essas tarefas eram feitas pelos próprios desenvolvedores, o que, muitas vezes, comprometia a qualidade do software.

Segundo Sommerville (2011, p. 147), o processo de teste envolve testes manuais e automatizados. No teste manual, o testador utiliza-se de alguns dados, executa o programa e ao fim compara os resultados verificando se há conformidade com o que é esperado; ele anota e reporta as divergências aos desenvolvedores do programa. Em testes automatizados, os testes são codificados e cada vez que o sistema em desenvolvimento é testado, os scripts de teste são executados.

Durante o processo de desenvolvimento de software é muito provável que dezenas, centenas ou até milhares de defeitos sejam encontrados. Percebe-se que a medida que o sistema evolui, o esforço demandado para realizar os testes se torna exorbitante, sem o auxílio de ferramentas de automatização.

## 1.1 Definição do Problema

O Laboratório de Computação Ubíqua e Móvel (LUMO) da Universidade Federal da Paraíba desenvolve um produto denominado SisEnex, proveniente da ação de extensão estudantil, na qual presta apoio à realização do Encontro de Extensão (ENEX), que ocorre todos os anos. O sistema está disponível tanto na plataforma web quanto móvel, contando com 3 perfis de usuários: monitores, avaliadores e gerentes.

O principal objetivo do SisEnex é proporcionar uma maior agilidade no que diz respeito à avaliação e organização do evento. Até o ano de 2019 o SisEnex contava apenas com testes manuais, na plataforma web e móvel, o que demandava grandes esforços na realização dos testes a cada liberação de versão.

Embora saiba-se que nem todos os testes podem ser automatizados e parametri-

zados, é importante ressaltar os diversos benefícios que cercam a automação de testes, sobretudo, diante de tarefas repetitivas, onde a porcentagem de falha humana é maior.

Tendo em vista a importância da automação de testes no processo de garantia da qualidade de software, esse trabalho aborda a implantação e execução dos testes automatizados, analisando os desafios e limitações diante ao processo de automação na plataforma web do SisEnex.

## 1.2 Objetivo geral

Diante do exposto na problemática, o objetivo deste trabalho é apresentar uma proposta de implementação e execução de automação dos testes considerando o escopo do projeto Sisenex.

## 1.3 Objetivos específicos

1. Avaliar e viabilizar ferramentas de automação de teste;
2. Definir um processo de automação de testes para o SisEnex;
3. Verificar o impacto da automação de testes, com relação a redução de tempo e esforço.

## 1.4 Estrutura da monografia

Este trabalho divide-se em sete capítulos, incluindo este primeiro. No capítulo 2, encontra-se os conceitos gerais, apresentando o embasamento teórico necessário. O capítulo 3, apresenta o estado da arte, no que refere-se ao processo de implantação de testes automatizados e sua importância. O capítulo 4, trata da metodologia utilizada. No capítulo 5, encontra-se a proposta de processo de implantação de testes automatizados. O capítulo 6, aborda as dificuldades encontradas. Por fim, o capítulo 7 apresenta a conclusão do trabalho.

# 2 CONCEITOS GERAIS E REVISÃO DA LITERATURA

Para compreender a proposta do trabalho, é preciso conhecer alguns conceitos básicos envolvidos com a qualidade de software e o papel dos testes em relação a essa qualidade, dentre outros pontos correlacionados. Esse capítulo aborda os seguintes fundamentos: Qualidade de Software, Teste de Software, Desenvolvimento Orientado a Comportamento e Automação de Testes.

## 2.1 Qualidade de Software

O conceito de qualidade é muito amplo, podendo haver diferentes interpretações com base nos pressupostos e visão de mundo de cada indivíduo. Assim como em todas as áreas, a qualidade da entrega, seja do produto ou seja do serviço, é de extrema importância, não sendo diferente quando se trata de produtos de software.

Conforme Bartié (2002, p. 16) "Qualidade de Software é um processo sistemático que focaliza todas as etapas e artefatos produzidos com o objetivo de garantir a conformidade de processos e produtos, prevenindo e eliminando defeitos.". Com isso, a qualidade de software está estreitamente interligada às expectativas definidas previamente, desde o princípio da etapa da elaboração do processo de desenvolvimento do sistema até a entrega final do produto.

Pressman (2011, p. 359) afirma que "No desenvolvimento de software, a qualidade de um projeto engloba o grau de atendimento às funções e características especificadas no modelo de requisitos."

> Os requisitos de um sistema são as descrições do que o sistema deve fazer, os serviços que oferece e as restrições a seu funcionamento. Esses requisitos refletem as necessidades dos clientes para um sistema que serve a uma finalidade determinada, como controlar um dispositivo, colocar um pedido ou encontrar informações. (SOMMERVILLE, 2011, p. 57)

Conforme Sommerville (2011, p. 455) o gerenciamento de qualidade do sistema acaba fornecendo uma verificação do software independente do processo de desenvolvimento. O gerenciamento de qualidade tem como objetivo verificar se ao final da entrega do produto os requisitos estão em conformidade com as exigências do cliente.

A próxima subseção abordará com mais detalhes a importância dos testes para a qualidade do software, bem como, seus níveis e tipos.

## 2.2 Teste de Software

É perceptível que no decorrer dos anos a importância dos testes de software veio ganhando ainda mais destaque, sobretudo, com objetivo de se ter um maior controle de qualidade em relação ao sistema.

Para Pressman (2011, p. 402) "Teste é um conjunto de atividades que podem ser planejadas com antecedência e executadas sistematicamente. Por essa razão, deverá ser definido para o processo de software um modelo (template) para o teste."

> Uma estratégia de teste de software deve acomodar testes de baixo nível, necessários para verificar se um pequeno segmento de código fonte foi implementado corretamente, bem como testes de alto nível, que validam as funções principais do sistema de acordo com os requisitos do cliente. Uma estratégia deverá fornecer diretrizes para o profissional e uma série de metas para o gerente.(PRESSMAN, 2011, p. 402)

### 2.2.1 Níveis e tipos de teste de software

Os testes de software são divididos em diversos níveis e tipos, na qual variam de acordo com cada estágio do ciclo de desenvolvimento, logo, os testes podem ocorrer em quatro níveis, sendo eles: Unidade, Integração, Validação e Sistema.


**Figura 1: Espiral de estratégia de teste**
Fonte: PRESSMAN (2011, p. 404)

- Teste de Unidade: É o tipo de teste que visa testar pequenas partes de código do sistema, na qual possui comportamento já definido. Normalmente, é relacionado a funções para linguagens procedimentais e métodos em linguagens orientadas a objetos (BERNARDO, 2011).

- Teste de Integração: Para Pressman (2011, p. 409) " O teste de integração é uma técnica sistemática para construir a arquitetura de software ao mesmo tempo que conduz testes para descobrir erros associados com as interfaces."

- Teste de Validação: Conforme Pressman (2011, p. 416) o teste de validação se dá a partir do término do teste de integração, após todos os componentes serem testados e todos os erros relacionados corrigidos.

- Teste de Sistema: Pressman (2011, p. 418) afirma que "Teste de sistema é na realidade uma série de diferentes testes cuja finalidade primária é exercitar totalmente o sistema."

Além dos níveis descritos anteriormente, alguns tipos de teste foram definidos para que pudesse ser assegurada a qualidade não somente diante a validação da interface e a conformidade com os requisitos predefinidos pelo cliente, mas também diante da integridade e consistência do sistema.

A categorização dos tipos de teste pode ser feita com base no modelo FURPS que tem como atributos de qualidade: Funcionalidade, Usabilidade, Confiabilidade, Desempenho e Suportabilidade. Segundo Galitezi (2012) :

1. Funcionalidade

   Tem como objetivo verificar se o sistema está de acordo com os requisitos funcionais, além de avaliar o seu pleno funcionamento. Essa característica é subdividida em:

   - Teste de Função: Tem como objetivo validar o funcionamento do sistema verificando se há conformidade com os requisitos definidos.
   - Teste de Segurança: Visa verificar a proteção do sistema diante a acessos não autorizados, por exemplo.
   - Teste de Volume: Tem como objetivo verificar o comportamento do sistema diante a situações onde há um volume de dados muito grande, além de acessos no banco de dados em grande escala.

2. Usabilidade

   Tem como finalidade verificar a relação entre interface de usuário, prezando por uma boa experiência através da estética, acessibilidade, etc.

   - Teste de Usabilidade: Segundo Pressman (2011, p. 477) testes de usabilidade avaliam com que grau o usuário pode interagir com o sistema e ter uma consistência no retorno das suas ações.

3. Confiabilidade

   O atributo de confiabilidade subdivide-se em:

   - Teste de Integridade: Aborda o comportamento do sistema diante as falhas e verifica sua robustez.
   - Teste de Estrutura: Tem como objetivo avaliar a formação e o design do sistema como um todo.
   - Teste de Stress: Tem como objetivo analisar o comportamento do sistema diante a situações inesperadas como por exemplo, um grande número de acessos simultaneamente , recursos de hardware limitados e etc.

4. Desempenho

   - Teste de avaliação de desempenho: Refere-se a comparação do desempenho do sistema com um padrão de desempenho conhecido.
   - Teste de contenção: são testes que verificam se o sistema lida de forma aceitável com demandas com vários atores, compartilhando recursos simultaneamente, como por exemplo, compartilhamento de memória, etc.
   - Teste de carga: Pressman (2011, p. 486) afirma que "A finalidade do teste de carga é determinar como a WebApp e seu ambiente do lado do servidor responderá a várias condições de carga."
   - Perfil de desempenho: Teste em que há o monitoramento no fluxo de execução do sistema a fim de identificar possíveis dificuldades no processamento do mesmo.

5. Suportabilidade

   - Teste de configuração: Verifica como o software reage diante diferentes configurações de hardware.
   - Teste de instalação: Verifica o comportamento do software diante a instalação em situações adversas, como por exemplo, espaço de memória limitado.

   Conhecer e identificar os diversos tipos de teste é extremamente importante, tendo em vista a necessidade de aplicá-los corretamente de acordo com o estágio de desenvolvimento que o software se encontra. A subseção seguinte mostra a relação do desenvolvimento orientado a comportamento e o teste de software, indicando quais benefícios podem ser gerados adotando essa técnica.

## 2.3 Desenvolvimento Orientado a Comportamento (BDD)

BDD (*Behavior-Driven Development*) se trata de uma metodologia orientada a comportamento do software, onde seu foco é na linguagem e nas interações com o usuário. Amodeo (2015, p. 130) afirma que, " o BDD visa não só automatizar testes, mas também para melhorar a comunicação entre especialistas de domínio e desenvolvedores de software.".

O desenvolvimento orientado a comportamento evita uma documentação extensa e massiva, buscando proporcionar um desenvolvimento mais claro e de fácil compreensão por todos. A escrita do BDD se dá a partir da descrição de cenários relacionados a uma funcionalidade, apresentando o comportamento esperado, através da sintaxe: Dado, E, Quando e Então. A figura 2 representa um exemplo dessa sintaxe.

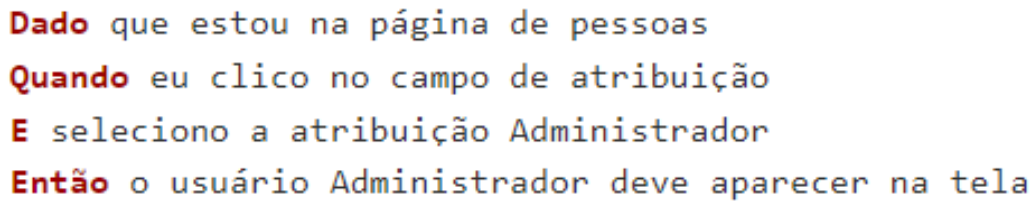


**Figura 2: Sintaxe BDD**
Fonte: Elaborado pelo autor.

Explicando cada conceito apresentado acima, temos:

- Dado (do inglês *Given*) : Se trata do contexto relacionado ao cenário em questão, uma precondição.
- Quando (do inglês *When*): Refere-se ao evento em si, ou seja, a ação efetivada.
- Então (do inglês *Then*): Trata-se do resultado da ação do autor.

Conforme Chelimskky et al. (2010), os três princípios básicos do BDD são:

(a) O suficiente é o suficiente: Trata-se da ideia de que não devemos fazer menos do que podemos fazer para iniciar o processo de desenvolvimento, nem mais que o necessário para que esforços não sejam desperdiçados, princípio que pode ser aplicado também ao processo de automação.

(b) Entregue valor às partes interessadas : Se o que está sendo desenvolvido não agrega valor ou não há perspectiva do aumento da capacidade de agregar valor, pare e faça algo a mais.

(c) É tudo comportamento: Independente do nível que se está, seja no código fonte ou seja na interface do usuário, a descrição do comportamento é o mesmo.

Segundo Smart (2015, p.59) "Os princípios do BDD são aplicáveis a todos os níveis de desenvolvimento de software, desde descoberta e definição de requisitos até a codificação de baixo nível e teste de regressão.". Ou seja, o BDD proporciona muitos benefícios, independentemente da fase de desenvolvimento do sistema, colaborando com uma maior integração da regra de negócio e código fonte.

A subsubseção seguinte apresenta a linguagem base que apoia a utilização da técnica BDD e suas características.

### 2.3.1 Gherkin

Para Amodeo (2015, p. 130) Gherkin é uma linguagem de domínio específico, especializada na descrição de recursos e cenários. Trata-se de uma linguagem de fácil entendimento, pois baseia-se na linguagem natural fazendo com que especialistas de domínio possam compreender e ler os recursos descritos sem exigir muito esforço.

A sintaxe Gherkin está estritamente relacionada à técnica de BDD principalmente quando refere-se a automação, pois, é ela que fornece a base para a aplicação do processo de desenvolvimento orientado a comportamento, padronizando a escrita das especificações dos cenários, com foco na regra de negócio e no comportamento esperado da funcionalidade.

Como já apresentada na subseção anterior, a figura 3 ilustra um exemplo da sintaxe utilizada pela técnica BDD, onde o esquema é composto pela identificação da funcionalidade associada (Efetuar login). Em seguida pelo cenário que descreve o comportamento esperado do sistema através das precondições (Dado que eu estou na tela de login do sistema) e (Quando eu insiro meus dados e clico em "Entrar"). Por fim, há a validação das ações anteriores, indicando o resultado esperado (Então o login deve ser realizado com sucesso).

Como podemos perceber o esquema da linguagem Gherkin é intuitivo e de fácil entendimento, podendo ser interpretado por qualquer pessoa sem grandes dificuldades, além de estar estritamente relacionada ao desenvolvimento orientado a comportamento. A próxima subseção apresenta o conceito da automação de testes, juntamente com seus benefícios.

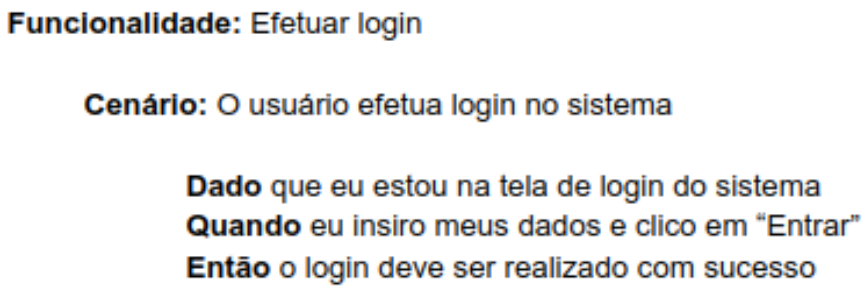


**Figura 3: Esquema BDD em Gherkin**
Fonte: Elaborado pelo autor.

## 2.4 Automação de Testes

A automação de teste é uma forma de proporcionar maior eficiência, rapidez e agilidade diante do processo de teste manual, que por sua vez, demanda tempo, e esforço, sobretudo em sistemas de maior porte, com uma tendência de menor efetividade em relação aos testes automatizados.

> Testes automatizados (em oposição aos testes manuais) é a prática de tornar os testes de software independentes de intervenção humana. Testar é uma prática intrínseca ao desenvolvimento de sistemas, mas testes de software só começaram a se tornar uma área de estudo da engenharia de software na década de 1970 e, desde então, têm ganho cada vez mais importância. (BERNARDO, 2011).

De acordo com Bernardo (2011) a independência da intervenção humana torna possível um maior aproveitamento dos recursos do computador, proporcionando benefícios como uma maior velocidade na execução, a execução paralela dos testes, reduzindo o tempo, flexibilidade na quantidade e momento das execuções dos testes, além da criação de casos de testes complexos com uma maior facilidade.

Para Sommerville (2011) " O uso de testes automatizados tem aumentado consideravelmente nos últimos anos. Entretanto, os testes nunca poderão ser totalmente automatizados, já que testes automáticos só podem verificar se um programa faz aquilo a que é proposto."(p. 147).

Embora a prática de automação de testes tenha conquistado um espaço importante devido aos seus diversos benefícios no decorrer dos anos, seguindo a evolução e a busca por uma maior qualidade na entrega do software, é necessário considerar em quais casos é viável a automação.

Ferreira (2018) afirma que o processo de automação de teste está distribuído em três grandes blocos, formando a Pirâmide de Automação de Testes, sendo eles:


**Figura 4: Pirâmide de Automação de Testes**
Fonte: FERREIRA (2018)

Conforme Ferreira (2018) na base da pirâmide, estão os testes unitários, pois eles são muitos, e normalmente são fáceis de implementar e baratos de manter. No meio da pirâmide estão os testes de integração, na qual, são responsáveis por efetuar a comunicação entre duas ou mais classes através de métodos, ou ainda permitir a comunicação do sistema com o banco de dados.

Para Ferreira (2018) no topo da pirâmide estão os testes de interface e aceitação, pois, são testes mais difíceis de serem implementados e mais caros de serem mantidos, visto que é necessário uma maior preparação para executá-los. Acima da pirâmide há a representação dos testes manuais que, segundo Ferreira (2018), " Eles dependem totalmente da intervenção humana desde a preparação da base de dados, são muito caros de serem construídos e mantidos. Todavia são importantes para avaliar a experiência do usuário."

### 2.4.1 Ferramentas para automação de teste

Para efetivar a realização da automação dos testes é necessário que um conjunto de ferramentas sejam adotadas, dessa forma, a escolha da ferramenta que irá apoiar o processo de automação é de extrema importância. Algumas das ferramentas que são tendência segundo Costa (2020), são:

1. Nível interface e aceitação

   - Selenium: Ferramenta utilizada para testes no nível de interface e aceitação, de grande popularidade no mercado. Selenium é composto pelas seguintes

ferramentas: Selenium IDE que trata-se de um ambiente integrado que facilita a criação dos scripts de teste, conta com Selenium WebDriver, na qual se utiliza o driver do próprio navegador para automatizar e normalmente utilizado para auxiliar na criação de scripts mais complexos, e por fim, Selenium Grid que permite a execução dos testes em diferentes máquinas, de modo remoto.

- Cypress: Trata-se de uma ferramenta para automação de testes de fácil instalação e utilização, que auxilia principalmente nos testes de interface e aceitação simulando as ações do usuário.
- Appium: Ferramenta destinada para testes funcionais de interface em plataformas móveis, IOS e Android.
- Cucumber: Conforme Bui (2018) " O Cucumber é uma ferramenta usada para executar testes de aceitação automatizados que foram criados em um formato BDD."

2. Nível integração

   - Rest-Assured: Segundo Mesquita (2020) Rest-Assured é uma ferramenta que tem como objetivo facilitar a criação dos testes automatizados para APIs Rest, auxiliando na validação das requisições HTTP.
   - Postman: Costa (2020) afirma que é uma plataforma de colaboração para desenvolvimento de API, na qual permite o envio de uma requisição e valida a resposta.

3. Nível unitário

   Existem muitos frameworks que se propõem a entregar uma boa performance com relação aos testes unitários de acordo com a linguagem que deseja-se utilizar. A tabela a seguir mostra algumas dessas ferramentas, listando algumas de suas principais características.

**Tabela 1: Ferramentas para testes unitários**

| Linguagem | Framework |
|---|---|
| PHP | PHPUnit |
| JavaScript | Jest |
| Ruby | RSpec |
| Python | Pytest |

   - PHPUnit: Tem como seu objetivo principal criar testes unitários na linguagem PHP, detectando falhas tanto de lógica como de digitação.
   - Jest: Criado para realizar testes utilizando Javascript, tendo como um dos seus pontos fortes a simplicidade na implementação dos testes.

- RSpec: Ferramenta que auxilia na criação dos testes em Ruby, sobretudo, para testes BDD, contando com uma boa documentação.
- Pytest: Ferramenta muito utilizada quando se trata de linguagem Python, permite integração com diversas bibliotecas e conta com uma sintaxe de fácil entendimento.

Embora haja muitas outras ferramentas e frameworks que auxiliam na automação dos testes, esse trabalho detalha algumas dessas, por serem mais apropriadas ao seu contexto.

# 3 ESTADO DA ARTE

Sabendo que, esse trabalho tem como objetivo apresentar uma proposta de processo de automação de teste, esse tópico refere-se a trabalhos presentes na literatura relacionados ao processo de automação e a sua importância para a qualidade de software.

Bernardo e Kon (2008) trazem uma reflexão sobre a importância dos testes automatizados, onde controlar a qualidade do software é um grande desafio devido à alta complexidade dos sistemas e dificuldades relacionadas ao processo de desenvolvimento. Em seu trabalho, é apresentada uma abordagem convencional de desenvolvimento dos testes, que consiste na execução de forma manual. A execução de um caso de teste manualmente pode ser rápida e fácil, mas se levada em consideração execuções repetitivas de forma manual percebe-se que acarreta um esforço muito grande, tornando-se uma tarefa exaustiva.

Além da abordagem convencional é apresentada ainda uma nova abordagem segundo o autor, utilizando testes automatizados, ressaltando algumas vantagens, tais como, capacidade maior de reprodutividade e a possibilidade de criar e executar casos de teste mais complexos do que os elaborados manualmente.

Segundo Mahajan, Shedge e Patkar (2016), os estágios da automação de teste, são: análise da viabilidade da automação de teste, seleção das ferramentas que irão apoiar no processo de automação, implantação e execução dos scripts, além da manutenção dos testes. A necessidade de se automatizar também é abordada, onde consiste na análise de alguns critérios, como por exemplo, a estabilidade do ambiente. Benefícios dos testes automatizados sobre os testes manuais também são apresentados no trabalho de Mahajan, Shedge e Patkar (2016), tais como:

- Redução no tempo gasto nos testes;
- Maior capacidade de repetitividade;
- Confiabilidade;
- Scripts reutilizáveis;
- Programável;
- Proporciona uma maior cobertura para os testes de regressão;
- Aumenta a produtividade;
- Registros das execuções dos testes com mais detalhamento;
- Execução dos scripts em diversas plataformas;
- Melhoria na redução dos custos.

A melhor visualização da comparação dos testes automatizados e testes manuais pode ser observada na tabela 2 abaixo:

**Tabela 2: Teste automatizado vs Teste manual**

| Informação | Teste automatizado | Teste manual |
|---|---|---|
| Tipos de teste | Teste de Regressão | Teste de Usabilidade |
| Velocidade da execução | Execução rápida | Mais lento que o teste automatizado |
| Sequência | Depois do teste manual | Antes do teste automatizado |
| Recursos | Utilizando a ferramenta é dispensado grande número de recursos humanos | Grande número de recursos humanos é obrigatório |

Mesmo havendo muitos benefícios proporcionados pela automação de testes, é preciso levar em consideração as dificuldades que envolvem o processo de implantação dos testes automatizados diante ao contexto que se está inserido.

Silva, Alves e Bruno (2014) trazem uma reflexão sobre as dificuldades da implantação do processo de automação, bem como no desenvolvimento de uma aplicação, levando em consideração uma equipe inexperiente, com altos custos iniciais e com falta de mão de obra especializada para determinada atividade. Alguns fatores que dificultam o processo de implantação dos testes automatizados, principalmente em empresas, são com relação a compra e alto investimento em ferramentas para a criação e execução dos scripts, treinamento da equipe e contratação de mão de obra qualificada.

Alguns critérios de avaliação devem ser levados em consideração, um deles é a escolha correta da ferramenta que apoiará no processo de automação. Segundo Silva, Alves e Bruno (2014) em um contexto geral podem ser considerados os critérios de avaliações:

- Grau de complexidade da ferramenta;
- Custo da ferramenta;
- Custo de treinamentos e a aprendizagem da aplicação;
- Reutilização do código automatizado do teste;
- Tipo de plataformas a serem utilizadas;
- Maior cobertura para os testes de regressão.

Os trabalhos apresentados nesta seção foram tomados como base para a proposta de implantação e execução dos testes automatizados desenvolvida, bem como são citadas dificuldades semelhantes enfrentadas durante o processo de desenvolvimento que será apresentado nas seções seguintes deste trabalho.

# 4 PROCEDIMENTOS METODOLÓGICOS

Considerando a grande importância dos testes de software na busca pela qualidade, esse trabalho aborda a seguinte problemática: a implementação do processo de automação de teste é factível em um software, desenvolvido em um ambiente acadêmico? Considerando as limitações de tempo e recurso, o presente trabalho mostra a implementação de testes automatizados a nível de aceitação, bem como a execução dos scripts de teste. Para isso, foi definido um processo de automação de testes que posteriormente foi aplicado no produto SisEnex desenvolvido no Laboratório de Computação Ubíqua e Móvel (LUMO), na qual, se trata de um laboratório de desenvolvimento, pesquisa e extensão nas áreas de computação ubíqua, computação móvel, bem como áreas correlatas, vinculado ao Centro de Informática (CI) da Universidade Federal da Paraíba (UFPB).

A proposta do processo de testes no SisEnex levou em consideração algumas restrições. Com isso, foi definido um processo baseado em quatro etapas, sendo elas: Planejar, especificar, executar e encerrar (FREITAS, 2020). A figura 5 mostra com mais detalhes a visão do processo implantado, bem como as etapas citadas anteriormente.

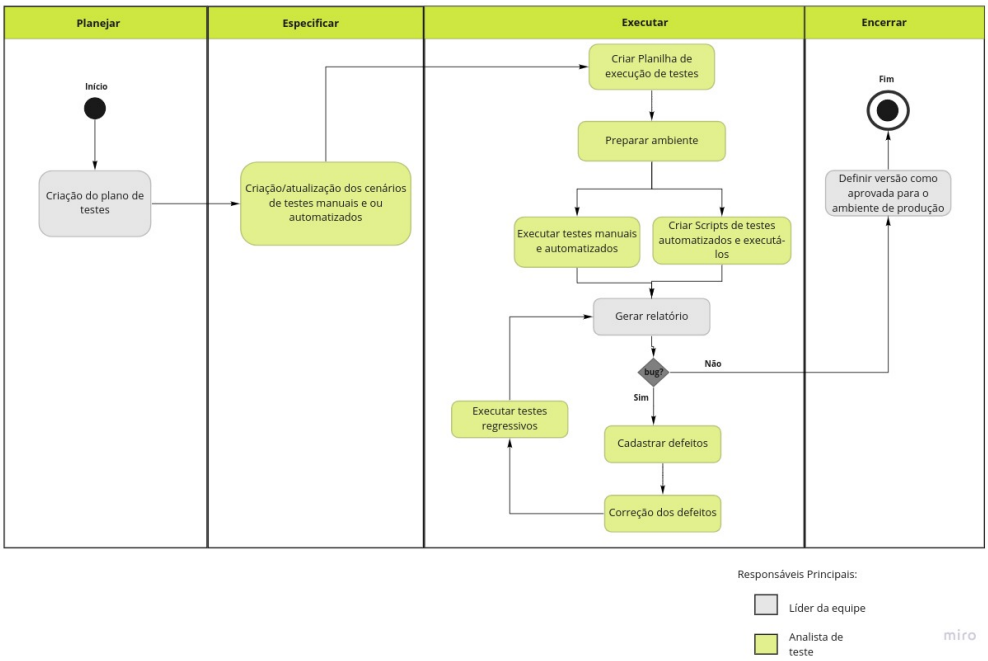


**Figura 5: Visão geral do processo de testes**
Fonte: FREITAS (2020)

Segundo Freitas (2020, p. 44) "A etapa de planejar é responsável pelo planejamento dos testes, onde será definida a estratégia de testes para o projeto e criado o plano de testes."

O planejamento dos testes trata-se de uma etapa de extrema importância, sendo um dos pilares responsáveis pelo sucesso do processo de teste em um sistema.

A etapa de especificar refere-se a criação dos cenários e casos de testes, na qual serão executados ou de forma manual ou automatizada além de ser criado projeto de teste (FREITAS, 2020).

Freitas (2020, p.43) aponta que na etapa de execução devemos criar e manter as planilhas de execução com os cenários e casos de testes, bem como devemos criar scripts de testes automatizados, executar os cenários e casos testes manualmente e automaticamente, além de gerar relatórios e efetuar o cadastramento de bugs.

Para Freitas (2020, p.44) "A etapa de encerramento, nela a versão é aprovada para ir para o ambiente de produção, o time de testes aprova uma versão quando esta passou por todos os critérios de aceite definidos pelo processo de testes [...]".

Com o processo de teste bem definido, foi possível o planejamento para que a inserção dos testes automatizados obtivesse sucesso. Para isso, os passos executados para que fosse possível a realização do trabalho foram:

(a) Definir um processo de automação dos testes.

Foi elaborada uma proposta de automação considerando as limitações impostas, onde a escolha das ferramentas também foi dada levando em consideração alguns critérios para que a proposta de processo de automação não fosse comprometida no decorrer do seu desenvolvimento. Alguns dos critérios foram:

i. Ser gratuita;
ii. Ser de fácil configuração e instalação.

(b) Selecionar os casos de testes aptos a serem automatizados.

Como parte do processo de automação, um conjunto de casos de teste foram selecionados com base na prioridade definida, neste caso, os casos de teste a nível de aceitação.

(c) Implementar os testes automatizados.

A partir da seleção dos casos de teste já existentes, foram criados cenários e scripts de testes automatizados. Os scripts foram desenvolvidos com apoio do framework Cypress, além do gerenciador de repositório e versões GitLab.

(d) Executar testes automatizados.

Nessa atividade foi efetivada a execução dos testes, onde os resultados foram registrados em planilhas, bem como no repositório do GitLab.

(e) Analisar os resultados.

Ao final da execução dos testes automatizados foi feita uma análise a fim de verificar os benefícios da automação no projeto .

As próximas seções mostraram de forma mais detalhada a realização dessas etapas.

# 5 ESTUDO DE CASO

As próximas subseções apresentam a definição da proposta de automação de testes e as atividades que a compõem, desenvolvidas para o SisEnex.

## 5.1 O produto SisEnex

O SisEnex surgiu da necessidade de agilizar a organização do Encontro de Extensão, tendo em vista que são mais de 2000 pessoas envolvidas no evento. Uma das maiores dificuldades encontradas na execução do evento era com relação ao quantitativo de folhas de avaliação geradas e o esforço demandado para consolidar as notas de forma manual, com isso, o processo se tornava sobretudo demorado. A avaliação se dava com base em oito critérios e os projetos divididos em oito áreas temáticas, considerando a avaliação em pares.

Esse produto foi desenvolvido na linguagem de programação JavaScript com apoio da biblioteca React, contou ainda com diversos frameworks, sendo subdividido a nível de desenvolvimento: Backend, Frontend e Testes. A nível de usuário contou com módulos de: Avaliador, Monitor e Gerente, sendo este último disponível apenas na plataforma Web.

O Avaliador é o usuário responsável por inserir as notas de acordo com cada critério de avaliação. O Monitor é o usuário responsável por acompanhar o processo de avaliação, prestando suporte durante o evento. O usuário Gerente por sua vez, organiza o evento em si, e acompanha os status de todas avaliações.

O produto vem sendo desenvolvido desde 2019, onde a versão de 2019 contemplava as plataformas web e móvel, tendo um foco maior na plataforma móvel. Com a pandemia de 2020, o sistema sofreu algumas alterações para atender o novo cenário, sendo assim, algumas das funcionalidades adicionadas foram:

- Criação do módulo avaliador na plataforma web.
- Avaliação de artefatos, como vídeos e videochamadas, contribuindo para que as avaliações fossem feitas de forma remota.

O projeto era composto por três times, somando 18 pessoas, sendo 4 desenvolvedores mobile, 3 desenvolvedores frontend, 4 desenvolvedores backend, 4 analistas de teste e 3 designers. A execução do projeto era dividida diante a reuniões de alinhamento semanais e os testes eram realizados a cada nova versão liberada, tanto na versão web quanto na versão mobile.

A documentação que prestou apoio ao desenvolvimento do projeto incluía casos de testes, especificação dos requisitos e estágios de teste, detalhamento necessário devido à complexidade da regra de negócio envolvida.

## 5.2 Processo de automação

O processo de automação consiste em definir as ferramentas que irão auxiliar na escrita e na execução dos scripts de teste automatizados, onde a escolha é feita de forma particular na qual é necessário analisar o contexto em que se está inserido, buscando atender da melhor forma a necessidade de cada processo. Selecionar os casos de teste candidatos a automação, ou seja, definir em qual escopo será aplicado, também é uma das etapas que compõem a automação de testes. Planejar e desenvolver os scripts propriamente, essa etapa engloba a configuração do ambiente para que esteja apto de forma que os scripts sejam escritos, contando também com a escrita dos scripts de teste automatizados de fato.

Os scripts de teste automatizados serão executados na etapa seguinte do desenvolvimento, ou seja, na etapa de execução dos testes, onde os scripts criados irão receber dados de entrada, processar esses dados e por fim, sinalizar se houve sucesso ou falha na execução. Finalizando o processo de automação se encontra a etapa de reportar das falhas encontradas.

### 5.2.1 Objetivos de Automação

A pretensão pela automação dos testes surgiu em 2019, porém devido algumas limitações não foi possível obter êxito. A equipe de testes por meio de reuniões analisou as necessidades e limitações que envolveriam o processo de automação para o SisEnex. Algumas das limitações identificadas foram: necessidade de ferramentas e frameworks gratuitos tendo em vista que, o laboratório não conta com recursos financeiros destinados para tal finalidade, além da priorização de ferramentas de fácil configuração e instalação para que a automação fosse viável para todos integrantes da equipe.

Alguns problemas motivaram a implementação da automação dos testes do SisEnex, entre eles o grande esforço demandado para realizar os testes nas duas plataformas de maneira manual, tendo em vista que as equipes eram formadas por estudantes, logo, não havia dedicação plena às atividades que envolviam o projeto. Diante a essa situação, os objetivos de automação se basearam na otimização do tempo de execução dos testes funcionais a nível de sistema, bem como na redução do esforço aplicado.

## 5.3 Proposta de processo de automação

Considerando o contexto apresentado anteriormente, foi pensado em um processo de automação que considerasse as limitações impostas de recursos e ferramentas, tendo em vista que, as ferramentas utilizadas para dá apoio a automação

deveriam ser gratuitas e open source. Foi levado em consideração também que fosse incluído de uma forma mais simples dentro do processo geral. Aplicando o processo de automação para o SisEnex a figura 6 mostra como foi definido o processo de automação de testes, onde as atividades relacionadas a automação foram desenvolvidas de forma paralela a execução dos testes manuais.

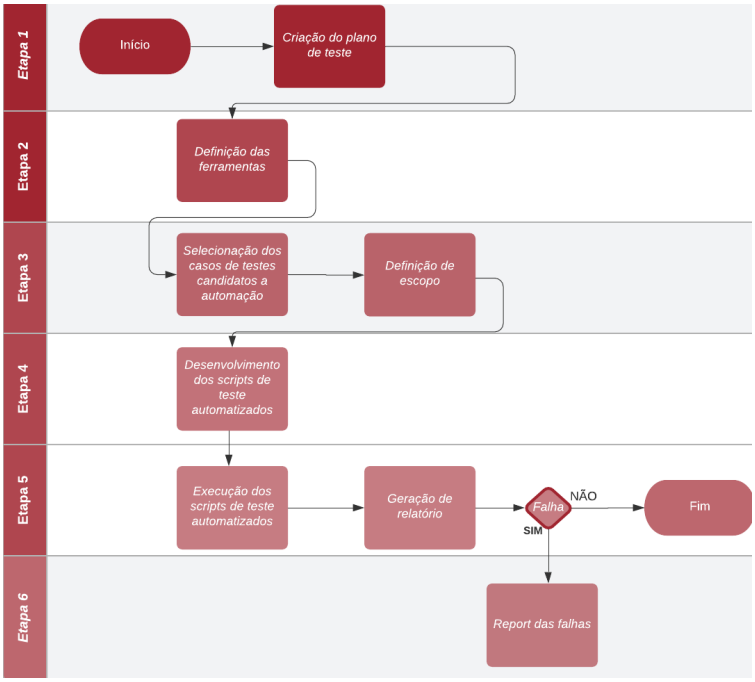


**Figura 6: Processo de automação de testes no SisEnex**
Fonte: Elaborado pelo autor.

Nas próximas subseções serão apresentadas as atividades relacionadas ao processo de automação e a maneira que foram executadas ao longo do projeto.

### 5.3.1 Seleção das ferramentas de automação

A seleção das ferramentas se deu com base na análise do contexto do SisEnex, levando em consideração diversos fatores. Para que fosse escolhida, a ferramenta deveria seguir alguns critérios como, ser gratuita, open source e de fácil configuração e aprendizagem. Para isso, alguns estudos foram feitos destacando os pontos positivos e negativos das ferramentas disponíveis, ao final, em comum acordo a equipe optou pela escolha das seguintes ferramentas: Cypress e Cucumber.

Cypress é uma ferramenta open source, utiliza-se da linguagem de programação JavaScript e suporta navegadores como Google Chrome, Firefox e Electron. Essa ferramenta é utilizada nos testes a nível de aceitação e interface, contudo, é possível utilizá-la também para testes de integração (teste de APIs).

É uma ferramenta que vem se tornando tendência por diversos motivos, entre eles, a facilidade no processo de instalação e configuração do ambiente, além de visar uma construção de scripts de teste mais simples, objetivos e intuitivos.

Conforme José (2020) algumas vantagens de se utilizar o Cypress são:

- O *setup* contém diversas definições por padrão, facilitando no momento da criação dos testes.
- As avaliações são mais rápidas e assertivas.
- Os testes são mais compactos.
- Possibilita a execução de uma *feature* específica.
- Sincroniza as alterações de forma assistida, revalidando as modificações efetuadas.
- Disponibiliza a integração ao repositório do *GitHub* e *GitLab*.
- Permite a gravação testes, através de vídeos da execução, além de capturas de tela.
- Viabiliza testes de aceitação através de simulações de API quando necessário.
- Dispõe da funcionalidade *Time Travel* que tem como objetivo identificar e remover possíveis erros a partir da própria interface da ferramenta.

Conforme Bui (2018) Cucumber é uma ferramenta utilizada com o intuito de executar testes de aceitação automatizados que foram criados em um formato BDD através da linguagem Gherkin, com descrições intuitivas e de fácil entendimento.

O Cucumber facilita na comunicação entre todos os membros envolvidos, incluindo desenvolvedores, testadores e clientes. Bui (2018) afirma que “Além de

melhorar a comunicação entre os membros de um mesmo time de testes, o Cucumber também ajuda a aproveitar as habilidades do testador com eficiência."

Conforme Silva (2015), para testes automatizados com grande valor de negócio Cucumber consiste em duas faces, sendo elas: Face de negócio e Face de tecnologia. A face de negócio é apresentada na figura 7, e consiste de um arquivo contendo as funcionalidades e os cenários com os passos escritos em português.

```
Feature: [FT- 009] Pesquisar apresentação

    Scenario: CEN-01: Gerente pesquisa apresentacao por codigo e visualiza
    Given Sou um gerente e estou na página de apresentacoes
    When Eu insiro o codigo no campo pesquisar e solicito
    Then O codigo deve ser checado
```

**Figura 7: Exemplo face de negócio**
Fonte: Elaborado pelo autor.

Para Silva (2015) a face de tecnologia consiste em definições de passo, código de suporte e de biblioteca automação. A definição do passo por sua vez é implementada por meio de uma linguagem de programação diante cada passo do cenário nos arquivos de extensão .feature.

A figura 8 representa a face de tecnologia, que no exemplo é implementada em javascript. Os métodos apresentados consistem na definição dos passos descritos no arquivo de funcionalidade, que foram mostrados na figura 7.

```
import ApresentacaoPage from '../pageobjects/ApresentacaoPage'
import PesquisaPage from '../pageobjects/PesquisaPessoaPage';
import AcessoPage from '../pageobjects/AcessoPage';

const Apresentacao = new ApresentacaoPage;
const Pesquisa = new PesquisaPage
const Acesso = new AcessoPage;

//Scenario: Gerente pesquisa apresentacao por codigo
Given ("Sou um gerente e estou na página de apresentacoes", ()=>{
    Acesso.acessar('apresentacoes')
})

When ("Eu insiro o codigo no campo pesquisar e solicito", ()=>{
    Pesquisa.pesquisa('84717244')
})

Then ("O codigo deve ser checado", ()=>{
    Apresentacao.verificarTexto('84717244','84717244')
})
```

**Figura 8: Exemplo face de tecnologia**
Fonte: Elaborado pelo autor.

### 5.3.2 Seleção dos casos de teste para automação

Foram selecionados inicialmente apenas os casos de teste relacionados ao módulo Gerente, pois até então o módulo de Avaliador estava disponível apenas na versão mobile do SisEnex, motivo no qual as avaliações dos anos anteriores eram feitas de forma presencial, não havendo a necessidade de avaliadores na web. Como foi citado anteriormente, diante a pandemia de 2020 houve a necessidade de que alguns ajustes fossem feitos, como por exemplo a implementação do módulo avaliador na plataforma web, para que as avaliações fossem feitas de forma remota, respeitando o momento de distanciamento social vivido em todo o mundo. Com isso, os casos de teste relacionados ao módulo Avaliador, juntamente com os casos de teste do módulo Gerente, ambos na plataforma web. A criação do módulo de Avaliador Web facilitou, bem como contribuiu para o sucesso na realização de todo evento.

Através dos objetivos de automação, os casos de teste selecionados para automação foram os relacionados a navegação e aceitação, simulando as ações do usuário, divididos por módulos, sendo eles: Módulo Gerente e Módulo Avaliador web, mostrados nas tabelas 1 e 2 abaixo.

**Tabela 3: Módulo de Gerente**

| Gerente | | |
|---|---|---|
| TC | CEN | Descrição |
| TC01 - Pesquisar Projeto | CEN01 | Pesquisar projeto por código |
| | CEN02 | Pesquisar projeto por título |
| | CEN03 | Pesquisar projeto inexistente por código |
| TC02 -Filtrar Projeto | CEN04 | Filtrar projeto por área temática |
| | CEN05 | Filtrar projeto por modalidade |
| | CEN06 | Filtrar projeto por alocação |
| | CEN07 | Filtrar projeto por status |
| TC03 - Visualizar Projeto | CEN08 | Visualizar projeto na página inicial |
| | CEN09 | Pesquisar e visualizar projeto |
| TC04 - Editar Projeto | CEN010 | Editar modalidade do projeto |
| | CEN011 | Adicionar projeto a uma apresentação |
| TC05 -Pesquisar Apresentação | CEN012 | Pesquisar apresentação por código |
| TC06 -Filtrar Apresentações | CEN013 | Filtrar por categoria e-Tertúlia |
| | CEN014 | Categoria vídeo |
| | CEN015 | Área temática Comunicação |
| | CEN016 | Área temática Cultura |
| | CEN017 | Área temática Direitos Humanos e Justiça |
| | CEN018 | Área temática Educação |
| | CEN019 | Área temática Meio Ambiente |
| | CEN020 | Área temática Saúde |
| | CEN021 | Área temática Tecnologia e Produção |
| | CEN022 | Área temática Trabalho |
| | CEN023 | Modalidade Tertúlia |
| | CEN024 | Modalidade Performance |
| | CEN025 | Estado Em Execução |
| | CEN026 | Estado Pendente |
| | CEN027 | Estado Finalizada |
| TC07 - Visualizar Apresentação | CEN028 | Visualizar na página inicial |
| TC08 - Pesquisar Pessoa | CEN029 | Pesquisar por número de matrícula |
| | CEN030 | Por número de telefone |
| | CEN031 | Por email |
| | CEN032 | Usuário por nome |
| TC09 - Filtrar usuário por atribuição | CEN033 | Filtrar pessoa com uma atribuição |
| TC010 - Mudar Atribuição | CEN034 | Mudar atribuição com sucesso |
| | CEN035 | Mudança de atribuição não permitida |

**Tabela 4: Módulo do Avaliador**

| Avaliador | | |
|---|---|---|
| TC | CEN | Descrição |
| TC011 - Visualizar artefatos | CEN036 | Visualizar artefato vídeo |
| | CEN037 | Particiár da conferência de uma e-tertúlia |
| TC012 -Avaliar projetos | CEN38 | Avaliar projeto sem inserir comentário |
| | CEN39 | Avaliar projeto inserindo comentário |
| | CEN40 | Inserir mais de duas casas decimais em um critério |
| | CEN41 | Inserir nota 100 a todos os critérios |
| | CEN42 | Inserir nota 8,785 a todos os critérios |
| | CEN43 | Inserir nota 100 no último critério |

As próximas seções irão descrever as atividades de desenvolvimento com relação aos casos de teste selecionados.

### 5.3.3 Desenvolvimento dos scripts de teste automatizados

Os scripts de teste automatizados foram desenvolvidos com o auxílio do ambiente de desenvolvimento integrado Visual Studio Code. Inicialmente os cenários de teste foram criados e divididos por features relacionadas aos casos de teste já existentes, seguindo a técnica BDD para facilitar a implementação.

```
1  Feature: [FT-011] Visualizar Apresentação
2
3      Scenario: CEN-01: Visualizar apresentação na página inicial
4          Given sou um gerente e estou na página de apresentação
5          When clico em visualizar uma apresentação
6          Then a apresentação deve ser exibida
```

**Figura 9: Cenário de teste para o caso de teste Visualizar apresentação**
Fonte: Elaborado pelo autor.

A figura 9 representa a estrutura dos cenários, tomando como exemplo o caso de teste Visualizar apresentação. A descrição da funcionalidade que engloba os cenários, se dá pela palavra reservada *Feature*, proveniente da sintaxe da linguagem gherkin, seu objetivo é apresentar de forma clara qual a finalidade do teste de um modo geral. *Scenario* contém a descrição do cenário que será executado. *Given* se trata da pré condição, no caso da figura acima a pré condição é que o usuário tenha permissão de gerente e esteja na página de apresentação. *When* trata-se de ação de fato que será realizada. Por fim, a palavra reservada *Then*, na qual refere-se ao resultado da ação realizada, onde no exemplo acima remete a visualização da apresentação propriamente.

A estruturação dos scripts de teste se deu a partir da utilização do Cypress juntamente com Cucumber, ambas ferramentas descritas na subseção 5.3.1. Algumas pastas foram criadas inicialmente para auxiliar na melhor organização do desenvolvimento, entre elas:

- integration: Pasta responsável por armazenar os arquivos .feature relacionados ao formato BDD.
- plugins: Tem como objetivo comportar as configurações do Cucumber.
- support: Pasta responsável por comportar os scripts criados.
- node_modules: Armazena os arquivos relacionados ao funcionamento do Cypress bem como do Cucumber.

Além das pastas citadas anteriormente foi necessário a criação de algumas pastas adicionais para a efetivar a iniciação do projeto. A escolha pela adoção das pastas adicionais se deu a partir da necessidade em manter um padrão de organização, bem como a concentração dos elementos que compõem a sintaxe utilizada pelo Cypress alinhada à técnica de BDD. As pastas adicionais criadas foram:

(a) pageobjects: Pasta destinada ao armazenamento dos scripts .js escritos com o Cypress relacionados a cada fluxo de teste desejado.

(b) steps: Pasta responsável armazenar os arquivos de conexão entre o Cypress e o Cucumber.

A figura 10 mostra os scripts de teste implementados a partir da estrutura mostrada acima, tomando como exemplo o caso de teste Visualizar apresentação. Utilizando a linguagem de programação Javascript, a classe VisualizarApresentacaoPage foi criada, contendo os métodos necessários para o acesso à página de apresentações com auxílio de elementos próprios do Cypress.

```
/// <reference types="Cypress" />

class VisualizarApresentacaoPage{

    clicaApresentacao(key){
    cy.on('uncaught:exception', (err, runnable) => {
     return false
     })
     cy
     .get('[data-row-key='+key+'] > [style="text-align: center;"] > [style="margin: 10px;"] > .ant-btn')
     .click();
    }
    showApresentation(apresentacao){
        cy.get('.ant-modal-body > :nth-child(1) > :nth-child(1)').should('contain', apresentacao)
    }
    }
export default VisualizarApresentacaoPage;
```

**Figura 10: Script de teste relacionado a conexão Cypress e Cucumber**
Fonte: Elaborado pelo autor.

A figura 11 representa a classe contida na pasta Steps, que tem como objetivo a conexão entre os elementos do Cypress definidos da figura anterior e o Cucumber, contendo a funcionalidade e o cenário em questão, ambos apresentados anteriormente.

```
import VisualizarApresentacaoPage from '../pageobjects/VisualizarApresentacaoPage'
import AcessoPage from '../pageobjects/AcessoPage'
import TesteVisualizarProjetoPage from '../pageobjects/VisualizarProjetoPage'


const acesso = new AcessoPage;
const visualiza = new VisualizarApresentacaoPage;

Given("sou um gerente e estou na página de apresentação", ()=> {
    acesso.acessar('apresentacoes');
})
When("clico em visualizar uma apresentação", ()=> {
    visualiza.clicaApresentacao('18372984');

})

Then("a apresentação deve ser exibida", ()=> {
    visualiza.showApresentation('18372984')
})
```

**Figura 11: Script de teste relacionado ao fluxo de Visualizar apresentação**
Fonte: Elaborado pelo autor.

A subseção a seguir, apresenta como foi feita a execução dos scripts de testes, mostrando mais detalhes do desenvolvimento dessa etapa.

### 5.3.4 Execução dos testes automatizados

Diante a criação dos casos testes e posteriormente seus respectivos cenários, iniciou-se a execução dos testes automatizados nos módulos Gerente e Avaliador do sistema SisEnex. Foram executados 43 cenários de teste utilizando o navegador Google Chrome.

A execução dos testes contou com o suporte do Cypress integrado ao Vscode, com isso, foi possível o acompanhamento dos resultados através da interface do próprio framework, contendo seu estado de sucesso ou falha conforme mostrado na figura 11.

O teste inicia-se verificando a pré-condição representada por Given na linguagem natural Gherkin, e segue conforme a estrutura que foi escrito no cenário de teste. Os comandos são executados de forma interativa e visualizados em tempo de execução. No exemplo abaixo observa-se que o cenário executado é descrito por: Gerente realiza mudança de atribuição não permitida.

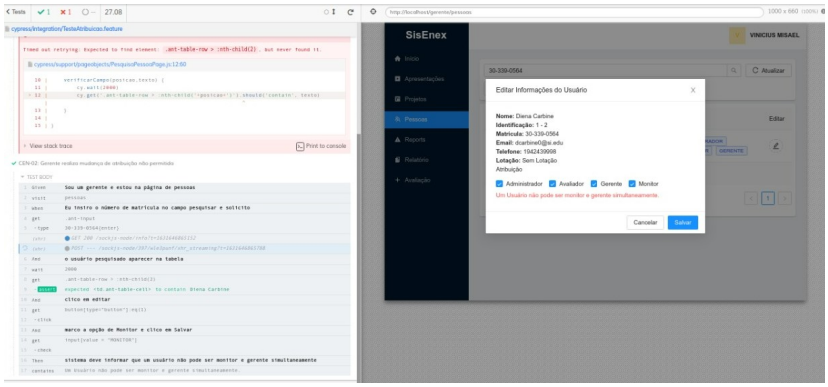

**Figura 12: Resultado da execução do caso de teste Mudar Atribuição**
Fonte: Elaborado pelo autor.

A integração dos testes automatizados com o pipeline de desenvolvimento do sistema se deu a partir do fluxo, onde, os testes automatizados eram executados todas as vezes que uma nova release era passada da branch Devel para a Stagging e da Staging para a Master. Branchs são ramificações do projeto e auxiliam no isolamento da parte estável do código fonte. No projeto tiveram branchs locais, utilizadas pelos desenvolvedores e testadores para o desenvolvimento dos scripts e todo o código fonte do sistema de forma local. A branch Devel também foi utilizada, de modo que, após desenvolver os scripts localmente era necessário passar todo o código de forma que qualquer pessoa do time tivesse acesso remotamente através do Gitlab. A branch Stagging contemplava a versão mais estável do código. Por fim, a branch Master, onde havia o código fonte que iria para produção, ou seja, a versão do código mais estável e aceita, apta para ser lançada e utilizada pelo usuário final.

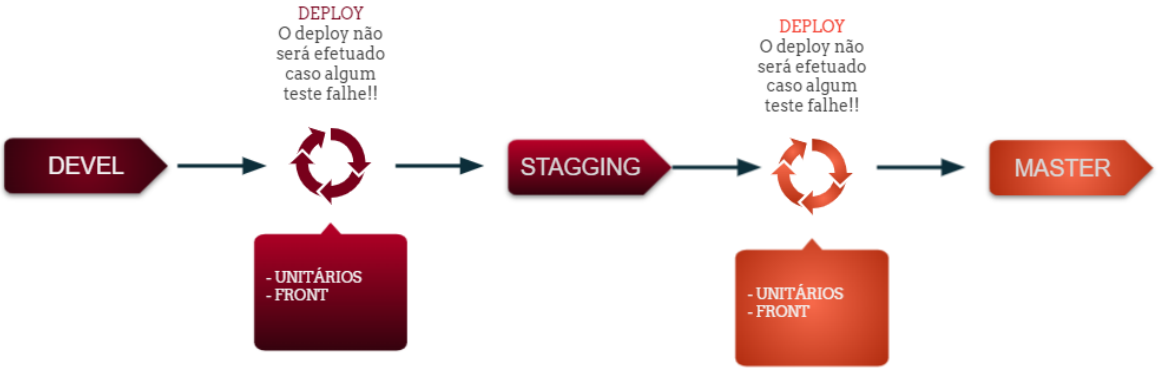


**Figura 13: Fluxo de testes**
Fonte: Adaptado de FREITAS (2020).

A figura 13 mostra o fluxo de teste, em que, os testes automatizados eram executados a cada tentativa de atualização das branchs, seguindo como critério as falhas geradas ao final de cada execução. Caso algum cenário falhasse o *deploy*[1] para a branch seguinte não era permitido.

[1]Deploy é o processo de implantar as modificações que já foram concluídas no sistema/projeto.

A próxima subseção abordará os resultados obtidos após a execução dos scripts de teste automatizados, com ênfase no relatório gerado a partir do pacote do Cucumber.

### 5.3.5 Resultados e métricas

Diversas versões foram geradas da Devel para Stagging e da Stagging para a Master. Em média foram geradas 10 versões da Devel para a Stagging e 4 versões geradas da Stagging para Master. Como o processo de automação de testes foi incluído no pipeline de desenvolvimento, a execução desse, proporcionou diversos benefícios como, ganho de tempo, comparando com a execução manual dos testes.

A visualização e análise dos testes se deu a partir dos resultados contidos em planilhas e relatórios conforme os ciclos de execução de cada módulo. Os resultados relacionados ao avaliador web, foram registrados em planilhas de execução Excel e contaram com 5 ciclos, 2 para a versão 1.0 e 3 para a versão 2.0, com o objetivo de validar a versão testada. Já os resultados relacionados ao gerente web foram registrados com auxílio do pacote *cucumber html report* que possibilitou a geração de relatórios automáticos de fácil visualização e interpretação como apresentado na figura 14. Esse pacote permite a geração dos relatórios no formato html que pode ser visualizado em qualquer navegador, com ele é possível visualizar as *features* que foram executadas, bem como quais cenários foram executados e obtiveram sucesso ou falha.


**Figura 14: Report gerado automaticamente pelo Cucumber**
Fonte: Elaborado pelo autor.

Na figura 14 é possível visualizar os resultados de uma execução do ciclo de testes automatizados. Como se observa, o detalhamento dos resultados é apresentado de forma objetiva e clara usando as cores para indicar falhas e sucessos

(vermelho e verde, respectivamente), e o número contendo o conjunto de cenários testados. Observa-se que, foram executados 35 cenários de teste, onde, 31 passaram (88,6 %) e 4 falharam (11,4 %).No caso específico dessa execução, as falhas ocorridas não foram necessariamente relacionadas a bugs do sistema, mas sim, relacionadas a limitações na implementação e execução dos testes, tendo em vista que, os casos de teste automatizados foram desenvolvidos e testados em ambiente de desenvolvimento tornando algumas informações da base de dados inconsistentes.

Além da visualização dos cenários de teste, é possível visualizar de forma abrangente a relação das features que obtiveram sucesso ou fracasso durante a execução dos testes automatizados. Por exemplo, na figura 15, temos o teste para feature Editar Projeto.

A figura 15 mostra com maior detalhamento os cenários que falharam após a execução dos testes automatizados, bem como o número de ocorrências dos estados de: Sucesso, Falha e Salto, diante a cada etapa dos passos que regem o BDD.

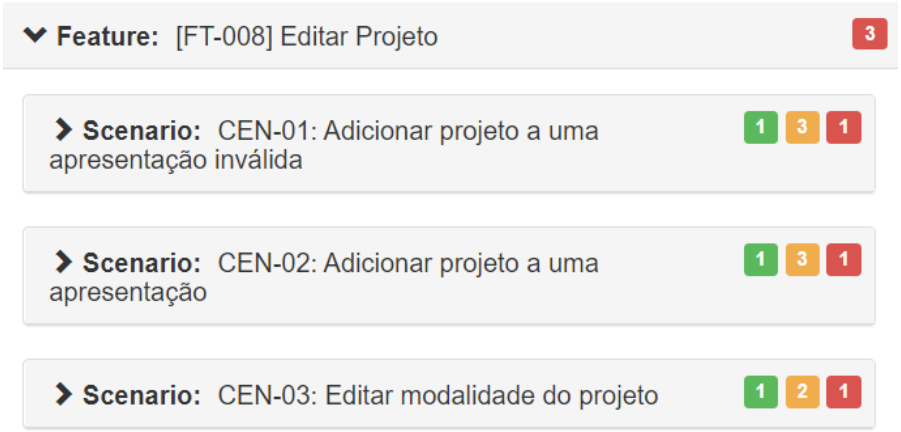


**Figura 15: Detalhamento do report dos cenários**
Fonte: Elaborado pelo autor.

Após identificar quais cenários falharam, o reporte era feito na ferramenta definida para este fim (Gitlab), e o critério de aceitação para que versão passasse para a branch master, ou seja, para a branch de produção, segundo Freitas (2020) eram:

- 95% dos cenários/casos de testes devem estar com status "passou".
- Nenhum bug de criticidade alta deve estar em aberto.

# 6 DIFICULDADES ENCONTRADAS

Algumas dificuldades foram encontradas no decorrer do desenvolvimento do trabalho, uma delas foi com relação ao conhecimento prévio sobre testes de software, como também sobre automação. Foi necessário estudos sobre ferramentas e frameworks que dão suporte ao processo de automação de teste tendo em vista que, esse conhecimento não é abordado e nem faz parte da grade curricular do curso de Ciência da Computação ofertado pela UFPB.

Foi necessário construir um grupo de estudo, onde o conhecimento da área de teste foi explorado e explanado periodicamente com o grupo envolvido. Houve ainda a dificuldade de conciliação com o tempo de disciplinas e tempo de execução do projeto, bem como dificuldade de conseguir pessoas interessadas em testes.

# 7 CONCLUSÕES E TRABALHOS FUTUROS

O objetivo deste trabalho foi propor e implementar um processo de automação de teste para um projeto executado em meio acadêmico. Esse processo foi apresentado no capítulo 5, em que se apresentou de forma exitosa o processo e seus resultados. Além disso, apresentamos um conjunto de ferramentas de testes que deram suporte e possibilitaram essa implementação.

Os resultados deste trabalho proporcionou ganhos de tempo no desenvolvimento e implantação da plataforma SisEnex adotada durante o ENEX de 2020. Os testes automatizados se mostraram eficientes para o contexto do SisEnex, ainda que tenha necessitado de um tempo maior destinado a escrita dos scripts e cenários de teste, tendo em vista que a equipe não contava com dedicação exclusiva para a execução do projeto.

Com este trabalho será possível estender os testes automatizados para a plataforma mobile, com isso, como trabalho futuro, o objetivo será proporcionar uma otimização do tempo e reduzir o esforço demandado também na aplicação móvel do SisEnex.

# REFERÊNCIAS


[1] BARTIÉ, Alexandre. **Garantia da qualidade de software**. Rio de Janeiro: Campus, 2002.

[2] SOMMERVILLE, Ian. **Engenharia de software**. São Paulo: Pearson Prentice Hall, 2011.

[3] PRESSMAN, Roger S. **Engenharia de Software: Uma abordagem profissional**. São Paulo: Pearson Makron Books, 2011.

[4] SMART, John F. (Foreword by Dan North) **BDD in Action: Behavior-Driven Development for the whole software lifecycle**. NY: Manning Publications, 2015.

[5] CHELIMSKKY, D. et al. **The RSpeec Book: Behaviour-Driven Development with RSpec,Cucumber, and Friends**. [S.l.: s.n.], 2010.

[6] AMODEO, Enrique. **Learning Behavior-driven Development with JavaScript**. Packt Publishing, 2015.

[7] SILVA, Paulo César Barreto da; ALVES, Thiago Salhab; BRUNO, Elisângela Andrade. AUTOMAÇÃO DE TESTES FUNCIONAIS. **Revista de Ciências Exatas e Tecnologia: Testes funcionais automatizados de software**, v. 6, 6, p. 113-133, abr, 2014.

[8] MAHAJAN, Prasad; SHEDGE, Harshal; PATKAR, Uday. Automation Testing In Software Organization. **International Journal Of Computer Applications Technology And Research**, Pune, v. 5, 4, p. 198-201, maio, 2016.

[9] BERNARDO, P. C. **Padrões de Testes Automatizados**. 2011. Dissertação (Mestrado em Engenharia Industrial) – Instituto de Matemática e Estatística, Universidade de São Paulo, 2011.

[10] FREITAS, Dayane Félix. Implantação de um Processo de Testes de Software em um Ambiente Acadêmico. 2020. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Universidade Federal da Paraíba, João Pessoa, 2020. [Orientador: Profa. Dr. Danielle Rousy Dias Ricarte]

[11] GALITEZI, Thiago. Conceito: Tipos de Testes. Araraquara: Organização, 2012. Disponível em:<http://www.galitezi.com.br/2012/02/conceito-tipos-de-testes.html>. Acesso em: 11 mar. 2021.

[12] NORTH, Dan. Introducing BDD. 2003. Disponível em:<https://dannorth.net/introducing-bdd/>. Acesso em: 12 mar. 2021.

[13] FERREIRA, Avelino. Testes automatizados: como garantir a qualidade do software. 2018. Disponível em:<https://k21.global/blog/qualidade-do-software-testes-automatizados>. Acesso em: 15 mar. 2021.

[14] COSTA, Nathalia. Ferramentas de automação tendências para 2021. 2020. Disponível em:<https://dbccompany.com.br/dbc/ferramentas-de-automacao-tendencias-para-2021/>. Acesso em: 18 mar. 2021.

[15] TRONG, Bui. Testes de Automação com Cucumber BDD em times Ágeis. 2018. Disponível em:<https://www.infoq.com/br/articles/cucumber-bdd-automation-testing/>. Acesso em: 18 mar. 2021.

[16] JOSÉ, Mateus. Descomplicando testes E2E com Cypress. 2020. Disponível em:<https://medium.com/totvsdevelopers/descomplicando-testes-e2e-com-cypress-ca4a8952e6e9>. Acesso em: 18 mar. 2021.

[17] SANTOS, Raphael. Testes end-to-end com Cypress.io. 2019. Disponível em:<https://blog.novatics.com.br/teste-end-to-end-com-cypress-io-358576dc05c3>. Acesso em: 18 mar. 2021.

[18] CUCUMBER, Cucumber. Disponível em:<¡https://cucumber.io/¿>. Acesso em: 18 mar. 2021.

[19] SILVA, Taise. 3 noções básicas essenciais para a criação de uma suíte de automação para aplicativos web. 2015. Disponível em:<https://www.thoughtworks.com/pt/insights/blog/3-essential-basics-setting-automation-suite-web-apps>. Acesso em: 18 mar. 2021.

[20] MESQUITA, Gabriel. TESTES DE INTEGRAÇÃO COM REST ASSURED. 2020. Disponível em:<https://blog.onedaytesting.com.br/testes-de-integracao-com-rest-assured/>. Acesso em: 9 abr. 2021.

[21] COMO ESCREVER UM BOM BDD?. 2020. Disponível em:<https://www.base2.com.br/2020/06/03/como-escerver-um-bom-bdd/>. Acesso em: 20 abr. 2021.

[22] HAMILTON, Thomas. Automation Testing Tutorial — What is Automated Testing?. 2021. Disponível em:< https://www.guru99.com/automation-testing.html>. Acesso em: 25 nov. 2021.

[23] MIRANDA, Diogo. Teste unitário e Qualidade de Software. 2017. Disponível em:< https://medium.com/assertqualityassurance/teste-unit

[24] BERNARDO, C. P.; KON, F. A Importância dos Testes Automatizados. 2008. Disponível em:< https://www.ime.usp.br/ kon/papers/EngSoftMagazine-IntroducaoTestes.pdf>. Acesso em: 26 nov. 2021.

[25] FREITAS, Dayane Felix. Processo de Testes. João Pessoa: Universidade Federal da Paraíba, 2020. Color.