UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO DE CIÊNCIAS EXATAS, NATURAIS E DA SAÚDE
DEPARTAMENTO DE COMPUTAÇÃO

PAULO VICTOR CANAL PANNO

TESTES FUNCIONAIS AUTOMATIZADOS EM APLICAÇÕES WEB COM BEHAVIOUR DRIVEN DEVELOPMENT

ALEGRE
ESPÍRITO SANTO
2023

PAULO VICTOR CANAL PANNO

TESTES FUNCIONAIS AUTOMATIZADOS EM APLICAÇÕES WEB COM BEHAVIOUR DRIVEN DEVELOPMENT

Trabalho de Conclusão de Curso apresentado ao Curso de Ciência da Computação do Centro de Ciências Exatas, Naturais e da Saúde da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do título de Cientista da Computação.

Orientador: Prof. Dsc. Clayton Vieira Fraga Filho

ALEGRE - ES
2023

PAULO VICTOR CANAL PANNO

TESTES FUNCIONAIS DE APLICAÇÕES WEB COM BEHAVIOUR DRIVEN DEVELOPMENT

Trabalho de Conclusão de Curso apresentado ao Departamento de Computação do Centro de Ciências Exatas, Naturais e da Saúde da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 19 de julho de 2023.

COMISSÃO EXAMINADORA

Prof. Dr. Clayton Vieira Fraga Filho
Universidade Federal do Espírito Santo
Orientador

Prof. Dr. Bruno Vilela Oliveira
Universidade Federal do Espírito Santo

Prof. Dra. Simone Dornelas Costa
Universidade Federal do Espírito Santo

AGRADECIMENTOS

Agradeço primeiramente a Deus, que me proporcionou sabedoria e resiliência para superar todas as adversidades.

Agradeço à Universidade Federal do Espírito Santo pela disponibilidade de materiais, bem como os livros da biblioteca para que o trabalho fosse realizado com sucesso.

Agradeço imensamente aos meus pais que me deram forças e que me incentivaram todos os dias. Não tenho palavras para descrever a importância de vocês! Amo vocês!

Agradeço ao meu irmão Pedro Henrique, que mesmo longe, sempre me apoiou nos momentos difíceis.

Aos meus tios e meus primos, Raphael Canal Maximino e Sarah Canal Maximino, por toda a ajuda e consideração, não tenho como agradecê-los.

Agradeço ao Professor Clayton Fraga Filho, meu orientador, pelos incontáveis ensinamentos e conselhos, com as conversas enriquecedoras. Não tenho como lhe agradecer por tudo!

Agradeço a todos os professores do Departamento de Ciência da Computação por me acolherem, e pelas inúmeras dúvidas respondidas sempre que necessário. Muito obrigado pela ajuda, levarei por toda minha caminhada os inúmeros conhecimentos passados!

Agradeço a minha namorada Ester da Fonseca Lucas, por todo o carinho e consideração, e por todos os momentos em que me ajudou quando precisei.

Um agradecimento à Paula Tavares, por todos os desabafos, que me aconselhava com conversas descontraídas e me colocavam “pra cima”. Muito Obrigado!

Agradeço também também a todos os membros da Click Jr., pelos conhecimentos passados e por despertar em mim a vontade de seguir o caminho de qualidade de software e pelo crescimento individual adquirido.

Agradeço às minhas amigas Cintia e Vitória, por todos os desabafos e brincadeiras, e demonstrando que a amizade não depende da distância. Muito obrigado!

Aos meus amigos que de alguma forma contribuíram com a minha formação. Principalmente a Isabela Amorim Ribeiro, por todo o suporte durante este fim de faculdade.

RESUMO

Este texto apresenta um estudo comparativo sobre a implementação de testes funcionais de aplicações web com a metodologia Behaviour Driven Development (BDD) e o framework Cypress. O objetivo foi melhorar a eficiência e qualidade dos processos de teste. A metodologia incluiu a configuração do ambiente, organização da estrutura do projeto, escrita dos testes automatizados e validação dos resultados. Foram definidas métricas para avaliar os resultados, como taxa de sucesso dos testes, cobertura de requisitos, tempo de resposta do sistema e taxa de defeitos encontrados. Os resultados mostraram que a alocação adequada de esforços nas etapas de planejamento e implementação dos testes de aceitação foi fundamental para obter resultados satisfatórios. A execução dos testes automatizados foi eficiente, economizando tempo e recursos. A análise dos resultados e a identificação das falhas foram essenciais para a melhoria contínua do software, garantindo a qualidade do software testado. Em resumo, a implementação de testes funcionais com BDD e o framework Cypress foi bem-sucedida na automação dos cenários de teste da aplicação web, proporcionando maior eficiência, confiabilidade e agilidade nos processos de teste. Recomenda-se a adoção dessa abordagem para melhorar a qualidade de aplicações web, investindo adequadamente nas diferentes etapas do processo de teste.

SUMÁRIO

1. INTRODUÇÃO 11
2. OBJETIVOS 14
2.1 Objetivo Geral 14
2.2 Objetivos específicos 14
3 REVISÃO BIBLIOGRÁFICA 15
3.1. Test-Driven Development 15
3.2. Behavior Driven Development 17
3.3. Teste de Software 22
3.4. Teste de aceitação 25
4. METODOLOGIA 28
4.1. Estudo do MESUR 28
4.2. Escolha da Empresa 30
4.3. Formatação BDD 31
4.4. Cypress 32
4.5. Identificação de Cenários de Teste 35
4.6. Implementação de testes automatizados com Cypress 35
4.7. Planejamento da execução e avaliação de teste 36
5. RESULTADOS 38
5.1. Resumo dos Casos de Uso do MESUR 38
5.2. Cenários BDD 39
Funcionalidade: Incluir Medição 39
5.3. Cenários de Teste Identificados 40
5.4. Testes Automatizados Implementados 44
5.5. Execução dos Testes e Avaliação dos Resultados 46
5.6. Análise Comparativa dos Resultados dos testes de aceitação automatizados 50
6. CONCLUSÃO 60
6.1. REFERÊNCIAS 68
8. APÊNDICES 71
8.1. APÊNDICE A - Cenários BDD do MESUR 71
Funcionalidade: Incluir Medição 71
Funcionalidade: Incluir Item na medição 72
Funcionalidade: Visualizar Medição 74
Funcionalidade: Apresentar dashboard 74
Funcionalidade: Exportar Medição 75
Funcionalidade: Imprimir Relatório 76
8.2. APÊNDICE B - Scripts do Cypress 77
Funcionalidade: Incluir Medição 79
Funcionalidade: Visualizar de Medição 85
Funcionalidade: Apresentar Dashboard 85
Funcionalidade: Exportar de Medição 87
Funcionalidade: Imprimir Relatório Agrupando Itens pela Situação de Aferição 88

9. ANEXOS 91
Anexo 1: Termo de autorização de imagem do MESUR 91

LISTA DE FIGURAS

Figura 1 – Passos do TDD (Borges, 2006) 16
Figura 2 – História de usuário no BDD (Ribeiro, 2019) 17
Figura 3 – Sintaxe de Gherkin no BDD 19
Figura 4 - Exemplo da programação de testes automatizados (autor) 21
Figura 5 - Expressa a diferença entre esses conceitos (NETO, 2007) 23
Figura 6 - Diagrama de caso de uso de todas as funções de um usuário que utiliza a versão gratuita do MESUR. 28
Figura 7 - Diagrama de caso de uso das funções de maior grau de acessibilidade dos usuários 29
Figura 8 - Comparativo feito pela empresa ToolsQA analisando os frameworks Cypress e Selenium 32

LISTA DE TABELAS

Tabela 1 - Retorno do investimento de testes informais, manuais e automatizados (BLACK, 2000, p3) 24
Tabela 2 - Tempo de execução de testes manual e automatizados 47
Tabela 3 - Registro de esforço das atividades Incluir Medição 50
Tabela 4 - Registro de esforço das atividades Incluir Item na medição 51
Tabela 5 - Registro de esforço das atividades Visualizar Medição 52
Tabela 6 - Registro de esforço das atividades “Exportar Medição” 54
Tabela 7 - Registro de esforço das atividades Apresentar dashboard 55
Tabela 8 - Registro de esforço das atividades Imprimir Relatório 56
Tabela 9 - Registro de esforço das atividades e do número de falhas encontradas 57
Tabela 10 - Comparativo nome da funcionalidade com o nome do script em cypress. 78

LISTA DE ANEXOS

Anexo 1: Termo de autorização de imagem do MESUR 91

1. INTRODUÇÃO

Um levantamento feito pelo Instituto Brasileiro de Geografia e Estatística (IBGE), em 2019, afirma que a Internet chega a oito em cada dez domicílios do Brasil. Ainda no ano de 2019, o IBGE constatou que cerca de 12,6 milhões de domicílios tiveram acesso à Internet (IBGE, 2021), isto é, houve uma crescente no uso da tecnologia em lares brasileiros. A pandemia decorrente do COVID-19 intensificou a quantidade de usuários utilizando serviços web, os dados divulgados pela Anatel, o número de acessos de banda larga fixa no Brasil cresceu 241% desde março de 2020 (EXAME, 2022).

De acordo com os dados divulgados pela plataforma ‘’Índico’’, que entrevistou 2,4 mil pessoas de todo o território nacional em 2021, constatou que 60% dos brasileiros trabalham de casa no momento, dos quais 40% usam aplicativos web como ferramenta de trabalho (EXAME, 2022). Paralelo a isso, a cada ano o mercado de desenvolvimento de Software cresce em todo o mundo. No Brasil, por exemplo, segundo uma pesquisa realizada pela empresa Associação Brasileira das Empresas de Software (ABES) em parceria com o International Data Corporation (IDC), o mercado nacional de software foi responsável por uma receita de US$ 6,3 bilhões, o que apresentou um crescimento de 14,9% na comparação anual, representando para o país uma participação de 1,1% no mercado mundial (ABES, 2017)

O elevado nível da taxa de crescimento, faz com que fatores como custos, prazos e até mesmo concorrências sejam incentivados e forcem as empresas de software a criarem mecanismos nos quais os atendimentos das demandas sejam viabilizados e a satisfação dos seus clientes seja atendida (ALMEIDA, 2017). Ou seja, o mundo corporativo atual está em constante mudança, isto é, independente da profissão que exerça, haverá altas mudanças em uma dada frequência.

No mundo corporativo, à medida que a utilização da internet e tecnologias como ferramentas de trabalho aumentam, surgem mudanças significativas que exigem a adoção de ferramentas e técnicas que facilitem e otimizem o trabalho. Nesse contexto, o Manifesto Ágil, iniciado em 2001, busca aprimorar as práticas de gestão e desenvolvimento de sistemas, com foco na satisfação do cliente e na funcionalidade do produto final (STOPA; RACHID, 2019).

Compreendendo a necessidade em garantir a qualidade em projetos de desenvolvimento de software, os testes assumem uma posição fundamental neste

contexto (PRESSMAN; MAXIM, 2018).

Inserido nesta lógica, o modelo estratégico de pirâmide de testes, ilustrado por Cohn (2010), apresenta em sua estrutura a composição dos diferentes tipos de testes. No ápice, estão dispostos os testes de aceitação, que, embora em menor quantidade, carregam a responsabilidade em validar a conformidade do software perante os requisitos determinados pelos stakeholders.

Em contrapartida, *Behavior Driven Development* (BDD), vertente do Test Driven Development (TDD), demonstra-se efetiva na realização de testes funcionais de aceitação, pois o mesmo concentra-se em descrever o comportamento do sistema em um formato compreensível para todas as partes envolvidas (ADZIC, 2011). Deste modo, a comunicação entre desenvolvedores, testadores e partes interessadas é facilitada, garantindo assim que o desenvolvimento esteja em consonância com as necessidades do negócio.

O TDD, prática derivada da metodologia ágil *Extreme Programming* (XP), oferece um contexto inicial para o surgimento do BDD (BECK; ANDRES, 2004). A efetividade do BDD se manifesta em sua capacidade de fornecer uma linguagem comum que aproxima o domínio técnico e o domínio de negócios. Além disso, BDD promove uma abordagem de desenvolvimento orientado a comportamento, permitindo a validação antecipada de funcionalidades e a identificação de possíveis incompatibilidades ou inconsistências no sistema.

No ambiente do desenvolvimento ágil, a aplicação do BDD se estabelece como uma prática valiosa, oferecendo feedback rápido, facilitando a comunicação entre os envolvidos e assegurando que o produto final esteja em conformidade com as expectativas do usuário.

Souza (2016) defende que muitas ferramentas de teste interagem com ferramentas baseadas no uso do BDD, oferecendo suporte para o ciclo de vida completo do teste, a exemplo disso, Bui (2018), sugere o *Selenium* e o *Cucumber* como aplicativos de automação que aceitam o BDD como desenvolvimento.

O BDD originou-se como resposta ao TDD, porém com o foco de tornar essas práticas mais efetivas e acessíveis para a equipe de desenvolvimento, utilizando ferramentas como sintaxe Gherkin, que é uma linguagem orientada a espaços, ela usa indentação para definir a estrutura, em vista disso, tem sido uma opção muito forte na automação, pois se trata de um formato que pode utilizar para padronizar, documentar, e até reciclar código e funcionalidades, a user stories, sprints, o maior isolamento da

aplicação e estimulando a participação do cliente (RODRIGUES; FARINA, 2021).

A técnica de BDD tem como prática incluir todas as partes do time no desenvolvimento do software para um conhecimento único e claro do que é requisitado pelo cliente, tendo em vista que, simplifica a comunicação usando cenários descritos pelo cliente ou analista, cada cenário é dividido em no mínimo três palavras-chave: *Given* que representa um contexto, *When* quando acontecer um evento e *Then* determinando o que é esperado (RODRIGUES; FARINA, 2021). Por conseguinte, propõe-se produzir um software acima das expectativas do cliente final, fazendo o uso de linguagem informal para descrever as funções das aplicações e os integrantes da equipe compreendam todas as interações do homem com o computador, passos lógicos e simples (NORTH, 2014).

Em suma, o avanço da tecnologia e a crescente utilização de serviços web têm impulsionado o mercado de desenvolvimento de software no Brasil e no mundo. Nesse contexto, o Manifesto Ágil surge como uma abordagem dinâmica e colaborativa para a gestão e desenvolvimento de sistemas, priorizando a satisfação do cliente e a entrega contínua de valor. Como parte desse processo, os testes de software desempenham um papel fundamental na verificação da conformidade do sistema e na detecção precoce de defeitos, contribuindo para a qualidade e eficiência do desenvolvimento. Nesse sentido, os testes automatizados baseados na metodologia ágil BDD têm se mostrado uma tendência promissora, permitindo a escrita de cenários em linguagem natural e facilitando a manutenção do código.

.

2. OBJETIVOS

2.1 Objetivo Geral

Implementar testes automatizados baseados na metodologia ágil BDD nos cenários de testes da aplicação web MESUR, a fim de realizar um estudo comparativo dos resultados obtidos antes e depois da implementação da metodologia. No anexo 1 encontra-se o termo de autorização de imagem da aplicação web MESUR.

2.2 Objetivos específicos

Os objetivos específicos para este trabalho são:

a) Selecionar as funcionalidades alvo de testes de aceitação;
b) Registrar o esforço das atividades de testes de aceitação realizados;
c) Identificar os casos de testes da aplicação web MESUR;
d) Utilização do framework Cypress na automatização dos cenários de testes;
e) Avaliar os resultados obtidos em (a) e (b).

3 REVISÃO BIBLIOGRÁFICA

3.1. Test-Driven Development

As falhas de software possuem um grande impacto em relação aos custos e tempo no desenvolvimento de um software (BORGES, 2006). Apesar de não ser possível resolver os defeitos em um sistema, é possível minimizar consideravelmente a quantidade de defeitos utilizando o TDD, permitindo a identificação e remoção das falhas de forma mais eficiente e com curto espaço de tempo (BORGES, 2006).

Segundo o estudo publicado em 2002 pelo National Institute of Standards and Technology (NIST) com a união do Departamento de Comércio dos Estados Unidos informa que os defeitos de software têm um prejuízo de cerca de 60 bilhões de dólares por ano (NIST, 2002). Visto que, é necessário o aprofundamento das técnicas orientadas a testes para um bom desempenho e resultado econômico (NIST, 2002).

O TDD é uma técnica derivada do método *Extreme Programming* (XP) (BECK, 2000) e do Manifesto Ágil (AGILE ALLIANCE, 2000), sua base são técnicas de desenvolvimento utilizadas por décadas (GELPERIN; HETZEL, 1987) (LARMAN; BASILI, 2003). Na prática a implementação de um sistema começa pelos casos de testes de um objeto, sendo assim, escrevendo os casos e depois implementando os objetos e métodos, faz com que surja a necessidade de outros métodos e objetos.

No TDD, os programadores usam os testes para guiar o projeto que está sendo desenvolvido, eles automatizam esses testes para serem executados constantemente, por meio dos resultados dos testes é possível definir o progresso do desenvolvimento (BORGES, 2006). Programadores criam continuamente decisões pequenas para aumentar a funcionalidade do software, e só passará para novas etapas do desenvolvimento quando todos os casos de testes estiverem com o resultado de sucesso (BORGES, 2006).

Conforme Beck (2002), o TDD pode ser visualizado como um conjunto de iterações, cada iteração envolve quatro passos listados a seguir e ilustrados na Figura 1.

- Escolher a área do projeto ou requisitos da tarefa para gerir melhor o desenvolvimento.
- Projetar um teste concreto tão simples quando é requerida a orientação do desenvolvimento. Verificar se o teste falha.
- Modificar o sistema para corresponder aos testes e outros possíveis testes.

- Refinar o sistema para remover redundância, sem quebrar nenhum teste.

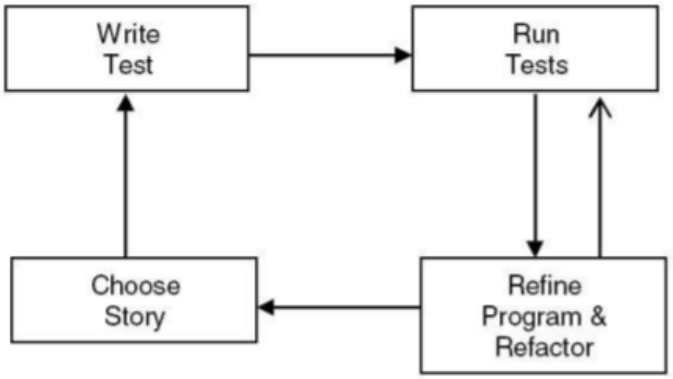

Figura 1 – Passos do TDD (Borges, 2006)

Uma importante regra no TDD é: “Se você não pode escrever um teste para o que está prestes a codificar, então você nem deveria estar pensando em codificar” (CHAPLIN, 2001). Outra regra no TDD diz que quando um defeito de software é encontrado, casos de teste de unidade são adicionados ao pacote de teste antes de corrigir o código, reduzindo o acoplamento dos componentes do software, uma vez que, torna-se possível gerar decisões de projeto em cada estágio do desenvolvimento (BORGES, 2006).

Outro estudo realizado por George e William (2004) em 2004, verificou a qualidade do software, o estudo foi feito com 6 equipes, e o resultado foi que na prática o TDD obteve um código com maior qualidade quando equiparado com a prática de método tradicional, além disso, os programadores que usaram o TDD tiveram um desempenho de 16% mais rápido do que os outros competidores que não utilizaram.

O TDD aplica as técnicas do XP chamada de *refactoring* para alcançar o gerenciamento da complexibilidade e compreensão do código (BECK, 2000). Com grandes sistemas a modificação é frequente, ele resulta em um sistema muito complexo e, portanto, é necessário a facilitação da manutenção (LEHMAN NELADY, 1985). O ciclo *test-then-code* resulta em um feedback frequente, falhas são identificadas mais cedo no momento que o novo código é adicionado, sendo assim, o tempo de depuração diminui suprido pela escrita e execução dos casos de teste (BORGES, 2006). De acordo com o estudo de Corbi (1989) em 1989, cerca de 50% das tarefas no processo de manutenção de software envolvia o entendimento do

código.

A técnica do TDD auxilia no entendimento de programas, posto que, os casos de testes esclarecem melhor o funcionamento do sistema, porém esta técnica não atinge a totalidade do sistema, para entender a sua totalidade é inevitável o uso de abstrações (BORGES, 2006). Os resultados são métodos ou funções testados automaticamente, esses valores serão comparados com os resultados esperados ainda na etapa de codificação, entretanto, na etapa de manutenção, unidades de teste serão criadas permitindo avaliar de forma mais fácil os novos defeitos que podem ser apresentados no software, sendo essencial para o controle de novas releases e na etapa de desenvolvimento, isto é, minimizando as chances de novas falhas no sistema.

3.2. Behavior Driven Development

BDD é uma resposta ao TDD, já que, os testes são escritos anteriormente à codificação, é uma técnica de desenvolvimento ágil que possui o foco de integrar regras de negócios com a linguagem de programação, focando no comportamento do software (BULGARELI; FOSCHINI, 2015).

De uma perspectiva de desenvolvimento de software ágil, o BDD é escrito por meio de histórias e cenários de usuários, organizadas conforme o fluxo normal, alternativas e exceções das histórias do usuário. As histórias de usuário são realizadas por meio de linguagem natural de diversas formas, uma de suas utilidades é entender a necessidade do cliente, itens de planejamento ou a descrição de um produto, como pode ser observado na Figura 2 (ARAUJO, 2017).

```
Funcionalidade: Logar no Outlook 360
      Eu como usuário
      Quero fazer login na minha conta do Outlook 360
      Para ter acesso ao meu correio eletrônico

Cenário: Logar com Sucesso
      Dado que estou na página de login do Outlook 360
      Quando me autêntico com um usuário válido
      Então sou direcionado para minha caixa de entrada do Outlook 360
```

Figura 2 - História de usuário no BDD (Ribeiro, 2019)

Para desenvolver uma história de usuário é preciso seguir o conceito INVEST, isto é, Independente, Negociável, Valiosa, Estimável, Small (Pequena) e Testável:

- Independente significa que uma boa história de usuário não depende de outras e que outras não dependem dela.
- Negociável significa que toda boa história é uma co-criação, uma negociação entre profissionais de negócio e profissionais técnicos.
- Valiosa quer dizer que toda história deve agregar valor ao produto e principalmente, ao usuário-chave da mesma.
- Estimável é importante para o planejamento do projeto e para o gerenciamento das expectativas dos envolvidos.
- Small (Pequena) para ser possível entregá-la em no máximo uma sprint, caso contrário ela será um épico.
- Testável indica que toda história deve poder ser coberta de testes para validar se seus requisitos funcionais e não-funcionais foram atendidos, geralmente abordados nos critérios de aceitação da história.

Sendo assim, a história de usuário necessita ser independente, negociável, valiosa, estimável, pequena e testável, possuindo como foco melhorar a comunicação com o cliente final que não possui conhecimento na área de TI (CORRÊA, 2018).

A história de usuário do BDD busca especificações executáveis por meio de testes escritos, mantendo uma linguagem ubíqua entre equipe de desenvolvimento e *stakeholders,* de modo a auxiliar na especificação do teste (NORTH, 2014).

Apesar de ser desenvolvida direcionada para o entendimento das funcionalidades do sistema, as histórias de usuário possuem alguns aspectos na sua implementação, como por exemplo, a dificuldade de entender a linguagem natural, causado pela ambiguidade e o escopo da história de usuário não está explícito (RAHARJANA; HARRIS; JUSTITIA, 2020).

De acordo com Cohn (2004) e North (2014), é possível superar o problema de ambiguidade e o escopo da história de usuário não explícito com o detalhamento dos cenários de testes, este modelo de detalhamento conforme mostra a Figura 3.

Funcionalidade: <Título de uma linha descrevendo a funcionalidade>

<Narrativa>
Para <Benefício>
Como um <Papel>
Eu <Recurso/Funcionalidade>

<Critérios - apresentados como Cenários>

Cenário: <Título do Cenário 1>
Dado <Parâmetro de entrada 1>
E <Parâmetro de entrada 2>...
E <Parâmetro de entrada n>
Quando <Evento que acontece em um determinado momento>
Então <Valor de saída 1>
E <Valor de saída 2>...
E <Valor de saída n>

...

Cenário: <Título do Cenário n>
Dado <Parâmetro de entrada 1>
E <Parâmetro de entrada 2>...
E <Parâmetro de entrada n>
Quando <Evento que acontece em um determinado momento>
Então <Valor de saída 1>
E <Valor de saída 2>...
E <Valor de saída n>

Figura 3 – Sintaxe de Gherkin no BDD

O *Behavior Driven Development* (BDD) utiliza uma linguagem específica, denominada Gherkin, para a escrita de cenários de testes de forma estruturada e compreensível para todas as partes envolvidas em um projeto. A sintaxe do Gherkin é projetada para ser facilmente legível e para descrever de maneira clara o comportamento do sistema (WYNNE; HELLESDOTTIR, 2012).

A estrutura básica do Gherkin é constituída pelos seguintes elementos: Feature (Funcionalidade), Scenario (Cenário), Given (Dado), When (Quando), Then (Então), And (E) e But (Mas). Cada um destes elementos possui um papel fundamental na descrição do comportamento do sistema, criando um conjunto de cenários que são facilmente compreendidos e executados tanto por pessoas quanto por ferramentas automatizadas (SMART, 2014).

Feature (Funcionalidade) serve para descrever a funcionalidade a ser testada. Scenario (Cenário) descreve o contexto para o teste, ou seja, o que está sendo

testado. Given (Dado) é utilizado para descrever o estado inicial do sistema antes do teste. When (Quando) define a ação que é executada no teste. Then (Então) é utilizado para verificar se o comportamento do sistema está de acordo com o esperado após a ação ser executada. And (E) e But (Mas) são usados para adicionar mais informações aos passos Given, When e Then (WYNNE; HELLESDOTTIR, 2012).

Dessa forma, a linguagem Gherkin, com sua sintaxe intuitiva e descritiva, serve como uma ponte entre os aspectos técnicos do desenvolvimento e os requisitos de negócio, permitindo que todos os envolvidos no projeto compreendam e participem da definição e validação do comportamento do sistema (SMART, 2014).

Gherkin é uma linguagem que permite a descrição de cenários de teste em um formato de alto nível e facilmente legível. Sua sintaxe tem por objetivo descrever os requisitos de negócio e os casos de teste de uma maneira que seja compreensível tanto para os membros técnicos quanto para os não técnicos da equipe do projeto (WYNNE; HELLESDOTTIR, 2012).

A descrição de um cenário válido em Gherkin segue a seguinte estrutura:

```
Funcionalidade: Título da funcionalidade

Cenário: Título do cenário
  Dado que uma condição prévia é verdadeira
  Quando uma ação é executada
  Então um resultado é esperado
```

Por exemplo:

```
Funcionalidade: Login na aplicação

Cenário: Login bem-sucedido
  Dado que o usuário está na página de login
  Quando o usuário insere um nome de usuário e senha válidos
  Então o usuário é redirecionado para a página inicial
```

Neste exemplo, todas as partes do cenário estão presentes e ele segue a estrutura básica do Gherkin. Isso torna o cenário válido (SMART, 2014).

Por outro lado, um cenário inválido em Gherkin poderia ser algo como:

```
Funcionalidade: Login na aplicação
```

```
Quando o usuário insere um nome de usuário e senha válidos
```

Neste caso, o cenário é inválido porque falta a definição do cenário ("Cenário:"), o estado inicial do sistema ("Dado que...") é a verificação do resultado ("Então..."). A sintaxe do Gherkin é projetada para facilitar a compreensão dos cenários de teste e, se partes dela estão ausentes, a descrição do teste não será clara ou completa, tornando o cenário inválido (WYNNE; HELLESDOTTIR, 2012).

Posteriormente à criação dos cenários é possível programar os testes automatizados utilizando um compilador, como ilustra a Figura 4. Em sucessão, a execução dos testes será feita através do Cypress.

```
describe('Suite #701 - Incluir Usuário (CADASTRO)', function () {
      beforeEach(() => {

      //Faz o login preservando os dados de sessão
      cy.loginAdm()
      //acesssar menu usuário cadastro
      cy.AcessaMenu('usuarioCADASTRO')
      //acessar a tela de inclusão
      cy.get('#USU__incluir').click()
      })

      it('Usuário (Cadastro): Incluir - Não preencher campos obrigatórios',
function () {
      //Não preencher campo nome e email
      cy.get(':nth-child(1) > #USU_email').click()
      cy.get('#USU_nome').click()
      cy.get('.card').click()
      //Verificar se o botão salvar está desabilitado
      cy.get('#USU__salvar').should('be.disabled')
      //Verifica se apresenta a mensagem do toast
      cy.get('#USU_email_sem').should('exist')
      cy.get('#USU_nome_sem').should('exist')
      })

      it('Usuário (Cadastro): Incluir - Com Sucesso', function () {
      //Incluir usuário
```

```
      cy.get(':nth-child(1) >
#USU_email').type('usuarioautomatizado@hotmail.com.com.br')
      cy.get('#USU_nome').type('Usuário automatizado')
      cy.get('#USU__salvar').click()
      //validar a messagem de sucesso
      cy.get('.ng-tns-c105-59.ng-trigger-toastAnimation > .ng-trigger >
.p-toast-message-content').should('have.text', 'Dados salvos com sucesso!')
      //Verificar a existência do registro no grid
      cy.get('.col-12').should('contain',
'visualizadorautomatizado@cadastro.com.br')
})
```

Figura 4 - Exemplo da programação de testes automatizados (autor)

O primeiro teste ("Usuário (Cadastro): Incluir - Não preencher campos obrigatórios") verifica se o sistema valida corretamente quando os campos obrigatórios (nome e e-mail) não são preenchidos, conferindo se o botão de salvar está desabilitado e se as mensagens de erro são exibidas corretamente. O segundo teste ("Usuário (Cadastro): Incluir - Com Sucesso") simula a inclusão de um usuário preenchendo os campos corretamente, clicando no botão de salvar e, em seguida, verifica se a mensagem de sucesso é exibida e se o novo usuário aparece no grid do sistema. Antes de cada um dos testes, o código realiza uma sequência de ações, incluindo o login como administrador, acessando o menu de cadastro de usuário e, finalmente, chegando à tela de inclusão.

Os resultados dos cenários são recursos de como o sistema deve interagir para cada evento que ocorra em uma determinada circunstância, ou seja, os cenários são ações que alteram a saída do sistema. (PATRIKAKIS, PAPADOPOULOS, KOGIAS, 2017). Ademais, a estrutura é composta por um título, um dado que é uma pré-condição, por quando é a condição para o usuário executar uma ação, e por último a resposta do sistema (RAHARJANA; HARRIS; JUSTITIA, 2020).

3.3. Teste de Software

A premissa para o entendimento dos testes de softwares é compreender as diferenças entre defeitos, erros e falhas. A Institute of Electrical and Electronics Engineers (IEEE, 1990), profere as seguintes definições:

- Um erro, também conhecido como "mistake" no contexto do desenvolvimento de software, ocorre quando um desenvolvedor realiza uma ação que não está conforme o esperado ou viola os procedimentos corretos de codificação, estes erros podem surgir de equívocos relacionados aos requisitos do software ou de erros humanos durante a implementação (IEEE, 1990).
- Por sua vez, um defeito, também referido como "bug" ou "fault", é uma parte incorreta do código que é resultado de um erro cometido anteriormente, tratando-se de um estado anômalo presente no software, capaz de gerar um comportamento indesejado (BEIZER, 1990). É importante ressaltar que um defeito pode existir no sistema sem causar problemas, a menos que sejam criadas determinadas condições que ativam esse defeito.
- Já a falha, denominada como "failure", ocorre quando um defeito é manifestado durante a execução do software. Isso acontece quando o software não funciona conforme o esperado, devido à ativação de um defeito previamente existente (KANER et al., 1999).

Dessa forma, é possível entender que um erro é a origem inicial do problema, o defeito é a consequência desse erro na programação e a falha é o resultado final, visível através da manifestação do defeito durante a execução do software.

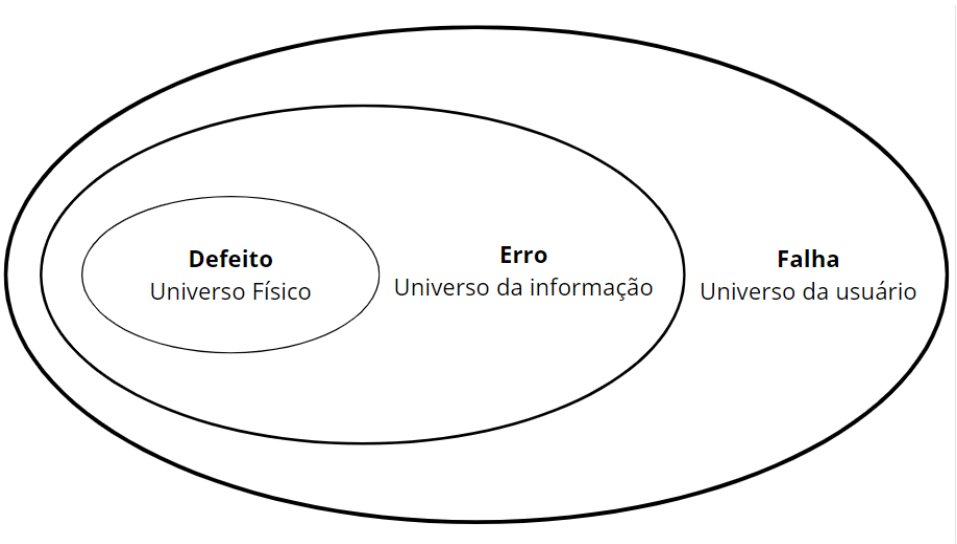

Figura 5 - Expressa a diferença entre esses conceitos (NETO, 2007)

Ao testar um software, o objetivo é agregar valor, confiabilidade através da identificação e remoção de defeitos e elevar a qualidade do produto, contudo, não se deve testar um software com o foco de mostrar que ele é funcional, mas sim, para

mostrar que o software contém defeitos e que os testes para serem bem sucedidos tem como a função encontrar o maior número de defeitos possíveis (MYERS, 2004).

Para desenvolver testes de software é preciso a elaboração de três etapas, são elas: o planejamento, o projeto e a implementação do teste (ROCHA et al., 2001). Arruda (2006), defende que o planejamento do teste é a estruturação de alguns critérios, como, por exemplo, quando e quais os testes devem ser executados e o responsável por eles. Arruda (2006) sugere também que o projeto de teste define as etapas necessárias para interagir com a função do sistema, a pré e a pós-condição do teste, e por último a implementação do teste no qual concebe os resultados esperado e a sequência de passos para que os testadores aferem se a funcionalidade está sendo executada corretamente.

Ao comparar os três tipos de testes, é relevante analisar a pesquisa de Rex Black (2000) a fim de verificar o retorno investido a partir das atividades de testes. Ao analisar a Tabela 1, é notório que a utilização de testes automatizados garante um retorno de investimento de 445%.

Tabela 1 - Retorno do investimento de testes informais, manuais e automatizados (BLACK, 2000, p3)

		Teste Informal	**Teste Manual**	**Teste Automatizado**
Investimento	Pessoal	0	60.000	60.000
	Infraestrutura	0	10.000	10.000
	Ferramentas	0	0	12.500
	TOTAL	0	70.000	82.500
Fase de desenvolvimento	Defeitos encontrados	250	250	250
	Custos	2.500	2.500	2.500

	encontrados			
Fase de testes	Defeitos encontrados	0	350	500
	Custos para corrigir	0	35.00	50.000
Produção	Defeitos encontrados	750	400	250
	Custos para corrigir	750.000	400.000	250.000
Custo da Qualidade	Conformidade (investimento em prevenção)	0	70.000	82.500
	Não conformidade (custo para corrigir os defeitos encontrados)	752.500	437.500	302.500
	TOTAL	752.500	507.500	385.500
Retorno de Investimento		**N/A**	**350%**	**445%**

3.4. Teste de aceitação

Os testes de aceitação são uma etapa crucial no ciclo de desenvolvimento de software, ocorrendo antes da implantação do sistema no ambiente do usuário final (ISTQB, 2018).

Conforme mencionado por Sommerville (2007), o teste de aceitação representa a etapa final do processo de teste, onde o sistema é avaliado antes de ser considerado pronto para uso operacional. Esses testes são geralmente conduzidos por um grupo restrito de usuários finais do sistema, que executam operações típicas para verificar se o software se comporta conforme as especificações definidas nas histórias de usuário (ASSIS, 2012). Além dos usuários finais, a equipe de desenvolvimento e outros stakeholders do projeto, como o cliente ou um representante do usuário final, também podem realizar os testes de aceitação (FEWSTER e GRAHAM, 1999).

O sistema é testado em um ambiente simulado semelhante ao real, utilizando dados fornecidos pelo cliente em vez de dados fictícios. Os testes de aceitação têm como base os requisitos do usuário e visam testar a funcionalidade do software por meio de sua interface gráfica, concentrando-se principalmente nas regras de negócios e na experiência do usuário (KANER et al., 1993).

Por meio desse processo, garante-se que o software atenda aos requisitos do usuário final e às expectativas de desempenho no mundo real. No entanto, durante os testes, a equipe de projeto muitas vezes encontra situações que não foram previstas na análise de falhas realizada durante a etapa de definição dos critérios de aceitação, prototipagem funcional ou no wireframe do projeto (ASSIS, 2012).

Portanto, os testes de aceitação desempenham um papel vital na garantia de que o software esteja pronto para ser implantado e utilizado no mundo real. Eles podem ser automatizados ou realizados manualmente, dependendo das necessidades e recursos do projeto (KANER et al., 1993).

Os testes de aceitação automatizados são uma parte integral do processo de desenvolvimento de um sistema web, visando verificar se o sistema atende às expectativas do usuário e aos requisitos estabelecidos. Este procedimento pode ser dividido em várias etapas (ASSIS, 2012).

Inicialmente, a definição dos critérios de aceitação é primordial. Os critérios de aceitação são declarações detalhadas que descrevem as funcionalidades que o sistema deve ter para satisfazer o usuário final ou os requisitos do cliente (ASTELS, 2006). Tais critérios devem ser estabelecidos em colaboração com todas as partes interessadas, incluindo desenvolvedores, gerentes de produto, analistas de negócios e usuários finais.

A segunda etapa envolve a escrita de casos de teste de aceitação, que detalham as ações e os resultados esperados no sistema para verificar se os critérios de aceitação são atendidos. A elaboração de casos de teste deve seguir a abordagem de cenário, representando situações do mundo real nas quais o sistema será utilizado (PUGH, 2011).

Após a definição dos casos de teste, a etapa subsequente é a automação desses testes. A automação de testes de aceitação tem o potencial de economizar tempo e esforço, além de aumentar a consistência e a precisão dos testes (FEWSTER; GRAHAM, 1999). O processo de automação geralmente envolve a seleção de uma ferramenta de automação apropriada, como o Selenium ou o Cypress,

e a programação dos casos de teste utilizando a linguagem e a sintaxe compatíveis com a ferramenta escolhida.

A execução dos testes de aceitação automatizados é a próxima etapa. Isso envolve a execução dos testes automatizados no ambiente de teste apropriado e a coleta de resultados dos testes.

Finalmente, a etapa de revisão e análise dos resultados dos testes é crucial. Isso inclui a análise das falhas dos testes, a determinação das causas raiz e a tomada de decisões com base nos resultados dos testes (KANER; NGUYEN; FALK, 1998).

Tipicamente as atividades envolvidas nos testes de aceitação de software são vitais para garantir que o sistema atenda às necessidades do usuário final. A seguinte discussão oferece uma análise detalhada dessas atividades, destacando a importância de registrar o esforço investido em cada uma delas.

1. Planejamento de Testes: Esta etapa envolve a identificação de critérios de teste, definição de casos de teste, além da elaboração de um plano de testes. É aqui que os testadores definem os parâmetros de sucesso para o teste de aceitação. O registro do esforço investido nesta fase ajuda a entender a quantidade de trabalho necessária para preparar um teste de aceitação eficaz (PERRY, 2006).
2. Implementação dos Testes de Aceitação: Esta atividade envolve a execução do plano de teste. Aqui, os testadores traduzem os casos de teste definidos na fase de planejamento em scripts de teste. Registrar o tempo gasto nesta atividade pode fornecer percepções sobre a eficiência da implementação do teste e pode ajudar a identificar possíveis áreas de melhoria (FEWSTER & GRAHAM, 1999).
3. Execução dos Testes: Esta fase é onde os testes são realmente executados. O esforço investido nesta etapa pode variar dependendo da complexidade do software e do número de casos de teste implementados. Registrar este esforço pode fornecer informações sobre a eficiência do processo de teste (JONES, 2008).
4. Análise dos Resultados: Depois que os testes são executados, os resultados devem ser analisados para determinar se o software cumpriu os critérios de teste. O registro do esforço investido nesta etapa pode ajudar a avaliar o tempo necessário para analisar os resultados e entender melhor o desempenho do software (KANER, BACH, & PETTICHORD, 2008).
5. Identificação das Falhas: Esta atividade envolve a identificação e documentação

de quaisquer falhas encontradas durante a execução dos testes. Registrar o esforço gasto nesta etapa pode fornecer percepções sobre a qualidade do software (MYERS, SANDLER, & BADGETT, 2011).

6. Busca pelos Defeitos: Esta etapa envolve a identificação das causas das falhas encontradas. O registro do esforço gasto pode ajudar a entender quão desafiador é identificar a raiz dos problemas no software (ZHANG, ZHOU, & XU, 2008).
7. Correção dos Defeitos: Esta atividade envolve a correção dos defeitos identificados. Registrar o esforço gasto nesta etapa pode fornecer informações sobre a eficiência da equipe de desenvolvimento na correção de defeitos (JORGENSEN, 2003).
8. Reportar a quantidade de falhas encontradas: Este é um indicador importante da qualidade do software. Acompanhar esta métrica ao longo do tempo pode oferecer percepções sobre a eficácia das práticas de desenvolvimento e teste (JONES, 2008).

4. METODOLOGIA

A metodologia proposta consiste em utilizar a aplicação MESUR para automatizar o fluxo de informações na medição da produção de software, utilizando a técnica BDD para escrever cenários e definir os requisitos funcionais e comportamentais. Em seguida, os casos de uso do MESUR são convertidos para o formato BDD, identificando os atores e escrevendo cenários em Gherkin. Os cenários são refinados e, posteriormente, os testes são automatizados utilizando o framework Cypress. Os cenários de teste específicos do MESUR são identificados e mapeados, e os passos de teste são especificados. Por fim, os casos de teste vão ser refinados e os resultados dos testes serão validados. A implementação dos testes automatizados com Cypress envolve a configuração do ambiente, a estruturação do projeto, a escrita dos testes e a validação dos resultados. Os resultados dos testes são coletados, analisados e utilizados para avaliar se os objetivos foram alcançados.

4.1. Estudo do MESUR

O sistema web MESUR, desenvolvido e mantido pela empresa Fatto na sua versão gratuita, foi utilizado como objeto de estudo para esta pesquisa. A premissa principal desta aplicação é prover suporte a todo o fluxo de informações inerente à medição da produção de software, bem como a automatização de cálculos utilizando métodos amplamente reconhecidos e estabelecidos como padrões de direito e de facto. Este sistema constitui um componente integrante da Plataforma de Gestão de Software disponibilizada pela FATTO, trabalhando em sinergia com os serviços oferecidos pelo Escritório de Métricas e o Centro de Orçamento desenvolvida pela FATTO.

Para ser possível a automação dos cenários é preciso utilizar os cenários escritos na técnica BDD do aplicativo web MESUR. O sistema é extenso e possui uma complexidade para a execução da automação, como pode ser visto na Figura 6.

.

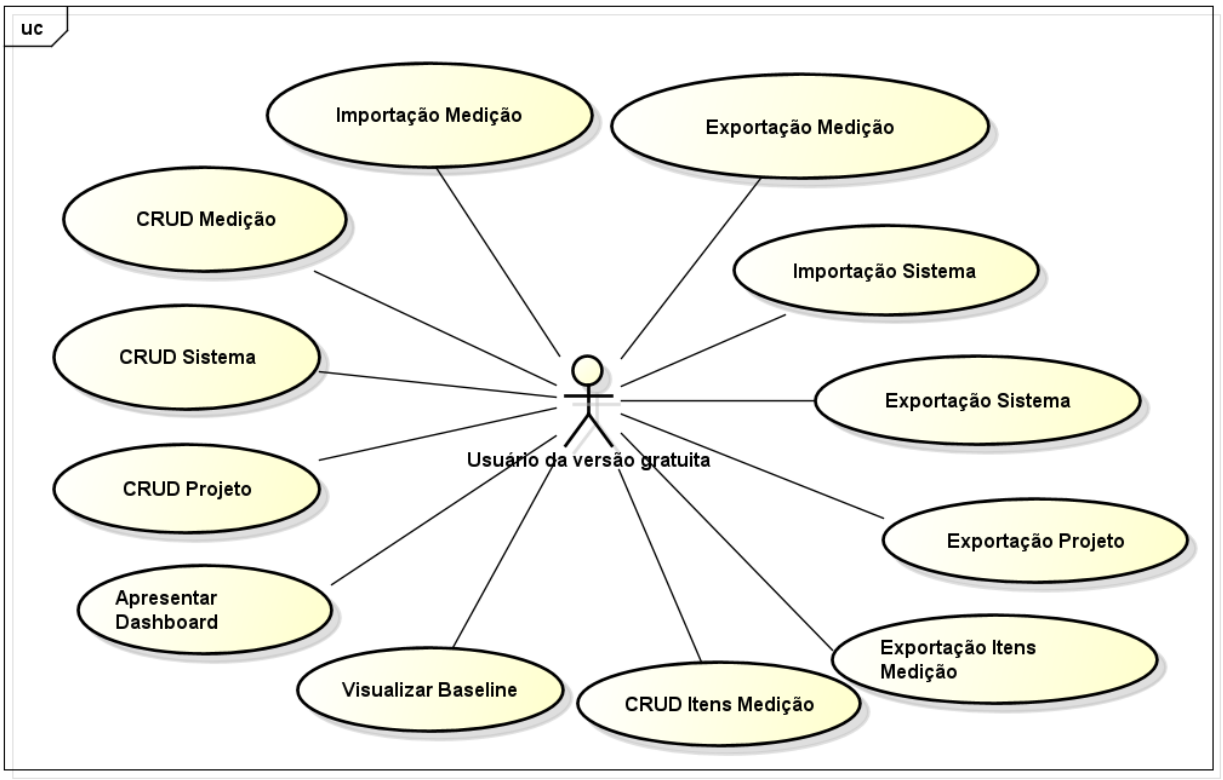

Figura 6 - Diagrama de caso de uso de todas as funções de um usuário que utiliza a versão gratuita do MESUR (Autor)

A escolha dos casos de uso para o MESUR gratuito foi baseada em um cenário comum de uso, considerando as necessidades e expectativas típicas dos usuários ao criar um projeto de métricas. O primeiro caso de uso consiste na importação medição, essa funcionalidade permite a importação de dados de medição de alguma fonte externa para o sistema MESUR. Pode ser útil para incorporar dados de medição de outras ferramentas ou formatos para análise ou acompanhamento dentro do sistema.

O segundo caso de uso se refere à exportação medição, essa funcionalidade permite exportar dados de medição armazenados no sistema MESUR para um formato externo. Isso possibilita compartilhar os resultados ou integrar os dados em outros sistemas ou ferramentas.

O terceiro caso de uso refere-se a funcionalidade CRUD Medição: CRUD é um acrônimo para *Create* (Criação), *Read* (Leitura), *Update* (Atualização) e *Delete* (Exclusão). Essa funcionalidade permite realizar todas essas operações com relação aos dados de medição dentro do sistema MESUR. Ou seja, permite criar novas medições, ler informações existentes, atualizar medições e excluir dados quando necessário.

O quarto caso de uso é a funcionalidade CRUD Sistema, semelhante ao CRUD

de Medição, o CRUD Sistema permite criar, ler, atualizar e excluir informações relacionadas aos sistemas ou subsistemas que estão sendo medidos ou avaliados pelo MESUR.

O quinto caso de uso se refere a funcionalidade CRUD Projeto, essa funcionalidade permite gerenciar informações sobre projetos dentro do sistema MESUR. Isso pode incluir a criação de novos projetos, a atribuição de equipes responsáveis, o registro de detalhes do projeto e a atualização ou exclusão de projetos quando necessário.

O sexto caso de uso refere-se a funcionalidade Apresentar Dashboard, o painel de controle ou dashboard é uma interface visual que apresenta informações importantes resumidas e em tempo real do sistema MESUR. Isso pode incluir métricas-chave, gráficos, indicadores de desempenho e outras informações relevantes para ajudar os usuários a terem uma visão geral rápida e eficaz do status dos sistemas ou projetos monitorados.

O sétimo caso se refere a funcionalidade Visualizar Baseline, a funcionalidade de visualização de baseline permite que os usuários vejam uma referência estática ou uma linha de base dos dados de medição. Isso é útil para comparar o desempenho atual com uma situação anteriormente estabelecida como padrão ou objetivo.

O oitavo caso refere-se a funcionalidade CRUD Itens Medição, assim como o CRUD Medição, o CRUD Itens Medição permite criar, ler, atualizar e excluir informações específicas sobre os itens de medição ou componentes que estão sendo avaliados dentro do sistema MESUR.

O nono caso consiste na Exportação Itens Mediçãom, essa funcionalidade possibilita a exportação de dados ou informações específicas sobre os itens de medição para um formato externo, permitindo a análise detalhada fora do sistema MESUR.

Por fim, o último caso de uso Exportação Projeto e Exportação Sistema: São funcionalidades semelhantes às outras exportações mencionadas, mas direcionadas para os dados relacionados a projetos e sistemas, respectivamente. Essas funcionalidades adicionais proporcionam maior flexibilidade e acessibilidade aos usuários, ampliando as possibilidades de utilização do MESUR gratuito em diferentes contextos.

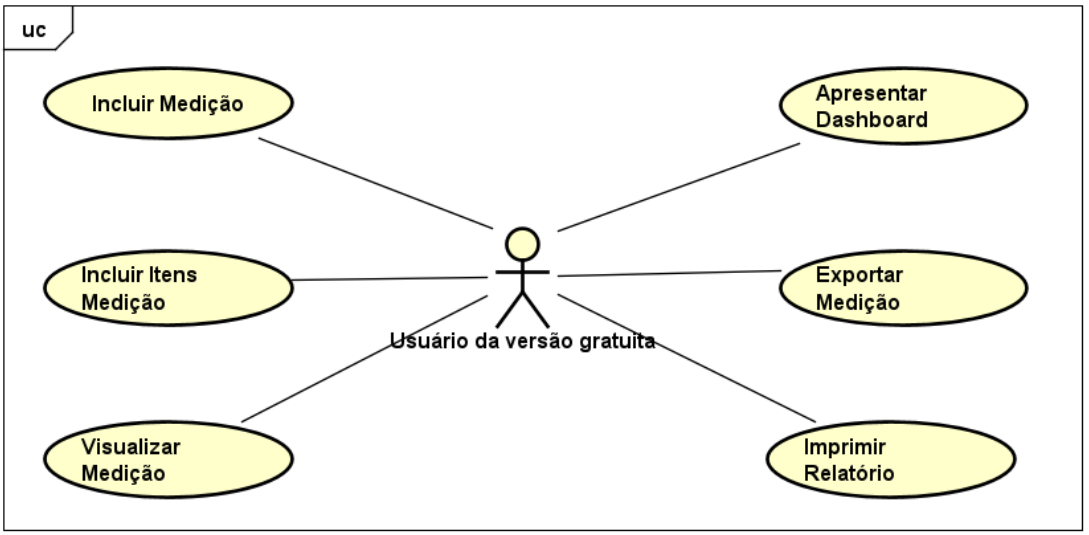

Figura 7 - Diagrama de caso de uso das funções de maior grau de acessibilidade dos usuários (Autor)

O autor acessou a tela de auditoria do sistema para verificar as telas mais utilizadas pelos usuários. Utilizando o Diagrama de Caso de Uso, ele identificou as funcionalidades essenciais e prioritárias, priorizando ações de maior grau de acessibilidade para garantir uma interação mais inclusiva e eficiente com o sistema.

4.2. Escolha da Empresa

A Fatto, uma organização estabelecida em Vitória, Espírito Santo, atua há 24 anos no setor de soluções para gestão de software. A empresa se dedica principalmente à otimização de resultados e ao aperfeiçoamento do relacionamento entre clientes e desenvolvedores. Dentre suas atividades-chave, incluem-se a correção de cenários redigidos na técnica BDD, o desenvolvimento de métricas, a contagem de pontos de função e a definição e implementação de soluções em Gestão de Software. Para a condução desta pesquisa, a Fatto gentilmente concedeu acesso à aplicação MESUR, que foi empregada na criação e automação de cenários, além da disponibilização de imagem do referido software.

4.3. Formatação BDD

O Desenvolvimento Orientado a Comportamento (BDD - *Behavior-Driven Development*) configura-se como uma metodologia que privilegia a colaboração entre os diversos atores envolvidos no processo de desenvolvimento de software,

englobando desenvolvedores, analistas de negócios e especialistas do domínio. O BDD destaca-se por buscar um entendimento unificado dos requisitos do software, utilizando para tal exemplos concretos que se centram no comportamento (NORTH, 2014). No âmbito da transição dos casos de uso do MESUR para o formato BDD, o procedimento compreende os seguintes passos:

1. Identificação e análise dos casos de uso do MESUR: Esta fase envolve o estudo dos casos de uso existentes no MESUR, buscando familiaridade com os requisitos e a lógica de negócios descritos nesses casos de uso.
2. Extração dos requisitos funcionais: Este passo consiste na identificação dos requisitos funcionais cruciais de cada caso de uso. Tais requisitos descrevem o comportamento que se espera do sistema, abordando aspectos como entradas, saídas e interações com o usuário (IEEE, 1998).
3. Identificação dos atores: Nesta etapa, são identificados os atores pertinentes a cada caso de uso. Os atores configuram-se como entidades externas que interagem com o sistema.
4. Escrita de cenários no formato Gherkin: Com base nos requisitos funcionais e nos atores identificados, os cenários são redigidos no formato Gherkin, uma linguagem de especificação utilizada no BDD, conhecida por sua sintaxe simples e legível. Cada cenário ilustra um exemplo específico do comportamento esperado do sistema em uma determinada situação.
5. Especificação dos passos dos cenários: Para cada cenário, são especificados os passos que delineiam as interações entre o sistema e os atores. Esses passos são redigidos utilizando palavras-chave específicas do Gherkin, tais como "Dado" (Given), "Quando" (When) e "Então" (Then), que auxiliam na descrição do estado inicial, da ação realizada e do resultado esperado (CHEROSKY, 2012).
6. Refinamento dos cenários: Nesta fase, os cenários redigidos são revistos e refinados, se necessário, para assegurar que eles abordam adequadamente os requisitos funcionais e os diversos casos de teste pertinentes.
7. Automação dos testes BDD: Após a definição e o refinamento dos cenários, os testes BDD podem ser automatizados utilizando a ferramenta Cypress. Esta ferramenta possibilita a execução dos cenários redigidos em formato Gherkin como testes automatizados, garantindo que o software desenvolvido atenda aos

requisitos especificados (HILLENBRAND, 2020).

Cabe ressaltar que a transição dos casos de uso do MESUR para o formato BDD não se resume a uma mera questão de formatação, mas sim a adoção de uma abordagem orientada ao comportamento, que coloca ênfase na compreensão e descrição clara do comportamento do sistema através de exemplos concretos.

4.4. Cypress

A seleção apropriada das ferramentas de automação de teste é um fator crítico para garantir o sucesso de um projeto de software. Entre as várias opções disponíveis, duas das mais amplamente utilizadas são o Cypress e o Selenium. No entanto, o Cypress se destaca devido a uma série de características técnicas que o tornam uma opção atrativa.

Primeiramente, a arquitetura do Cypress é projetada para superar as limitações do Selenium. Enquanto o Selenium opera executando comandos remotamente por meio do protocolo JSON Wire, o Cypress executa no mesmo ciclo de execução que a aplicação (CYPRESS, 2020). Essa arquitetura permite que o Cypress tenha controle direto e em tempo real sobre a aplicação e o navegador, resultando em testes mais confiáveis e consistentes.

Além disso, o Cypress é uma solução de teste *end-to-end* (E2E) que abrange testes de interface do usuário (UI), testes de unidade e testes de *Application Programming Interface* (API) em uma única ferramenta. Isso difere do Selenium, que geralmente requer a integração com outras ferramentas para atingir o mesmo nível de cobertura de teste. A abordagem "tudo-em-um" do Cypress simplifica o processo de configuração dos testes e aumenta a eficiência no desenvolvimento (MOBARAYA; ALI, 2019).

Outra característica técnica importante do Cypress é o seu suporte nativo para JavaScript, uma linguagem amplamente adotada no desenvolvimento de aplicações web modernas. Isso facilita a integração dos testes com o fluxo de trabalho de desenvolvimento e elimina a necessidade de aprender uma linguagem de script de teste separada, como é exigido em alguns outros ambientes de teste (MOBARAYA; ALI, 2019).

Por fim, o Cypress oferece recursos avançados de depuração. Ele tira

automaticamente capturas de tela em cada etapa do teste e grava vídeos de todo o processo. Essa funcionalidade proporciona uma visão detalhada do que ocorreu em cada ponto do teste, auxiliando na identificação e correção de problemas.

	Selenium	Cypress
Speed	😐	😊
Wait for Element	😐	😊 😊
Remote Execution	😊 😊 😊	😡
Parallel Execution	😊	😡
Headless	😊	😊
Screenshot	😐	😐
Video	😡	😊
Documentation	😐	😊
Community	😊 😊 😊	😐
Execute JS	😊	😊 😊 😊
Switch Tabs	😊	😡
Several Browsers	😊	😊
Load Extensions	😊	😊
Manage Cookies	😊	😊

Figura 8 - Comparativo feito pela empresa ToolsQA analisando os frameworks Cypress e Selenium

O Cypress e o Selenium são duas ferramentas populares para automação de testes de software, com algumas diferenças distintas. Em termos de velocidade, o Cypress é conhecido por sua execução rápida, enquanto o Selenium pode ser mais lento devido à sua arquitetura baseada em Selenium Grid. Em relação ao "Wait for Element" (espera por elemento), o Cypress possui um mecanismo integrado de espera inteligente, que aguarda automaticamente até que um elemento esteja disponível para interagir. No Selenium, é necessário escrever explicitamente comandos de espera para aguardar a disponibilidade do elemento.

Quanto à execução remota, o Cypress suporta apenas execução em um ambiente local, enquanto o Selenium pode ser configurado para executar testes remotamente em uma grade distribuída. Em termos de execução paralela, o Cypress possui suporte integrado para executar testes em paralelo em vários navegadores, enquanto o Selenium requer configuração adicional para alcançar a execução paralela.

O Cypress possui um modo *Headless* (sem cabeça) embutido, o que significa que os testes podem ser executados em segundo plano sem abrir uma janela do

navegador. O Selenium também oferece suporte ao modo *Headless* através de navegadores específicos. Em relação a capturar *screenshots* e vídeos durante a execução dos testes, o Cypress possui recursos integrados para isso, enquanto no Selenium é necessário usar bibliotecas externas para capturar *screenshots* e gravar vídeos.

Quanto à documentação e comunidade, tanto o Cypress quanto o Selenium têm uma extensa documentação e comunidades ativas, com recursos e suporte disponíveis. O Cypress permite executar código JavaScript diretamente nos testes, o que facilita a interação com a página, enquanto o Selenium requer o uso de métodos específicos para executar código JavaScript. Em relação a trocar de abas durante a execução dos testes, o Cypress oferece suporte nativo para isso, enquanto no Selenium é necessário usar manipulações mais complexas para alternar entre abas. O Cypress suporta vários navegadores de forma nativa, enquanto no Selenium é necessário configurar drivers específicos para cada navegador.

Devido às suas características e pontos fortes mencionados acima, o Cypress foi escolhido como a ferramenta de automação de testes para o presente trabalho. A velocidade de execução do Cypress, juntamente com seu mecanismo inteligente de espera por elemento, proporcionará uma eficiência maior durante a automação dos testes. Além disso, a capacidade de executar testes em paralelo em vários navegadores e o suporte nativo para modo *Headless* oferecerão flexibilidade e agilidade para o processo de teste.

A possibilidade de capturar screenshots e vídeos durante a execução dos testes, juntamente com a documentação abrangente e a comunidade ativa, garantem um suporte sólido e recursos adicionais para melhorar a qualidade dos testes. Além de que, a capacidade de executar código JavaScript diretamente nos testes e a facilidade de alternar entre abas são vantagens importantes para o presente trabalho.

Embora o Cypress não ofereça suporte nativo para carregar extensões de navegador e gerenciar cookies, esses requisitos não são prioritários para o projeto atual. Considerando todos esses aspectos, a escolha do Cypress como ferramenta de automação de testes é adequada e atenderá às necessidades específicas do trabalho, maximizando a eficiência, a confiabilidade e a qualidade dos testes automatizados.

4.5. Identificação de Cenários de Teste

1 A identificação de cenários de teste específicos do MESUR, originários dos casos de uso no formato BDD, pressupõe um processo de análise e transposição dos requisitos e comportamentos descritos nos cenários para cenários de teste mais detalhados (CUCUMBER, 2018). Tal processo contempla as seguintes fases:
2 Revisão dos cenários BDD: Este é o estágio inicial onde se dá a análise dos cenários BDD elaborados para cada caso de uso, procurando compreender cada um dos requisitos funcionais e comportamentais descritos nesses cenários (SMART, 2014).
3 Identificação dos casos de teste: Nesta fase, ocorre a análise de cada cenário BDD, com o intuito de identificar as diversas condições e comportamentos que necessitam de teste. Este processo pode englobar diferentes combinações de entradas, fluxos de trabalho, exceções e resultados esperados.
4 Mapeamento para casos de teste MESUR: Esta etapa envolve a tradução do comportamento descrito em cada cenário BDD para os passos e ações necessários nos casos de teste do MESUR, com base na análise realizada dos cenários BDD.
5 Especificação dos passos de teste MESUR: Para cada caso de teste identificado, são especificados os passos e as ações que devem ser realizadas para executar o teste. Esta etapa pode englobar a configuração do ambiente de teste, a preparação de dados, a execução das ações requeridas no sistema e a verificação dos resultados esperados (ADZIC, 2011).
6 Refinamento dos casos de teste: Neste último processo, os casos de teste do MESUR são revisados e refinados para garantir que eles abrangem de maneira adequada todas as condições e comportamentos relevantes identificados nos cenários BDD. Isso se concretiza garantindo que os casos de teste sejam claros, específicos e abranjam todas as funcionalidades e requisitos cruciais do sistema.

4.6. Implementação de testes automatizados com Cypress

A implementação dos testes para os cenários de teste, realizada com o uso do framework Cypress, compreendeu diversas etapas (DEITEL; DEITEL, 2020).

Inicialmente, houve a configuração do ambiente, etapa na qual foi

estabelecido o ambiente necessário para a execução dos testes automatizados com o Cypress. Esta fase implicou na instalação do Node.js e do próprio Cypress, além da estruturação adequada do projeto (MANNING; BODILY, 2018).

Na segunda etapa, foi organizada a estrutura do projeto de teste, de modo a estabelecer uma organização lógica. Para isso, foram criadas pastas para o armazenamento dos arquivos de teste, arquivos de suporte e quaisquer outros recursos necessários. Tal organização contribui para manter o projeto ordenado e escalável à medida que mais testes são incluídos.

A terceira etapa compreendeu a escrita dos testes, momento em que foram elaborados os testes automatizados para os cenários de teste previamente identificados.

Por último, na quarta etapa, realizou-se a validação de resultados. Após a execução das ações no aplicativo, foi essencial verificar se os resultados estavam conforme as expectativas estabelecidas nos cenários de teste. Para isso, empregou-se o comando cy.should, a fim de verificar se os elementos apresentados continham os valores esperados, se as mensagens de erro eram exibidas corretamente, dentre outras verificações (AHMAD, 2019).

4.7. Planejamento da execução e avaliação de teste

Na primeira fase daquele estudo, foram definidas métricas específicas para a avaliação dos resultados dos testes que foram executados por meio do Cypress. As seguintes métricas foram escolhidas: taxa de sucesso dos testes, cobertura de requisitos, tempo de resposta do sistema e taxa de defeitos encontrados. A taxa de sucesso dos testes avaliou a porcentagem de testes executados com sucesso em relação ao total de testes realizados. A cobertura de requisitos mediu o quanto os requisitos do sistema estavam sendo abrangidos pelos testes. O tempo de resposta do sistema avaliou o tempo necessário para o sistema responder a determinadas ações durante os testes. E a taxa de defeitos encontrados avaliou a quantidade de defeitos identificados durante os testes em relação ao total de testes executados.

Durante a coleta e análise dos resultados dos testes, foram utilizados relatórios de teste, registros de defeitos, capturas de tela e outras formas de documentação. Os relatórios de teste forneceram uma descrição detalhada dos testes realizados, os resultados obtidos e as métricas calculadas. Os registros de defeitos

foram utilizados para registrar detalhes de cada defeito encontrado, facilitando a identificação precisa dos problemas. Foram realizadas capturas de tela sempre que necessário, para documentar visualmente os resultados dos testes, incluindo erros ou comportamentos inesperados. Além disso, foram utilizados logs do sistema e outras formas de documentação relevantes para acompanhar e analisar os resultados dos testes.

5. RESULTADOS

5.1. Resumo dos Casos de Uso do MESUR

Os casos de uso do MESUR abrangem diferentes funcionalidades que permitem aos usuários interagir com a plataforma para realizar diversas tarefas. Abaixo está um resumo dos casos de uso selecionados da aplicação web MESUR que será utilizado neste projeto:

1. Inclusão de uma medição: Neste caso de uso, os usuários podem adicionar novas medições na plataforma MESUR. Isso envolve fornecer informações relevantes, como o tipo de medição, a data e hora da medição, a localização e os dados específicos da medição em si.
2. Inclusão de itens na medição: Os usuários têm a capacidade de adicionar itens específicos à medição. Esses itens podem ser componentes, dispositivos ou qualquer outra unidade de medida relacionada à medição em questão. Os usuários podem fornecer detalhes sobre cada item, como seu nome, quantidade, características e outros atributos relevantes.
3. Visualização dos dados na tabela(medição e itens): Nesse caso de uso, os usuários podem visualizar os dados de medição e os itens associados a eles em um formato de tabela. Isso permite uma visualização tabular dos dados, facilitando a análise e comparação de diferentes medições e seus respectivos itens.
4. Apresentação do *dashboard*: O MESUR oferece um painel de controle (*dashboard*) que apresenta uma visão geral dos indicadores de eficiência. Esse caso de uso permite aos usuários acessar o *dashboard* e visualizar gráficos, gráficos de tendência, métricas.
5. Exportação da medição: Por fim, os usuários podem exportar os dados de uma medição específica para uso externo. Isso pode ser útil para análises mais aprofundadas, geração de relatórios personalizados ou compartilhamento de dados com outras partes interessadas. A exportação pode ser feita em diferentes formatos, como CSV, Excel ou PDF, dependendo das necessidades do usuário.

5.2. Cenários BDD

A seguir, apresentamos os cenários *Behavior-Driven Development* (BDD) para os principais casos de uso do MESUR, incluindo a inclusão de medição, inclusão de itens, visualização dos dados na tabela, apresentação do *dashboard* e exportação de medição.

Funcionalidade: Incluir Medição

Como gestor do sistema MESUR

Quero incluir uma medição

Para que eu possa ter o controle de ponto de função da minha empresa

Cenário: Criar uma nova medição com sucesso

Dado que estou na página de criação de medição

Quando eu preencho o formulário com as seguintes informações:

| Nome = "Medição TCC"

| Sistema = "Sistema do TCC"

| Tipo de Contagem = "Projeto de Melhoria"

| Nível de Detalhamento = "Detalhada"

E eu clico no botão "Salvar"

Então eu vejo a mensagem "Medição salva com sucesso"

Cenário: Falha criar uma medição sem preencher os campos obrigatórios

Dado que estou na página de criação de medição

Quando eu preencho o formulário com as seguintes informações:

| Nome = " "

| Sistema = "   "

| Tipo de Contagem = "   "

| Nível de Detalhamento = "   "

Então eu vejo uma “dica de ferramenta” nos campos obrigatórios com a mensagem "Campo Obrigatório" e a borda dos campos vermelha.

Além dos cenários apresentados anteriormente, os demais cenários relacionados às funcionalidades do MESUR estão descritos de forma mais detalhada no apêndice A. Essa abordagem permite uma melhor organização e separação das informações, mantendo o foco principal no texto principal.

O apêndice A contém cenários adicionais que abordam diferentes casos de uso do MESUR, como a inclusão de itens, visualização dos dados no grid, apresentação do *dashboard* e exportação de medição. Cada cenário é descrito seguindo o mesmo padrão de escrita utilizado nos exemplos apresentados anteriormente, fornecendo um contexto, a ação realizada e a expectativa do resultado.

Essa estrutura facilita a compreensão e o desenvolvimento das funcionalidades do MESUR, uma vez que os cenários descritos no apêndice oferecem uma visão abrangente e detalhada das interações esperadas com o sistema em diferentes situações de uso.

Dessa forma, o texto principal apresenta uma visão geral das funcionalidades e cenários mais relevantes, enquanto o apêndice contém informações complementares e mais detalhadas, proporcionando um recurso adicional para consulta e entendimento completo do comportamento do MESUR.

5.3. Cenários de Teste Identificados

Os cenários de testes identificados a seguir foram desenvolvidos com base nos cenários *Behavior-Driven Development* (BDD) apresentados para os principais casos de uso do MESUR: Inclusão de Medição, Inclusão de Item na Medição, Visualização

de Medição, Apresentação do *Dashboard* e Exportação de Medição. O objetivo desses cenários de teste é garantir que a plataforma MESUR funcione conforme o esperado, cumprindo suas funcionalidades de forma precisa e eficiente.

1. **Teste de Inclusão de Medição:**

a) Cenário: Criar uma nova medição com sucesso
   - Verificar se é possível criar uma medição fornecendo informações válidas nos campos obrigatórios.
   - Verificar se a mensagem "Medição salva com sucesso" é exibida após a criação da medição.

b) Cenário: Falha ao criar uma medição sem preencher os campos obrigatórios
   - Verificar se o sistema exibe uma “dica de ferramenta” nos campos obrigatórios não preenchidos.
   - Verificar se as bordas dos campos obrigatórios ficam vermelhas após a tentativa de salvar.

c) Cenário: Falha ao criar uma medição com data inválida
   - Verificar se o sistema exibe uma “dica de ferramenta” e marca o campo de data em vermelho quando uma data inválida é fornecida.

d) Cenário: Cancelar a operação ao criar uma medição
   - Verificar se é possível cancelar a criação de uma medição e se o usuário é redirecionado para a página de listagem de medições.

2. **Teste de Inclusão de Item na Medição:**

a) Cenário: Criar um novo item com sucesso
   - Verificar se é possível criar um novo item fornecendo informações válidas nos campos obrigatórios.
   - Verificar se a mensagem "Item salvo com sucesso" é exibida após a criação do item.

b) Cenário: Falha ao criar um novo item sem preencher os campos obrigatórios
   - Verificar se o sistema exibe uma “dica de ferramenta” nos campos obrigatórios não preenchidos.
   - Verificar se as bordas dos campos obrigatórios ficam vermelhas após a tentativa de salvar.

c) Cenário: Falha ao criar um novo item devido ao nome já existente
   - Verificar se o sistema exibe a mensagem de erro "O item com o nome 'Item tcc' já existe na medição!" quando um item com o mesmo nome já existe na medição.

d) Cenário: Cancelar a operação ao criar um item
   - Verificar se é possível cancelar a criação de um item e se o usuário é redirecionado para a página de listagem de itens.

3. Teste de Visualização de Medição:

a) Cenário: Visualizar Medição
   - Verificar se a tabela exibe as medições cadastradas com as colunas Tipo de Contagem, Nível de Detalhe, Ponto de Função Bruto e Ponto de Função Líquido.
   - Verificar se as informações são exibidas corretamente no formato tabular.

b) Cenário: Falha ao visualizar tabela vazia
   - Verificar se o sistema exibe a mensagem "Nenhuma medição registrada" quando não há medições cadastradas.

c) Cenário: Filtrar medições por critérios específicos
   - Verificar se é possível filtrar as medições com base em critérios como data, tipo de contagem ou nível de detalhe.
   - Verificar se a tabela exibe apenas as medições que correspondem aos critérios de filtragem selecionados.

d) Cenário: Ordenar medições por colunas

- Verificar se é possível ordenar as medições de acordo com as colunas da tabela, como Tipo de Contagem, Nível de Detalhe, Ponto de Função Bruto ou Ponto de Função Líquido.
- Verificar se as medições são exibidas em ordem crescente ou decrescente, conforme a opção selecionada.

4. Teste de Apresentação do *Dashboard*:

a) Cenário: Exibir informações do *dashboard* corretamente
- Verificar se o *dashboard* exibe as principais informações, como o número total de medições, o número de medições concluídas, o número de medições em andamento, etc.
- Verificar se os gráficos e estatísticas são atualizados corretamente com base nos dados das medições.

b) Cenário: Verificar a interatividade do *dashboard*
- Verificar se é possível interagir com os gráficos do *dashboard*, como clicar em um ponto para exibir detalhes ou filtrar as informações exibidas.
- Verificar se as informações exibidas no *dashboard* são consistentes com as informações detalhadas das medições.

5. Teste de Exportação de Medição:

a) Cenário: Exportar medição para formato Valores Separados por Vírgulas (CSV)
- Verificar se é possível exportar uma medição para um arquivo CSV.
- Verificar se o arquivo CSV contém todas as informações relevantes da medição, como Tipo de Contagem, Nível de Detalhe, PF Bruto, PF Líquido, etc.

b) Cenário: Exportar medição para formato *Portable Document Format* (PDF)
- Verificar se é possível exportar uma medição para um arquivo PDF.
- Verificar se o arquivo PDF contém todas as informações relevantes da medição, formatadas de forma adequada.

c) Cenário: Verificar a integridade dos dados exportados

- Verificar se os dados exportados para CSV ou PDF correspondem exatamente às informações exibidas na plataforma MESUR.
- Verificar se não há perda de dados durante o processo de exportação.

6. Teste de Imprimir Relatório

a) Cenário: Imprimir relatório agrupando os itens pela situação da aferição
   - Verificar se os dados exportados estão agrupados pela situação da aferição.
   - Verificar se não há perda de dados durante o processo de exportação.
   - Verificar se é possível exportar um relatório de aferição para um arquivo PDF.

b) Cenário: Imprimir relatório agrupando itens pela situação da aferição
   - Verificar se os dados exportados não estão sendo agrupados pela situação da aferição.
   - Verificar se não há perda de dados durante o processo de exportação.
   - Verificar se é possível exportar um relatório de aferição para um arquivo PDF.

c) Cenário: Cancelar a operação de exportar um relatório
   - Verificar se é possível cancelar a exportação de um relatório e se o usuário continua na página de edição da medição .

5.4. Testes Automatizados Implementados

Neste tópico, exploramos os testes automatizados implementados com o Cypress e como essa ferramenta pode ser usada para melhorar a eficiência e a confiabilidade dos testes. Os detalhes adicionais sobre a implementação dos testes, juntamente com os códigos correspondentes, podem ser encontrados no apêndice B. Lá, você encontrará informações mais específicas sobre as configurações dos testes, casos de uso abordados e outras considerações relevantes para o entendimento completo da abordagem adotada. O apêndice B servirá como uma valiosa fonte de

referência para aprofundar seus conhecimentos sobre essa importante etapa do projeto.

1. Cenário de Teste: Inclusão de Item na Medição

Descrição do caso de uso: O usuário inclui um novo item na medição selecionada, fornecendo informações como nome do item, tipo de função e valor associado.

```
    it('Item Medição: Incluir - Com Sucesso ', function () {

        //selecionar a medição
           cy.get('.ui-grid-row:nth-child(1)  .ui-grid-cell:nth-child(1)  >
.ui-grid-cell-contents').click({ multiple: true })

        //clicar no botão itens
        cy.get('#MED__itens').click()
        cy.wait(2000)

        //clicar no botão Incluir Itens
        cy.get('#MED__incluir_item').click()
        cy.wait(2000)

        //Preencher o nome
        cy.get('#nomeItem').type('Item Automatizado{enter}')

        //Preencher o Tipo de Função
        cy.get('#tipo_funcao_id').select('INM')

        //Preencher INM
        cy.get('#inm_id').select('CA - Componente Arquivo')

        //Clicar no botão salvar
        cy.get('#MED__salvar_item').click()

        //Verificar a existencia do registro no grid
                     cy.get('.ui-grid-viewport').should('contain',   "Item
Automatizado");
    })
```

Explicação: O teste acima verifica se uma mensagem de sucesso é exibida corretamente ao tentar inserir um item em uma medição com valores válidos.

2. Cenário de Teste: Exportar medição via tabela - Todos os campos

```
it('Medição: Exportar - Todos os Campos', () => {

    //clicar para exportar
    cy.get('.p-speeddial-button').click()

    //exportar dados visíveis
    cy.get('#menuitem-8').click()

    //ler o arquivo e validar

cy.readFile('cypress/downloads/Medições.xlsx').should('exist').and('conta
ins', 'Título da Medição', 'Aferição',
        'Tipo de Contagem', 'Nível de Detalhe', 'Anexos', 'Sistema',
'Projeto', 'Tipo de projeto', 'Iteração', 'Status',
        'Liberar baseline', 'Situação', 'Responsável', 'Aferidor', 'PF
Bruto', 'PF Líquido', 'PF Fab Met', 'Itens', 'Reúso',
     'Esforço (h)', 'Custo ($)', 'UST', 'Total Medição de Terceiros', 'OS
Interna', 'OS Externa', 'Plataforma',
      'Área de negócio', 'Data Refer',     'Data Fechamento',   'Data da
última aferição',  'Marcadores', 'Link público',
    'Criado Em', 'Atualizado Em', 'Criado Por', 'Atualizado Por')
})
```

Explicação: O teste acima verifica se o arquivo exportado não possui perda de dados e também se está no formato csv.

5.5. Execução dos Testes e Avaliação dos Resultados

Apresentamos abaixo os resultados da execução dos testes automatizados, que foram realizados visando garantir a qualidade e a robustez do software. Esses testes foram projetados para validar o comportamento e a integridade do sistema em relação

aos requisitos estabelecidos.

A Tabela 2 a seguir oferece uma visão geral dos testes executados, incluindo uma comparação entre os testes realizados manualmente e os automatizados.

Essas métricas fornecem uma compreensão clara do desempenho do software durante o processo de teste automatizado. Ao analisar esses resultados, podemos identificar áreas que requerem atenção adicional, priorizar correções e otimizações, e fornecer informações valiosas para a equipe de desenvolvimento.

Tabela 2 - Tempo de execução de testes manual e automatizados

Teste	**Tempo de Execução Manual (em segundos)**	**Tempo de Execução Automatizado (em segundos)**	**Variação Percentual: Tempo de Execução**
Criar uma nova medição - Com Sucesso	85 segundos	67 segundos	27%
Falha criar uma medição sem preencher os campos obrigatórios	48 segundos	40 segundos	21%
Falha ao criar uma medição com data inválida	56 segundos	46 segundos	22%
Cancelar a operação ao criar uma medição	60 segundos	53 segundos	14%
Criar um novo item com sucesso	95 segundos	77 segundos	24%
Falha criar um novo item sem preencher os campos obrigatórios	70 segundos	55 segundos	27%
Falha criar um novo item devido ao nome já	67 segundos	60 segundos	11%

existente			
Cancelar a operação ao criar um item	96 segundos	74 segundos	30%
Visualizar Medição - Com Sucesso	38 segundos	32 segundos	17%
Falha ao Visualizar - Com Grid Vazio	12 segundos	10 segundos	12%
Apresentar dashboard - Com Sucesso	53 segundos	47 segundos	12%
Falha ao apresentar o dashboard sem medições cadastradas	39 segundos	31 segundos	26%
Exportar medição individual	19 segundos	16 segundos	16%
Exportar medição via grid – todos os dados como Excel	28 segundos	25 segundos	11%
Exportar medição via grid – dados visíveis como Excel	23 segundos	19 segundos	19%
Exportar medição individual via grid com botão direito	19 segundos	16 segundos	17%
Imprimir relatório sem agrupar os itens pela situação da aferição	42 segundos	33 segundos	27%
Imprimir relatório agrupando itens pela situação da aferição	26 segundos	22 segundos	18%
Cancelar a operação de exportar um relatório	21 segundos	17 segundos	24%

Total:			**321%**

A Tabela 2 apresenta uma representação dos testes automatizados, em que cada linha corresponde a um teste específico e cada coluna representa uma métrica específica a ser avaliada:

1. A coluna "Teste" lista os nomes ou identificadores dos testes executados.
2. A coluna “Tempo de Execução Manual”, registra o tempo necessário para a execução manual de cada teste.
3. A coluna "Tempo de Execução Automatizado", registra o tempo necessário para a execução automatizada de cada teste.
4. A coluna “Variação Percentual: Tempo de Execução”, registra a diferença percentual entre o tempo gasto em testes automatizados e manuais, revelando a eficiência e a rapidez alcançadas ao automatizar processos de testes.

O teste de aceitação, seja ele manual ou automatizado, é um componente essencial no desenvolvimento de software, garantindo que a solução atende às expectativas dos usuários finais (MOLINA et al., 2017). A Tabela 2 apresentada revela uma economia substancial de tempo na execução de testes automatizados em comparação com os manuais. Essa constatação é corroborada por vários estudos que mostram que a automação de testes pode melhorar a eficiência ao reduzir o tempo de execução do teste (FEWSTER e GRAHAM, 1999).

A variação percentual de tempo de execução observada varia de 11% a 30%. Isso indica que o teste automatizado é consistentemente mais rápido do que o teste manual em todos os cenários, refletindo um aumento de eficiência. No entanto, é importante notar que o tempo economizado na execução do teste é apenas uma das várias métricas a serem consideradas ao avaliar a eficácia do teste automatizado. Outras métricas, como a precisão do teste e o número de defeitos encontrados, também são vitais (DUSTIN et al., 2009).

Comparando com outros estudos, a média da variação percentual dos tempos de execução entre os testes manuais e automatizados no presente estudo é condizente com os resultados reportados por Datta (2012). Este autor observou que a automação de testes pode resultar em uma redução de até 20% no tempo de execução de testes em comparação com o teste manual. No entanto, essa economia de tempo pode variar dependendo da complexidade dos casos de teste e da eficácia

da ferramenta de automação utilizada (PETTICHORD, 2002).

Em conclusão, os dados fornecidos suportam a ideia de que o teste automatizado pode melhorar significativamente a eficiência do processo de teste em termos de tempo de execução. Isso ressalta o valor do teste automatizado como uma abordagem eficaz para otimizar o processo de desenvolvimento de software, embora outras métricas além do tempo de execução do teste também devam ser consideradas para obter uma avaliação completa da eficácia do teste.

5.6. Análise Comparativa dos Resultados dos testes de aceitação automatizados

A Tabela 3 apresenta o registro de atividades relacionadas à funcionalidade "Incluir Medição"

Tabela 3 - Registro de esforço das atividades Incluir Medição

Atividade	Data de início	Hora de Início	Data de Fim	Hora de Fim	Esforço	Percentual do Esforço total
Planejamento de Testes	01/06/2023	16:00	01/06/2023	17:00	01:00	28,44%
Implementação dos Testes Aceitação	02/06/2023	17:00	02/06/2023	18:00	01:00	28,44%
Execução dos Testes	03/06/2023	18:30	03/06/2023	18:35	00:05	2,37%
Análise dos Resultados	01/07/2023	13:00	01/07/2023	13:16	00:16	7,58%
Identificação das Falhas	04/06/2023	18:50	04/06/2023	19:00	00:10	4,74%
Busca pelos Defeitos	05/06/2023	19:10	05/06/2023	19:40	00:30	14,22%
Correção dos Defeitos	06/06/2023	20:00	06/06/2023	20:30	00:30	14,22%
Quantidade de falhas encontradas					3	
Total em horas					**03:31**	

Ao analisar a tabela 3, nota-se que a maioria do esforço foi alocada nas etapas de planejamento e implementação dos testes de aceitação, correspondendo a cerca de 28,44% do esforço total em cada uma. Este padrão está alinhado com a literatura da área, que indica que uma preparação e implementação cuidadosas podem potencializar a eficácia dos testes (FEWSTER & GRAHAM, 1999; Perry, 2006).

Por sua vez, a execução dos testes, fase onde os testes são de fato conduzidos, representou apenas 2,37% do esforço total. Este resultado indica uma elevada eficiência na execução dos testes, o que está de acordo com pesquisas anteriores que destacam que os testes automatizados podem ser mais velozes e eficientes que os testes manuais (DUSTIN, 2009).

A análise dos resultados e a identificação das falhas consumiram cerca de 12,32% do esforço total. Esta proporção também é consistente com o conhecimento prévio na área, que enfatiza a importância de alocar tempo suficiente para a análise dos resultados dos testes e para a identificação de falhas (MYERS, SANDLER, & BADGETT, 2011).

As etapas de busca pelos defeitos e correção dos defeitos, juntas, somaram 28,44% do esforço total. Este dado é esperado, uma vez que estas etapas são vitais para assegurar a qualidade do software (JORGENSEN, 2003).

A Tabela 4, apresenta o registro de atividades relacionadas à funcionalidade "Incluir Item na medição".

Tabela 4 - Registro de esforço das atividades Incluir Item na medição

Atividade	Data de início	Hora de Início	Data de Fim	Hora de Fim	Esforço	Percentual do Esforço total
Planejamento de Testes	07/06/2023	16:00	07/06/2023	17:00	01:00	27,52%
Implementação dos Testes Aceitação	08/06/2023	17:00	08/06/2023	18:00	01:00	27,52%
Execução dos Testes	09/06/2023	18:30	09/06/2023	18:35	00:05	2,29%
Análise dos Resultados	01/07/2023	13:20	01/07/2023	13:43	00:23	10,55%
Identificação	10/06/2023	18:50	10/06/2023	19:00	00:10	4,59%

das Falhas						
Busca pelos Defeitos	11/06/2023	19:10	11/06/2023	19:40	00:30	13,76%
Correção dos Defeitos	12/06/2023	20:00	12/06/2023	20:30	00:30	13,76%
Quantidade de falhas encontradas					4	
Total em horas					**03:38**	

Continuando com a análise realizada anteriormente, a funcionalidade "Incluir Item na Medição" do sistema MESUR também passou por rigorosos testes de aceitação, garantindo que os usuários possam adicionar itens de medição com sucesso. A descrição detalhada e os scripts Cypress foram utilizados para implementar e executar os testes de aceitação.

Os dados da tabela revelam que o planejamento e a implementação dos testes foram as etapas que mais exigiram esforço, cada uma ocupando 27,52% do esforço total. A execução dos testes, sendo a fase onde os testes são realmente realizados, representou apenas 2,29% do total de esforço.

A análise dos resultados e a identificação das falhas conjuntas constituíram 15,14% do esforço total. A análise dos resultados é uma etapa crucial no processo de testes, sendo necessário tempo suficiente para analisar os resultados e identificar falhas (MYERS, SANDLER, & BADGETT, 2011).

As etapas de busca e correção de defeitos representaram juntas 27,52% do esforço total. A busca e correção de defeitos são etapas essenciais para assegurar a qualidade do software (PERRY, 2006).

No que diz respeito à funcionalidade "Incluir Item na Medição", foram identificadas quatro falhas durante o processo de testes. Esse número, embora seja maior do que o observado na funcionalidade "Incluir Medição", está dentro do esperado para essa etapa do processo de testes.

Na sequência da análise anterior, a funcionalidade "Visualizar Medição" do sistema MESUR passou por um processo de testes de aceitação visando assegurar que os usuários serão capazes de visualizar as medições da sua empresa de maneira eficaz. A Tabela 5, apresenta o registro do esforço e número de falhas encontradas para as atividades de teste desta funcionalidade:

Tabela 5 - Registro de esforço das atividades Visualizar Medição

Atividade	Data de início	Hora de Início	Data de Fim	Hora de Fim	Esforço	Percentual do Esforço total
Planejamento de Testes	12/06/2023	16:00	12/06/2023	17:00	01:00	38,96%
Implementação dos Testes Aceitação	13/06/2023	17:00	13/06/2023	17:15	00:15	9,74%
Execução dos Testes	14/06/2023	18:30	14/06/2023	18:32	00:02	1,30%
Análise dos Resultados	01/07/2023	13:43	01/07/2023	13:50	00:07	4,55%
Identificação das Falhas	14/06/2023	18:50	14/06/2023	19:00	00:10	6,49%
Busca pelos Defeitos	15/06/2023	19:10	15/06/2023	19:40	00:30	19,48%
Correção dos Defeitos	16/06/2023	20:00	16/06/2023	20:30	00:30	19,48%
Quantidade de falhas encontradas					1	
Total em horas					**02:34**	

Os scripts Cypress utilizados foram responsáveis por verificar a existência de um registro específico no grid, o somatório de Ponto de Função Bruto de Medições, e a quantidade de medições realizadas. Com a aplicação desses testes, garantiu que o usuário pode não apenas visualizar a lista de medições, mas também ter acesso a importantes métricas relacionadas a estas medições.

Observando a tabela dos registros das atividades e o esforço dedicado aos testes de aceitação para essa funcionalidade, o planejamento dos testes foi a etapa que demandou o maior esforço, representando 38,96% do esforço total. Isso reafirma a importância da fase de planejamento, conforme destacado em estudos anteriores (FEWSTER & GRAHAM, 1999; JORGENSEN, 2003).

A implementação dos testes de aceitação, embora breve, ainda constituiu uma parte significativa do esforço total, chegando a 9,74%. Isso reforça que, mesmo com

testes automatizados, a fase de implementação ainda requer uma considerável atenção e esforço (DUSTIN, 2009).

A execução dos testes tomou apenas 1,30% do esforço total, ilustrando a eficiência da automação dos testes. A análise dos resultados e a identificação das falhas também foram eficientes, totalizando 11,04% do total de esforço.

As etapas de busca e correção de defeitos, juntas, totalizaram 38,96% do total de esforço total. Foi identificado e corrigido um único defeito durante o processo de testes, o que sugere que o processo de desenvolvimento inicial foi bastante eficaz.

A análise da funcionalidade "Exportar Medição" do sistema MESUR é discutida a seguir. Os scripts de teste Cypress foram desenvolvidos para abordar quatro cenários distintos: exportação de uma medição individual, exportação de todas as medições como um arquivo Excel, exportação apenas dos dados visíveis no grid como Excel, e a exportação de uma medição individual via interação com o botão direito no grid. Esses testes garantiram que os usuários têm flexibilidade para acessar e armazenar informações de acordo com suas necessidades específicas.

A Tabela 6, apresenta os dados de esforço registrados para as atividades de teste desta funcionalidade

Tabela 6 - Registro de esforço das atividades “Exportar Medição”

Atividade	Data de início	Hora de Início	Data de Fim	Hora de Fim	Esforço	Percentual do Esforço total
Planejamento de Testes	22/06/2023	16:00	22/06/2023	17:00	01:00	32,26%
Implementação dos Testes Aceitação	23/06/2023	17:00	23/06/2023	17:25	00:25	13,44%
Execução dos Testes	24/06/2023	18:30	24/06/2023	18:33	00:03	1,61%
Análise dos Resultados	01/07/2023	14:07	01/07/2023	14:20	00:13	6,99%
Identificação das Falhas	24/06/2023	18:50	24/06/2023	19:15	00:25	13,44%
Busca pelos Defeitos	25/06/2023	19:10	25/06/2023	19:40	00:30	16,13%
Correção dos Defeitos	26/06/2023	20:00	26/06/2023	20:30	00:30	16,13%

Quantidade de falhas encontradas					3	
Total em horas					**03:06**	

Analisando a Tabela 6, é possível observar que o planejamento dos testes foi a atividade que exigiu o maior esforço, representando 32,26% do esforço total. A fase de implementação dos testes de aceitação, que compreendeu 13,44% do esforço total, destacou novamente a importância desta fase, mesmo no contexto de testes automatizados (DUSTIN, 2009).

A execução dos testes foi rápida, consumindo apenas 1,61% do esforço total, ilustrando a eficiência dos testes automatizados. A análise dos resultados e a identificação das falhas levaram, em conjunto, 20,43% do esforço total.

A busca e correção de defeitos constituíram 32,26% do esforço total. Foram identificadas e corrigidas três falhas durante os testes, sugerindo que esta fase é crucial para garantir a qualidade do software.

A funcionalidade "Apresentar *dashboard*" no sistema MESUR desempenha um papel crucial na visualização e na compreensão de medidas por meio de gráficos. Isso é consistentemente apoiado pela literatura que enfatiza a importância dos painéis de controle como ferramentas essenciais para a tomada de decisão baseada em dados (RASMUSSEN, CHEN, & BANSAL, 2009). Eles fornecem uma representação visual dos dados, permitindo aos usuários ver tendências, padrões e percepções rapidamente, facilitando a tomada de decisões eficazes. A Tabela 7 apresenta o registro das atividades e seus respectivos esforços para as atividades de testes de aceitação.

Tabela 7 - Registro de esforço das atividades Apresentar *dashboard*

Atividade	Data de início	Hora de Início	Data de Fim	Hora de Fim	Esforço	Percentual do Esforço total
Planejamento de Testes	17/06/2023	16:00	17/06/2023	16:30	00:30	21,58%
Implementação dos Testes Aceitação	18/06/2023	17:00	18/06/2023	17:23	00:23	16,55%
Execução dos	19/06/2023	18:30	19/06/202	18:34	00:04	2,88%

Testes			3			
Análise dos Resultados	01/07/2023	14:00	01/07/2023	14:07	00:07	5,04%
Identificação das Falhas	19/06/2023	18:50	19/06/2023	19:05	00:15	10,79%
Busca pelos Defeitos	20/06/2023	19:10	20/06/2023	19:40	00:30	21,58%
Correção dos Defeitos	21/06/2023	20:00	21/06/2023	20:30	00:30	21,58%
Total em horas					**2:19**	

Os testes de aceitação para essa funcionalidade centraram-se em dois cenários: exibir com sucesso o painel de controle com todos os gráficos e falhar ao tentar exibir o painel de controle quando não há medidas cadastradas.

Os testes Cypress foram desenvolvidos para verificar a presença de sete gráficos diferentes no painel de controle. Nossos resultados mostram que o planejamento de testes e a busca e correção de defeitos representaram as maiores proporções de esforço, ambas com 21,58%. Isso está alinhado com o trabalho de Rocha (2001), que encontrou padrões semelhantes em suas investigações sobre o esforço de teste. A implementação de testes de aceitação representou 16,55% do esforço total, a identificação de falhas 10,79%, e a análise dos resultados 5,04%. A execução dos testes, mais uma vez, demonstrou a eficiência dos testes automatizados, consumindo apenas 2,88% do esforço total (DUSTIN et al., 2009).

A funcionalidade "Imprimir Relatório" no sistema MESUR foi avaliada por meio de testes automatizados em três cenários distintos: exportação do relatório sem agrupar itens pela situação de aferição, exportação do relatório agrupando itens pela situação de aferição e cancelamento da operação de exportação. Os resultados do esforço para as atividades de testes de aceitação são apresentados na Tabela 8.

Tabela 8 - Registro de esforço das atividades Imprimir Relatório

Atividade	Data de início	Hora de Início	Data de Fim	Hora de Fim	Esforço	Percentual do Esforço total
Planejamento de Testes	26/06/2023	16:00	26/06/2023	17:00	01:00	33,15%
Implementação dos Testes	27/06/2023	17:00	27/06/2023	17:25	00:25	13,81%

Aceitação						
Execução dos Testes	28/06/2023	18:30	28/06/2023	18:33	00:03	1,66%
Análise dos Resultados	01/07/2023	14:07	01/07/2023	14:15	00:08	4,42%
Identificação das Falhas	28/06/2023	18:50	28/06/2023	19:15	00:25	13,81%
Busca pelos Defeitos	29/06/2023	19:10	29/06/2023	19:40	00:30	16,57%
Correção dos Defeitos	30/06/2023	20:00	30/06/2023	20:30	00:30	16,57%
Quantidade de falhas encontradas					3	
Total em horas					**03:01**	

O esforço despendido durante as fases de teste é consistente com as descobertas de Van Solingen, Kusters e Trienekens (2020), que enfatizam que o planejamento de testes geralmente é a atividade que mais consome tempo. Neste trabalho, o planejamento de testes representou 33,15% do esforço total. As correções de defeitos e a busca por defeitos também foram fatores significativos, cada um com 16,57% do total de esforço.

A execução dos testes representou a menor parcela do esforço total (1,66%), destacando novamente a eficiência dos testes automatizados. Enquanto isso, a implementação dos testes de aceitação consumiu 13,81% do esforço total, seguida pela análise dos resultados (4,42%) e pela identificação de falhas (13,81%).

A Tabela 9 apresenta uma análise do esforço despendido e do número de falhas encontradas durante os testes de aceitação para várias funcionalidades do sistema MESUR.

Tabela 9 - Registro de esforço das atividades e do número de falhas encontradas

Funcionalidade	Esforço (horas:minutos)	Percentual do Esforço total	Quantidade de Falhas Encontradas
Incluir Medição	03:31	19,38%	3
Incluir Itens	03:38	20,02%	4
Visualizar Dados Medição	02:34	14,14%	1

Apresentação Dashboard	02:19	12,76%	1
Exportar Medição	03:06	17,08%	3
Imprimir Relatório	03:01	16,62%	3
Total em horas	**18:09**	**100%**	**12**

É possível observar que o total de esforço dedicado aos testes de aceitação foi de 18 horas e 9 minutos, com uma média de aproximadamente 3 horas dedicadas a cada funcionalidade. A funcionalidade que exigiu mais esforço foi a "Incluir Itens", com um total de 3 horas e 38 minutos e 4 falhas encontradas.

Esses resultados evidenciam que os processos de inclusão, sejam eles da medição ou dos itens, foram os que exigiram maior esforço, o que pode ser explicado pela maior complexidade dessas funcionalidades. Segundo O'Regan (2014), às funcionalidades mais complexas ou críticas tendem a consumir mais tempo durante os testes de aceitação, visto que exigem um maior cuidado e detalhamento para garantir sua correta funcionalidade.

Por outro lado, as funcionalidades "Visualizar Dados Medição" e "Apresentação *Dashboard*" exigiram menos esforço e encontraram menos falhas. O que pode estar relacionado à sua menor complexidade ou ao fato de serem funções mais diretas e com menos possibilidades de erro.

É importante mencionar que a quantidade de falhas encontradas não necessariamente reflete a qualidade ou a complexidade das funcionalidades, mas pode estar mais relacionada ao rigor e à eficácia do processo de teste (PEZZE & YOUNG, 2007).

A distribuição do esforço total de testes entre as funcionalidades indica uma estratégia de distribuição de esforço equitativa. Este é um aspecto fundamental na alocação de recursos em testes de software, conforme enfatizado por Jorgensen (2016).

A funcionalidade "Incluir Itens" recebeu o maior esforço em termos percentuais, correspondendo a 20,02% do total. De acordo com Kitchenham et al. (2002), uma maior dedicação de tempo em testes de uma funcionalidade pode levar à identificação de um número maior de falhas, corroborando a contagem de 4 falhas observadas nesta funcionalidade.

Contrastando com "Incluir Itens", a funcionalidade "Visualizar Dados Medição"

exigiu 14,14% do esforço total de testes, porém, resultou na identificação de apenas uma falha. Esse indicador sugere que o esforço de teste pode não ser o único fator determinante para a identificação de falhas, e que a qualidade inicial do código também desempenha um papel significativo (PFLEEGER e ATLEE, 2009).

Analisando tecnicamente a funcionalidade "Imprimir Relatório", que consumiu 16,62% do esforço total e resultou em 3 falhas detectadas, é possível inferir que o custo-benefício do esforço de teste foi consistente, de acordo com as métricas do projeto. Essa proporção de esforço e falhas detectadas reflete a relação entre o investimento em testes (esforço) e o retorno deste investimento (falhas detectadas), o que, na literatura de testes de software, é um fator determinante na avaliação da eficácia dos testes (PFLEEGER e ATLEE, 2009).

Entretanto, é importante destacar que a eficácia dos testes não pode ser avaliada apenas com base na contagem de falhas. Jones e Bonsignour (2011) argumentam que a eficácia dos testes também deve ser avaliada com base na importância das falhas detectadas, enquanto a detecção de falhas críticas proporciona um retorno maior do investimento em testes do que a detecção de falhas de baixa severidade.

Consequentemente, apesar de a funcionalidade "Imprimir Relatório" demonstrar uma proporção consistente entre o esforço de teste e as falhas detectadas, seria útil complementar essa análise com uma avaliação da severidade das falhas encontradas para uma compreensão mais completa da eficácia do teste para esta funcionalidade.

Com base na análise detalhada do esforço de teste aplicado às funcionalidades e as respectivas falhas encontradas, pode-se concluir que essas métricas fornecem uma visão crucial da eficácia dos testes de software em questão. Este entendimento evidencia potenciais áreas do processo que podem se beneficiar de aperfeiçoamentos, seja na otimização da alocação de esforço ou no foco em aspectos particulares que possam apresentar mais falhas. A análise destas informações contribui não apenas para a qualidade do produto de software em desenvolvimento, mas também para a eficiência do processo de desenvolvimento na totalidade, apontando para estratégias eficazes de teste e alocação de recursos (JONES e BONSIGNOUR, 2011; PFLEEGER e ATLEE, 2009).

6. CONCLUSÃO

Neste estudo, foram implementados testes automatizados baseados na metodologia ágil BDD (Behaviour Driven Development) na aplicação web MESUR, visando comparar os resultados antes e depois da implementação. Para isso, selecionaram-se funcionalidades-alvo para testes de aceitação e utilizou-se o framework Cypress para automatizar os cenários de teste. Os resultados indicaram maior eficiência e rapidez na execução dos testes automatizados em comparação com os testes manuais, com economia de tempo significativa. Além disso, a identificação e correção de falhas foram facilitadas, contribuindo para a garantia de qualidade do software. A metodologia BDD permitiu a definição clara dos requisitos funcionais e comportamentais, promovendo uma colaboração efetiva entre equipes e melhorando a eficácia dos testes automatizados. Os resultados enfatizam a relevância contínua da automação de testes e a importância de metodologias adequadas para o desenvolvimento de aplicativos web de qualidade.

A implementação de testes automatizados baseados na metodologia BDD proporcionou maior eficiência e rapidez na aplicação web MESUR. Os testes automatizados mostraram-se mais eficientes em comparação com os testes manuais, com economia de tempo variando de 11% a 30%. Isso evidencia os benefícios da automação na redução de erros humanos e na garantia de qualidade do software. Além disso, a metodologia BDD facilitou a definição dos requisitos e cenários de teste, promovendo uma colaboração efetiva entre as equipes.

Os resultados destacam a importância de um planejamento cuidadoso e uma implementação bem estruturada dos testes de aceitação. A etapa de planejamento e implementação demandou maior esforço em todas as funcionalidades analisadas, reforçando a necessidade de preparação adequada para garantir a eficácia dos testes automatizados.

A análise dos resultados e a identificação de falhas foram etapas cruciais para a melhoria contínua da aplicação web MESUR. Essas atividades consumiram uma parcela significativa do esforço total, enfatizando a importância de uma avaliação minuciosa dos resultados dos testes e a identificação rápida e efetiva das falhas encontradas.

Além dos benefícios já mencionados, a implementação de testes automatizados baseados em BDD na aplicação web MESUR também contribuiu para a agilidade no

processo de desenvolvimento. Com a automação dos testes, os desenvolvedores puderam obter feedback rápido sobre a qualidade do software, permitindo a detecção precoce de problemas e a realização de correções de forma mais ágil. Isso possibilitou um ciclo de desenvolvimento mais curto, com interações mais frequentes e entregas mais rápidas de novas funcionalidades.

Outro aspecto relevante foi o aumento da confiabilidade dos testes automatizados. Ao padronizar os cenários de teste em linguagem natural através do BDD, houve uma maior clareza e precisão na descrição dos comportamentos esperados do sistema. Dessa forma, os testes automatizados se tornaram mais robustos e menos propensos a erros de interpretação. A confiabilidade dos testes é crucial para garantir a precisão das validações e evitar falsos positivos ou negativos, resultando em um processo de teste mais confiável e confiante para toda a equipe de desenvolvimento.

Em resumo, a implementação de testes automatizados com base na metodologia BDD na aplicação web MESUR proporcionou benefícios abrangentes, incluindo maior eficiência, rapidez, qualidade, agilidade e confiabilidade. Essa abordagem reforça a importância contínua da automação de testes na engenharia de software e destaca o papel fundamental da metodologia BDD na criação de testes precisos e colaborativos. Com esses resultados em mente, espera-se que equipes de desenvolvimento considerem seriamente a adoção de testes automatizados e metodologias ágeis como BDD para aprimorar a qualidade e a eficácia de seus processos de desenvolvimento de aplicativos web.

6. REFERÊNCIAS

ADZIC, G. **Specification by Example: How Successful Teams Deliver the Right Software. Manning Publications**, 2011.

AGILE ALLIANCE. **Manifesto for Agile Software Development**, 2001. Disponível em: <http://www.agilealliance.org/>. Acesso em 02 de jul. de 2023.

AHMAD, F. **Cypress: A Comprehensive Guide. Independently Published**, 2019.

ALMEIDA, G. **Fatores de escolha entre metodologias de desenvolvimento de software tradicionais e ágeis**. São Paulo: Universidade de São Paulo, Escola Politécnica; 2017. Disponível em: <https://www.teses.usp.br/teses/disponiveis/3/3136/tde-11042017-143311/publico/GuilhermeAugustoMachadodeAlmeidaCorr17.pdf>. Acesso em 20 abr. de 2023.

ARAÚJO, M. C. S. **Uma abordagem orientada a aspectos para escrita de história do usuário com gherkin**. Recife, 2017. Disponível em: <https://attena.ufpe.br/bitstream/123456789/30871/1/DISSERTAÇÃO%20Mike%20Christian%20de%20Sousa%20Araujo.pdf>. Acesso em 05 de mar. 2022

ARRUDA, N. S. A. **Metodologias de teste de software**. Campinas Grande, 2006. Disponível em: <http://dspace.sti.ufcg.edu.br:8080/jspui/bitstream/riufcg/19175/1/NILO%20SÉRGIO%20DE%20AQUINO%20ARRUDA%20-%20RELATÓRIO%20DE%20ESTÁGIO%20ENG.%20ELÉTRICA%202006.pdf> Acesso em 06 de mar. 2022

ASSIS, F. B. **Desenvolvimento de software dirigido por teste de aceitação**. Belo Horizonte, 2012. Disponível em: <https://repositorio.ufmg.br/bitstream/1843/BUOS-94LMVC/1/fabianabragadeassis.pdf> Acesso em 06 de mar. 2022

ASSIS, R. F. de. **A importância do teste de aceitação na garantia da qualidade de software**. Dissertação (Mestrado em Ciência da Computação) - Universidade de São Paulo, 2012.

Associação brasileira das empresas de software (ABES). **Abes ponta crescimento acima da média para o mercado brasileiro de software em 2022**, 2022.. Disponível em:<https://www.etice.ce.gov.br/2022/05/20/abes-aponta-crescimento-acima-da-media-para-o-mercado-brasileiro-de-software-em-2022/>. Acesso em 20 abr. de 2023.

ASTELS, D. **Test-Driven Development: A Practical Guide**. Prentice Hall Professional, 2006.

B. GEORGE, L. WILLIAMS, **An Initial Investigation of Test Driven Development In Industry.** Disponível em http://http://collaboration.csc.ncsu.edu/laurie/Papers/TDDpaperv8.pdf. Acesso em 05 de mar. de 2023

BECK, K. **eXtreme Programming Explained**, Addison Wesley, 2000.

BECK, K. **Test-Driven Development By Example**. Addison Wesley, 2002. Estados Unidos.

BECK, K., ANDRES, C. **Extreme Programming Explained: Embrace Change**, 2004. Addison-Wesley Professional, Boston. Disponível em:<https://scirp.org/reference/referencespapers.aspx?referenceid=1241493> . Acesso em 27 de fev. de 2023.

BEIZER, B. **Software testing techniques**. Van Nostrand Reinhold Co, 1990.

BLACK, R. **Custo dos testes de Software x Custo do Defeito de Software**, 2000. Disponível em:

<https://thebugbox.wordpress.com/2011/07/10/custo-dos-testes-de-software-x-custo-do-defeito-de-software/>. Acesso em 15 de abr. de 2023

BLACK, REX. **Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing**. 1ª Ed. Wiley: 2000

Boehm, B., **Software Engineering, IEEE Transactions on Computers**, 1976.

BORGES, Eduardo N. **Conceitos e Benefícios do Test Driven Development**. Universidade Federal do Rio Grande do Sul / Instituto de Informática, 2006.

BULGARELI, J. A., FOSCHINI, I. J. **Requisitos testáveis com behaviour-driven development**, 2015.Disponível em: <http://sucesurs.org.br/sites/default/files/2019-04/Artigo_Requisitos_Testaveis.pdf>. Acesso em 05 de mar. de 2023

CHAPLIN, D. **Test first programming**, TechZone, 2001.

CHEN, T. Y.; et al. **Software Performance and Scalability: A Quantitative Approach.** Wiley, 2020.

CHEROSKY, N. **Gherkin: a language for executable specifications**. The Agile Warrior, 2012.

COHN, M. **Succeeding with Agile: Software Development Using Scrum**. Estados Unidos: Addison-Wesley, 2010.

CORRÊA, J. S. **Implementação da História de Usuário :Experiência na Fábrica de Tecnologias Turing** (FTT). Anápolis, 2018 . Disponível em: <http://anais.unievangelica.edu.br/index.php/adalovelace/article/view/4645/2774> Acesso em 05 de mar. 2022

CUCUMBER. **BDD and the new model for testing.** Cucumber.io, 2018.

CYPRESS. **Why Cypress?**, 2020. Disponível em: <https://docs.cypress.io/guides/overview/why-cypress> Acesso em 8 de abr. de 2023.

DATTA, S. **Test Automation Using Selenium WebDriver with Java: Step by Step Guide.** CreateSpace Independent Publishing Platform, 2012.

DE SOUZA, J. V. **Avaliação de Ferramentas de Apoio ao Teste de Sistemas de Informação**. 2016. Disponível em: <https://repositorio.ifpe.edu.br/xmlui/handle/123456789/596?show=full> Acesso em 27 de fev. de 2023.

DEITEL, P.; DEITEL, H. **JavaScript for Programmers.** Pearson, 2020.

DUSTIN, E. **Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality,** 2009.

DUSTIN, E.; RASHKA, J.; PAUL, J. **Automated Software Testing: Introduction, Management, and Performance: Introduction, Management, and Performance**, 2009.

EXAME**. O crescimento do negócio de provedor de internet na pandemia**. Exame, Rio de Janeiro, jan. 2022. Disponível em: <https://exame.com/bussola/o-crescimento-do-negocio-de-provedor-da-internet-na-pandemia/>. Acesso em: 21 fev. 2022

FEWSTER, M., GRAHAM, D. **Software Test Automation: Effective Use of Test Execution Tools**, 1999.

GELPERIN, D., HETZEL, B. **The growth of software testing**, 1987. Disponível em <https://dl.acm.org/doi/abs/10.1145/62959.62965> Acesso em 27 de fev. de 2023

GEORGE, L. WILLIAMS, **An Initial Investigation of Test Driven Development InIndustry**. Disponível em http://http://collaboration.csc.ncsu.edu/laurie/Papers/TDDpaperv8.pdfcios do Test Driven Development. Universidade Federal do Rio Grande do Sul / Instituto de Informática, 2006.

GEORGEA, B.; WILLIAMS, L. **A structured experiment of test-driven development.** Falls Church: 2004 Disponível em: <https://www.sciencedirect.com/science/article/abs/pii/S0950584903002040?via%3Dihub>. Acesso em: 21 de fev. 2022

HILLENBRAND, M. **Automated Testing in JavaScript: An Introduction to Cypress.** DEV Community, 2020.

IBGE. **Pesquisa mostra que 82,7% dos domicílios brasileiros têm acesso à internet.** IBGE. Disponível em: <https://www.gov.br/mcom/pt-br/noticias/2021/abril/pesquisa-mostra-que-82-7-dos-domicilios-brasileiros-tem-acesso-a-internet>. Acesso em: 21 fev. 2022

IBGE. **Uso de internet, televisão e celular no Brasil. IBGE**. Disponível em: <https://educa.ibge.gov.br/jovens/materias-especiais/20787-uso-de-internet-televisao-e-celular-no-brasil.html>. Acesso em: 21 fev. 2022

IEEE. **IEEE Guide to Software Requirements Specifications** (Std 830-1998). New York: IEEE, 1998.

IEEE. **IEEE Standard Glossary of Software Engineering Terminology,** Std 610.12-1990. 1990.

ISTQB (2018). **Certified Tester Foundation Level Syllabus** - International Software Testing Qualifications Board (ISTQB). Disponível em: https://www.bstqb.org.br/. Acesso em 08 de abr. de 2023.

JONES, C. **Applied Software Measurement: Global Analysis of Productivity and Quality.** 3rd ed. McGraw-Hill, 2008.

JONES, C.; BONSIGNOUR, O. **The Economics of Software Quality.** Addison-Wesley, 2011.

JØRGENSEN, M. **Unit effects in software project effort estimation: Work-hours gives lower effort estimates than workdays**, 2016. Disponível em: <https://www.sciencedirect.com/science/article/abs/pii/S0164121216300085>. Acesso em 15 de abr. de 2023.

JØRGENSEN, P.C. **Software Testing: A Craftsman's Approach.** CRC Press, 2003.

KANER, C.; BACH, J.; PETTICHORD, B. **Lessons Learned in Software Testing: A Context-Driven Approach**. Wiley, 2008.

KANER, C.; FALK, J. & NGUYEN, H. Q. **Testing computer software**, 2nd ed. John Wiley and Sons Publishing: EUA, 1999.

KANER, C.; NGUYEN, H. Q.; FALK, J. **Testing Computer Software, 2nd edition**. John Wiley & Sons, Inc., 1988.

KITCHENHAM EL AT. **Preliminary guidelines for empirical research in software engineering**, 2002. Disponível em: <https://ieeexplore.ieee.org/abstract/document/1027796> Acesso em 15 de abr. de 2023.

LARMAN, C., BASILI, V. R. **Iterative and IncrementalDevelopment:A Brief History**, 2003. Disponível em:

<https://www.craiglarman.com/wiki/downloads/misc/history-of-iterative-larman-and-basili-ieee-computer.pdf>. Acesso em 27 de fev. de 2023.

LEHMAN, M., BELADY, L. **Program Evolution: Processes of Software Change**, Academic Press, London, 1985.

MANNING, J.; BODILY, S. **Web Testing with Cypress: Modern End-to-End Testing for the Web.** Manning Publications, 2018.

MOBARAYA, F., ALI, S. **Technical Analysis Of Selenium And cypress as Functional Automation Framework For Modern Web Application Testing**, 2019. Disponível em: <http://www.producao.uff.br/conteudo/rpep/volume62006/RelPesq_V6_2006_03.pdf>. Acesso em 13 de fev. de 2023.

MOLINA, A.; FLORES, R.; VIZCAÍNO, A.; PIATTINI, M. **Web Acceptance Test: un enfoque para el diseño de casos de pruebas de aceptación en aplicaciones web.** Revista de Ciências Técnicas, v. 35, n. 1, p. 85-96, 2017.

MYERS, G. J. **The Art of Software Testing**, 2004. Disponível em: <https://onlinelibrary.wiley.com/doi/book/10.1002/9781119202486>. Acesso em 08 de abr. de 2023.

MYERS, G.J., SANDLER, C., BADGETT, T. **The Art of Software Testing. 3rd ed,** 2011.

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST). **Reports that software defects cost about $60 billion**, 2002. Disponível em: <http://www.producao.uff.br/conteudo/rpep/volume62006/RelPesq_V6_2006_03.pdf>. Acesso em: Acesso em 02 de jul. de 2023.

NORTH, D. **Introducing BDD**, 2014. Disponível em: <http://dannorth.net/introoducing-bdd/>. Acesso em: 15 jun. 2023

O'REGAN, G. **Introducing Software Testing.** Springer, 2014.

PAPADOPOULOS, I., KOGIAS, D., PATRIKAKIS, C. **Enhancing the Student's Logical Thinking with Gherkin Language**, 2017. IEEE Global Engineering Education Conference (EDUCON), Athens, Greece, 2017.

PERRY, W. **Effective Methods for Software Testing.** 3rd ed. Wiley, 2006.

PETTICHORD, B. **Is Automated Testing Worth It?** STQE Magazine, v. 4, n. 3, p. 23-27, 2002.

PEZZÈ, M.; YOUNG, M. **Software Testing and Analysis: Process, Principles, and Techniques.** John Wiley & Sons, 2007.

PFLEEGER, S. L.; ATLEE, J. M. **Software Engineering: Theory and Practice.** Pearson, 2009.

PRESSMAN, R. S. **Engenharia de Software**. 6ª. ed. São Paulo: MacGrow-Hill, 2006

PUGH, K. **Lean-Agile Acceptance Test-Driven Development: Better Software Through Collaboration.** Addison-Wesley Professional, 2011.

RAHARJANA, I.K., HARRIS, F., JUSTITIA, A. **Tool for Generating Behavior-Driven Development Test-Cases**, 2020. Disponível em: <https://e-journal.unair.ac.id/JISEBI/article/download/18082/10196>. Acesso em 23 de abr. de 2023.

RASMUSSEN, N., CHENC, Y., BANSAL, M. **Businnes Dashboards A visual Catalog For Design And Deployment**, 2009. Disponível em:

<https://books.google.com.br/books?hl=pt-BR&lr=&id=23A5oPOQ7pYC&oi=fnd&pg=PT5&dq=RASMUSSEN,+CHEN,+%26+BANSAL,+2009&ots=K3BtEfCN6r&sig=VUTT7r7xQh9msNfFHnEw7UH2GA0#v=onepage&q=RASMUSSEN%2C%20CHEN%2C%20%26%20BANSAL%2C%202009&f=false>. Acesso em 23 de abr. de 2023.

RIBEIRO, R. **Escrita do BDD no processo de desenvolvimento**, 2019. Disponível em: <https://medium.com/@ronilsonribeiro/escrita-do-bdd-no-processo-de-desenvolvimento-469ea46540db>. Acesso em 05 de mar. de 2023

ROCHA, A. R. C., MALDONADO, J. C., WEBER, K. C. et al. Qualidade **de software – Teoria e prátic**a, Prentice Hall, São Paulo, 2001.

RODRIGUES, F. M; FARINA. M. R. M. **Aplicação da técnica bdd como aperfeiçoamento em um projeto ágil (scrum)**. 2021. Disponível em: <https://www.e-journal.unair.ac.id/JISEBI/article/view/18082/10196>. Acesso em 28 de fev. 2022

SMART, J. F. **BDD in Action: Behavior-Driven Development for the whole software lifecycle**. Manning Publications, 2014.

SOMMERVILLE, I.. **Software Engineering.** 8ª edição. São Paulo: Addison Wesley, 2007. Sommerville 2007

STOPA, G. R.; Rachid, C. L. **Scrum: metodologia ágil como ferramenta de gerenciamento de projetos.** Juiz de Fora, 2019. Disponível em: <https://seer.uniacademia.edu.br/index.php/cesRevista/article/view/2026> Acesso em 05 de mar. 2022

VAN SOLINGEN, R., KUSTERS, R. J., TRIENEKENS, J M.**Identifying embedded software quality: two approaches**, 2000. Disponível em <https://onlinelibrary.wiley.com/doi/abs/10.1002/(SICI)1099-1638(199911/12)15:6<485::AID-QRE295>3.0.CO;2-3>. Acesso em 15 de abr. de 2023

WYNNE, M.; HELLESDOTTIR, A. **The Cucumber Book: Behaviour-Driven Development for Testers and Developers**. Pragmatic Bookshelf, 2012

ZHANG, H.; ZHOU, Y.; XU, B. **A Survey of Bug Priority and Severity Assignment.** IEEE Software, v. 25, n. 5, p. 46-53, 2008.

8. APÊNDICES

8.1. APÊNDICE A - Cenários BDD do MESUR

Funcionalidade: Incluir Medição

Como gestor do sistema MESUR

Quero incluir uma medição

Para que eu possa ter o controle de ponto de função da minha empresa

Cenário: Falha ao criar uma medição com data inválida

Dado que estou na página de criação de medição

Quando eu preencho o formulário com as seguintes informações:

| Nome = "Medição TCC"

| Sistema = "Sistema do TCC"

| Tipo de Contagem = "Projeto de Melhoria"

| Nível de Detalhamento = "Detalhada"

| Data de Fechamento = "30/02/2022"

Então eu vejo um tooltip no campo com a mensagem "Data inválida" e a borda do campo vermelha.

Cenário: Cancelar a operação ao criar uma medição

Dado que estou na página de criação de medição

Quando eu preencho o formulário com as seguintes informações:

| Nome = "Medição TCC"

| Sistema = "Sistema do TCC"

| Tipo de Contagem = "Projeto de Melhoria"

| Nível de Detalhamento = "Detalhada"

E eu clico no botão "Voltar"

Então eu sou redirecionado a página de listagem de medição e a medição não é incluída

Funcionalidade: Incluir Item na medição

Como gestor do sistema MESUR

Quero incluir um item em uma medição

Para que eu possa adicionar uma quantidade de ponto de função específica na medição.

Cenário: Criar um novo item com sucesso

Dado que estou na página de criação de item

Quando eu preencho o formulário com as seguintes informações:

| Nome do item = "Item TCC"

| Tipo de Função = "EE"

| INM = "INM do TCC"

E eu clico no botão "Salvar"

Então eu vejo a mensagem "Item salvo com sucesso".

Cenário: Falha criar um novo item sem preencher os campos obrigatórios

Dado que estou na página de criação de item

Quando eu preencho o formulário com as seguintes informações:

| Nome = " "

| Sistema = " "

| Tipo de Contagem = " "

| Nível de Detalhamento = " "

Então eu vejo um tooltip nos campos obrigatórios com a mensagem "Campo Obrigatório" e a borda dos campos vermelha.

Cenário: Falha criar um novo item devido ao nome já existente

Dado que estou na página de criação de item

E existe um item na medição com o nome “Item TCC”

Quando eu preencho o formulário com as seguintes informações:

| Nome do item = "Item TCC"

| Tipo de Função = "EE"

| INM = "INM do TCC"

E eu clico no botão "Salvar"

Então eu vejo a mensagem “O item com o nome "Item tcc" já existe na medição!”

Cenário: Cancelar a operação ao criar um item

Dado que estou na página de criação de item

Quando eu preencho o formulário com as seguintes informações:

| Nome do item = "Item TCC"

| Tipo de Função = "EE"

| INM = "INM do TCC"

E eu clico no botão "Voltar"

Então eu sou redirecionado a página de listagem de item e o item não é incluído

Funcionalidade: Visualizar Medição

Como usuário do sistema MESUR

Quero visualizar o grid de informações das medições da minha empresa

Para que eu possa ter um controle das medições da minha empresa.

Cenário: Visualizar Medição com sucesso

Dado que estou na página de visualização de medições

Quando eu acesso o grid com a lista medições cadastradas

Então eu vejo um grid com as colunas Tipo de Contagem, Nível de Detalhe, PF Bruto, PF Líquido.

Cenário: Falha ao visualizar o grid vazio

Dado que estou na página de visualização de medições

Quando eu acesso o grid sem medições cadastradas

Então eu vejo a mensagem "Nenhuma medição registrada."

Funcionalidade: Apresentar dashboard

Como usuário do sistema MESUR

Quero o dashboard relacionado as medições da minha empresa

Para que eu possa ter um controle das medições da minha empresa.

Cenário: Apresentar dashboard com Sucesso

Dado que estou na página de visualização de medição

Quando eu clico no botão Dashboard

Então eu vejo gráficos relacionados aos dados “Quantidade de medição por mês”, “Distribuição PF por fator de impacto”, “Distribuição PF por INM”, “Distribuição por serviço”, "Medições por sistema", "Distribuição PF por linguagem de programação" “e Distribuição PF por área de negócio”

Cenário: Falha ao apresentar o dashboard sem medições cadastradas

Dado que estou na página de visualização de medição

Quando eu clico no botão Dashboard

Então eu vejo gráficos relacionados aos dados “Quantidade de medição por mês”, “Distribuição PF por fator de impacto”, “Distribuição PF por INM”, “Distribuição por serviço”, "Medições por sistema", "Distribuição PF por linguagem de programação" “e Distribuição PF por área de negócio”

Funcionalidade: Exportar Medição

Como usuário do sistema MESUR

Quero exportar uma ou mais medições

Para que eu possa ter acesso a medição no meu computador.

Cenário: Exportar medição individual

Dado que estou na página de edição de medição

Quando eu clico no “Exportar”

Então um arquivo no formato “.xlsx” é baixado no meu computador com todas as informações relacionadas aquela medição

Cenário: Exportar medição via grid – todos os dados como Excel

Dado que estou na página de visualização de medições

Quando eu clico no “Exportar todos os dados como Excel”

Então um arquivo no formato ".xlsx" é baixado no meu computador com informações gerais relacionadas a todas as medições.

Cenário: Exportar medição via grid – dados visíveis como Excel

Dado que estou na página de visualização de medições

Quando eu clico no "Exportar dados visíveis como Excel"

Então um arquivo no formato ".xlsx" é baixado no meu computador com apenas informações que estão sendo exibidas no grid de todas as medições.

Cenário: Exportar medição individual via grid com botão direito

Dado que estou na página de visualização de medições

Quando eu clico em uma medição específica com botão direito

E clico na opção "Exportar Medição"

Então um arquivo no formato ".xlsx" é baixado no meu computador com todas as informações relacionadas aquela medição

Funcionalidade: Imprimir Relatório

Como usuário do sistema MESUR

Quero imprimir um relatório de uma medição específica

Para que eu possa ter acesso fisicamente a situação de aferição dos itens de uma medição.

Cenário: Imprimir relatório sem agrupar os itens pela situação da aferição

Dado que estou na página de edição de itens

Quando eu clico no "Relatório"

E clico no botão "Sim"

Então um arquivo no formato “.pdf” é baixado no meu computador com informações sobre a parte de aferição de todos os itens relacionados aquela medição sem agrupamento pela sua situação de aferição.

Cenário: Imprimir relatório agrupando itens pela situação da aferição

Dado que estou na página de edição de itens

Quando eu clico no “Relatório”

E clico no botão “Não”

Então um arquivo no formato “.pdf” é baixado no meu computador com informações sobre a parte de aferição de todos os itens relacionados aquela medição agrupados pela sua situação de aferição.

Cenário: Cancelar a operação de exportar um relatório

Dado que estou na página de edição de itens

Quando eu clico no “Relatório”

E clico no botão “Cancelar”

Então o arquivo relatório no formato .pdf não é baixado.

8.2. APÊNDICE B - Scripts do Cypress

Cypress e Gherkin são duas ferramentas poderosas que, embora distintas, podem trabalhar de forma complementar. Os scripts em Cypress possuem uma sintaxe que se assemelha à linguagem natural e, de certa forma, se aproximam de Gherkin, uma linguagem de especificação utilizada em testes BDD (*Behavior-Driven Development*). Essa proximidade é importante porque permite uma melhor comunicação entre desenvolvedores, testadores e stakeholders.

Ao escrever scripts em Cypress com uma sintaxe semelhante ao Gherkin, como "Given", "When" e "Then", as etapas dos testes tornam-se mais legíveis e compreensíveis para todas as partes envolvidas no desenvolvimento do software. Além disso, a abordagem BDD estimula a colaboração e a definição clara dos

requisitos do sistema, garantindo que o software atende adequadamente aos critérios de aceitação.

Portanto, a utilização dos scripts em Cypress com uma sintaxe próxima ao Gherkin possibilita a criação de testes automatizados mais eficientes, que refletem diretamente os cenários de negócio definidos em conjunto com os stakeholders, garantindo a entrega de um software de alta qualidade e aderente às expectativas dos usuários. A tabela 10 apresenta o cada funcionalidade e o respectivo script em Cypress escritos para testá-las..

Tabela 10 - Comparativo nome da funcionalidade com o nome do script em cypress.

Funcionalidade	Nome do Script
Criar uma nova medição - Com Sucesso	Medição: Incluir - Com Sucesso
Falha criar uma medição sem preencher os campos obrigatórios	Medição: Incluir - Não preencher os campos obrigatórios
Falha ao criar uma medição com data inválida	Medição: Incluir - Data Inválida
Cancelar a operação ao criar uma medição	Medição: Incluir - Cancelar Operação
Criar um novo item com sucesso	Item Medição: Incluir - Com Sucesso
Falha criar um novo item sem preencher os campos obrigatórios	Item Medição: Incluir - Não preencher os campos obrigatórios
Falha criar um novo item devido ao nome já existente	Item Medição: Incluir - Nome em uso
Cancelar a operação ao criar um item	Item Medição: Incluir - Cancelar Operação
Visualizar Medição - Com Sucesso	Medição: Listar - Verificar Medição incluídos
Falha ao Visualizar - Com Grid Vazio	Medição: Listar - Verificar sem medição cadastradas
Apresentar dashboard - Com Sucesso	Medição: Apresentar dashboard - Com Sucesso
Falha ao apresentar o dashboard sem medições cadastradas	Medição: Apresentar dashboard - Sem medição cadastrada
Exportar medição individual	Medição: Exportar - Publicar Medição Individual

Exportar medição via grid – todos os dados como Excel	Medição: Exportar - Todos os Campos
Exportar medição via grid – dados visíveis como Excel	Medição: Exportar - Campos Dados Visíveis
Exportar medição individual via grid com botão direito	Medição: Exportar - Com botão direito no grid
Imprimir relatório sem agrupar os itens pela situação da aferição	Medição: Imprimir Relatório da Aferição - Não Agrupar Itens pela Situação
Imprimir relatório agrupando itens pela situação da aferição	Medição: Imprimir Relatório da Aferição - Agrupar Itens pela Situação
Cancelar a operação de exportar um relatório	Medição: Imprimir Relatório da Aferição - Cancelar Operação

1. **Funcionalidade: Incluir Medição**

O código abaixo é responsável pelo cenário “Criar uma nova medição com sucesso” descrito no formato BDD, tem como objetivo verificar se uma mensagem de sucesso é exibida corretamente ao tentar inserir uma medição com valores válidos.

```
it('Medição: Incluir - Com Sucesso ', function () {

    //Clicar no botão medição
    cy.get('#MED__incluir').click()

    //Preencher nome da medição
    cy.get('#titulo').type("Medição Automatizada")

    //Preencher o campo Sistema
    cy.get('#sistema_id').click()
    cy.get('#sistema_id').type("Calculadora do Cidadão")
    cy.get('.ng-binding > strong').click()

    //Preencher o campo Tipo de Contagem
    cy.get('#tipo_de_contagem').select("string:M")

    //Preencher o campo Nível de detalhamento
    cy.get('#nivel_detalhe').select("string:EN")
```

```
            //clicar no botão salvar
            cy.get('#MED__salvar').click()

            cy.get('#MED__inicio').click()
            cy.get('#MED__inicio').click()

            //Verificar a existencia do registro no grid
                   cy.get('.ui-grid-viewport').should('contain', "Medição
Automatizada");
        })
```

2. Cenário: Falha ao inserir a medição devido a não preenchimento dos campos obrigatórios.

O código abaixo é responsável pelo cenário “Falha criar uma medição sem preencher os campos obrigatórios” descrito no formato BDD, tem como objetivo verificar se uma mensagem de erro é exibida corretamente ao tentar inserir uma medição com valores vazios.

```
        it('Medição: Incluir - Não preencher os campos obrigatórios',
function () {

        //Clicar no botão medição
        cy.get('#MED__incluir').click()

        //Verifica que o botão Salvar não é apresentado
        cy.get('#MED__salvar').should('not.exist');

        //Preencher nome da medição
        cy.get('#titulo').type("Medição Automatizada")

        //Verifica que o botão Salvar não é apresentado
        cy.get('#MED__salvar').should('not.exist');

        //Preencher o campo Sistema
        cy.get('#sistema_id').click()
        cy.get('#sistema_id').type("Calculadora do Cidadão")
        cy.get('.ng-binding > strong').click()

        //Verifica que o botão Salvar não é apresentado
```

```
        cy.get('#MED__salvar').should('not.exist');

        //Preencher o campo Tipo de Contagem
        cy.get('#tipo_de_contagem').select("string:M")

        //Verifica que o botão Salvar não é apresentado
        cy.get('#MED__salvar').should('not.exist');

        //Preencher o campo Nível de detalhamento
        cy.get('#nivel_detalhe').select("string:EN")

        //Verifica que o botão Salvar não é apresentado
        cy.get('#MED__salvar').should('not.exist');
    })
```

3. Cenário: Cancelar a inclusão da medição.

O código abaixo é responsável pelo cenário “Cancelar a operação ao criar uma medição” descrito no formato BDD, tem como objetivo verificar se a inclusão da medição foi cancelada corretamente, ou seja, não foi inserida no sistema.

```
    it('Medição: Incluir - Cancelar Operação', function () {

        //Clicar no botão medição
        cy.get('#MED__incluir').click()

        //Preencher nome da medição
        cy.get('#titulo').type("Medição Automatizada")

        //Preencher o campo Sistema
        cy.get('#sistema_id').click()
        cy.get('#sistema_id').type("Calculadora do Cidadão")
        cy.get('.ng-binding > strong').click()

        //Preencher o campo Tipo de Contagem
        cy.get('#tipo_de_contagem').select("string:M")

        //Preencher o campo Nível de detalhamento
        cy.get('#nivel_detalhe').select("string:EN")
```

```
        //clicar no botão voltar
        cy.get('#MED__inicio').click()

        //Verificar a existencia do registro no grid
               cy.get('.ui-grid-viewport').should('not.contain',  "Medição
Automatizada");
    })
```

4. Cenário: Falha ao incluir um item na medição devido ao nome já está em uso.

O código abaixo é responsável pelo cenário “Falha criar um novo item devido ao nome já existente” descrito no formato BDD, tem como objetivo verificar se uma mensagem de erro é exibida corretamente ao tentar incluir um item no qual o nome já está em uso na medição selecionada.

```
    it('Item Medição: Incluir - Nome em uso ', function () {

        //selecionar a medição
           cy.get('.ui-grid-row:nth-child(1)  .ui-grid-cell:nth-child(1)  >
.ui-grid-cell-contents').click({ multiple: true })
        cy.wait(2000)

        //clicar no botão Incluir Itens
        cy.get('#MED__incluir_item').click()
        cy.wait(2000)

        //Preencher o nome
        cy.get('#nomeItem').type('Item Automatizado{enter}')

        //Preencher o Tipo de Função
        cy.get('#tipo_funcao_id').select('INM')

        //Preencher INM
        cy.get('#inm_id').select('CA - Componente Arquivo')

        //Clicar no botão salvar
        cy.get('#MED__salvar_item').click()

        //Verificar a mensagem de erro
```

```
            cy.get('.toast-message').should('contain', 'O item com o nome "item automatizado" já existe na medição!')
    })
```

5. Cenário: Falha ao inserir um item devido a não preenchimento dos campos obrigatórios.

O código abaixo é responsável pelo cenário “Falha criar um novo item devido ao não preenchimento dos campos obrigatórios” descrito no formato BDD, tem como objetivo verificar se uma mensagem de erro é exibida corretamente ao tentar inserir um item com valores vazios.

```
      it('Item Medição: Incluir - Não preencher campos obrigatórios ', function () {

        //selecionar a medição
            cy.get('.ui-grid-row:nth-child(1) .ui-grid-cell:nth-child(1) > .ui-grid-cell-contents').click({ multiple: true })

        //clicar no botão itens
        cy.get('#MED__itens').click()
        cy.wait(2000)

        //clicar no botão Incluir Itens
        cy.get('#MED__incluir_item').click()
        cy.wait(2000)

        //Preencher o nome
        cy.get('#nomeItem').type('  ')

        //Preencher o Tipo de Função
        cy.get('#tipo_funcao_id').select('   ')

        //Preencher INM
        cy.get('#inm_id').select('   ')

        //Verifica que o botão Salvar não é apresentado
        cy.get('#MED__salvar_item').should('not.exist');
```

6. Cenário: Cancelar a inclusão de um item em uma medição.

O código abaixo é responsável pelo cenário “Cancelar a operação ao criar um item” descrito no formato BDD, tem como objetivo verificar se a inclusão do item foi cancelada corretamente, ou seja, não foi inserida no sistema.

```
    it('Item Medição: Incluir - Cancelar Operação ', function () {

        //selecionar a medição
           cy.get('.ui-grid-row:nth-child(1)  .ui-grid-cell:nth-child(1)  >
.ui-grid-cell-contents').click({ multiple: true })
        cy.wait(2000)

        //clicar no botão Incluir Itens
        cy.get('#MED__incluir_item').click()
        cy.wait(2000)

        //Preencher o nome
                cy.get('#nomeItem').type('Item  Automatizado  -  Cancelar
Operação{enter}')

        //Clicar no simbolo ⇆
        cy.get('.input-group-addon').click()

        //Preencher o Tipo de Função
        cy.get('#tipo_funcao_id').select('ALI')

        //Preencher INM
        cy.get('#fator_impacto_id').select('number:8332')

        //Clicar no botão Voltar
        cy.get('#MED__inicio').click()

        //Verificar a existencia do registro no grid
                 cy.get('.ui-grid-viewport').should('not.contain',  "Item
Automatizado - Cancelar Operação");
    })
```

7. **Funcionalidade: Visualizar de Medição**

O código abaixo é responsável pelo cenário “Visualizar Medição - Com Sucesso” descrito no formato BDD, tem como objetivo verificar se uma medição está inclusa no sistema e a quantidade de ponto de função bruto que ela possui.

```
    it('Medição: Listar - Verificar Medição incluídos', function(){

        //Verificar a existência do registro no grid
                    cy.get('.ui-grid-viewport').should('contain',  "Medição
Automatizada - Alterada");
        cy.get('.ui-grid-viewport').should('contain', "146");

            //Verifica se o 'Somatório PF Bruto Medições' corresponde a
'146,00'
            cy.get('.ui-grid-footer-cell-row > .ui-grid-coluiGrid-000O >
.ui-grid-cell-contents').should('have.text','146,00');

        //Verifica se a 'Quantidade de Medições corresponde a '1'
            cy.get('.ui-grid-footer-cell-row > .ui-grid-coluiGrid-000B >
.ui-grid-cell-contents').should('have.text','Total de 1 registros');

    })
```

8. **Funcionalidade: Apresentar Dashboard**

O código abaixo é responsável pelo cenário “Apresentar dashboard - Com Sucesso” descrito no formato BDD, tem como objetivo verificar se o dashboard do aplicativo MESUR é exibido corretamente as medições que estão no sistema.

```
    it('Medição: Apresentar dashboard Com Sucesso ', function () {

        //acessar o botão de dashboard
        cy.get('#MED__dashboard_medicao').click()

        //validar os dashboard
            cy.get('.col-md-6:nth-child(2)
.panel-heading').should("Quantidade de medições por mês")
            cy.get('.col-md-6:nth-child(3)
.panel-heading').should("Distribuição PF por fator de impacto")
            cy.get('.col-md-6:nth-child(4)         .panel-heading         >
.ng-binding').should("Distribuição PF por INM")
```

```
            cy.get('.col-md-6:nth-child(5)
.panel-heading').should("Distribuição 'UST por serviço")
            cy.get('.col-md-6:nth-child(6)         .panel-heading         >
.ng-binding').should("Medições por sistema")
            cy.get('.col-md-6:nth-child(8)         .panel-heading         >
.ng-binding').should("Distribuição PF por Linguagem de Programação")
            cy.get('.col-md-6:nth-child(9)
.panel-heading').should("Distribuição PF por Área de Negócio")
    })
```

9. Cenário de Teste: Falha ao apresentar dashboard sem medição.

O código abaixo é responsável pelo cenário “Falha ao apresentar o dashboard sem medições cadastradas” descrito no formato BDD, tem como objetivo verificar se o dashboard do aplicativo MESUR exibe o valor 0 em cada gráfico do dashboard

```
    it('Medição: Apresentar dashboard sem medições cadastradas', function
() {

        //acessar o botão de dashboard
        cy.get('#MED__dashboard_medicao').click()

        //validar os dashboard
            cy.get('.col-md-6:nth-child(2)
.panel-heading').should("Quantidade de medições por mês").contain("0")
            cy.get('.col-md-6:nth-child(3)
.panel-heading').should("Distribuição      PF      por      fator      de
impacto").contain("0")
            cy.get('.col-md-6:nth-child(4)         .panel-heading         >
.ng-binding').should("Distribuição PF por INM").contain("0")
            cy.get('.col-md-6:nth-child(5)
.panel-heading').should("Distribuição 'UST por serviço").contain("0")
            cy.get('.col-md-6:nth-child(6)         .panel-heading         >
.ng-binding').should("Medições por sistema").contain("0")
            cy.get('.col-md-6:nth-child(8)         .panel-heading         >
.ng-binding').should("Distribuição      PF      por      Linguagem      de
Programação").contain("0")
            cy.get('.col-md-6:nth-child(9)
.panel-heading').should("Distribuição      PF      por      Área      de
Negócio").contain("0")
```

```
})
```

9. **Funcionalidade: Exportar de Medição**

O código abaixo é responsável pelo cenário “Exportar medição individual” descrito no formato BDD, tem como objetivo verificar se o arquivo exportado não possui perda de dados e também se está no formato csv.

```
it('Medição: Exportar - Publicar Medição Individual', function () {

    //selecionar a medição
       cy.get('.ui-grid-row:nth-child(1)  .ui-grid-cell:nth-child(1)  >
.ui-grid-cell-contents').click({ multiple: true })

    //clicar no botão exportar
    cy.get('#MED__exportar').click()
    cy.get('#btn-confirmar-exportar-estrutura').click()

    //ler o arquivo exportada
     cy.readFile('cypress/downloads/APF-NomeIgual-Medição Automatizada
- Alterada.xlsx').should('exist')
})
```

10. Cenário: Exportar medição via grid - dados visíveis os campos

O código abaixo é responsável pelo cenário “Exportar medição via grid – todos os dados como Excel” descrito no formato BDD, tem como objetivo verificar se o arquivo exportado não possui perda de dados e também se está no formato csv.

```
it('Medição: Exportar - Campos Visiveis - Padrão', () => {

    //clicar para exportar
    cy.get('.p-speeddial-button').click()

    //exportar dados visíveis
    cy.get('#menuitem-9').click()

    //ler o arquivo e validar

cy.readFile('cypress/downloads/Medições.xlsx').should('exist').and('conta
ins', 'Título da Medição', 'Aferição',
         'Tipo de Contagem', 'Nível de Detalhe', 'Situação', 'PF
Líquido','Esforço (h)', 'Custo ($)')
```

```
})
```

11. Cenário: Exportar medição via grid - com botão direito

O código abaixo é responsável pelo cenário “Exportar medição via grid – dados visíveis como Excel” descrito no formato BDD, tem como objetivo verificar se o arquivo exportado não possui perda de dados e também se está no formato csv.

```
    it('Medição: Exportar - Com botão direito no grid', function () {

        //selecionar a medição e clicar com botão direito
           cy.get('.ui-grid-row:nth-child(1)  .ui-grid-cell:nth-child(1)  >
.ui-grid-cell-contents').rightclick({ multiple: true })

        //Exportar medição
        cy.get('#MED_context_menu_medicao_exportar_medicao').click()
        cy.get('#btn-confirmar-exportar-estrutura').click()

        //ler o arquivo e validar
         cy.readFile('cypress/downloads/APF-NomeIgual-Medição Automatizada
- Alterada.xlsx').should('exist')
        })
})
```

12. **Funcionalidade: Imprimir Relatório Agrupando Itens pela Situação de Aferição**

O código abaixo é responsável pelo cenário “Imprimir relatório agrupando itens pela situação da aferição” descrito no formato BDD, tem como objetivo verificar se o arquivo exportado não possui perda de dados, os itens estão agrupados por situação e também se está no formato pdf.

```
      it('Medição: Imprimir Relatório da Aferição - Agrupar Itens pela
Situação', function () {

        //Selecionar um registro no grid
           cy.get('.ui-grid-row:nth-child(1)  .ui-grid-cell:nth-child(1)  >
.ui-grid-cell-contents').click({ multiple: true })

        //exportar relatorio
        cy.get('#MED__exportar_relatorio_divergencia').click()
```

```
        cy.wait(2000)

        //confirmar no modal
        cy.get('#btn-confirmar-gerar-relatorio').click()

        cy.get('#MED__inicio').click()
    })
```

13. Cenário: Imprimir Relatório Sem Agrupar Itens pela Situação de Aferição

O código abaixo é responsável pelo cenário “Imprimir relatório sem agrupar os itens pela situação da aferição” descrito no formato BDD, tem como objetivo verificar se o arquivo exportado não possui perda de dados, os itens não estão agrupados por situação e também se está no formato pdf.

```
    it('Medição: Imprimir Relatório da Aferição - Não Agrupar Itens pela Situação', function () {

        //Selecionar um registro no grid
           cy.get('.ui-grid-row:nth-child(1)  .ui-grid-cell:nth-child(1)  > .ui-grid-cell-contents').click({ multiple: true })

        //exportar relatorio
        cy.get('#MED__exportar_relatorio_divergencia').click()
        cy.wait(2000)

        //recusar os itens agrupados no modal
        cy.get('#btn-recusar-gerar-relatorio').click()
    })
})
```

14. Cenário: Imprimir Relatório - Cancelar Operação

O código abaixo é responsável pelo cenário “Cancelar a operação de exportar um relatório” descrito no formato BDD, tem como objetivo verificar se o arquivo não foi exportado.

```
    it('Medição: Imprimir Relatório da Aferição - Cancelar Operação ', function () {
```

```
        //Selecionar um registro no grid
           cy.get('.ui-grid-row:nth-child(1)  .ui-grid-cell:nth-child(1)  >
.ui-grid-cell-contents').click({ multiple: true })

        //exportar relatorio
        cy.get('#MED__exportar_relatorio_divergencia').click()

        //Cancelar Operação no modal
              cy.get('#modal-confirmar-gerar-relatorio  >  .modal-dialog  >
.modal-content > .modal-footer > .btn-default').click()
    })
```

9. ANEXOS

9.1. Anexo 1: Termo de autorização de imagem do MESUR

UFES

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
TERMO DE AUTORIZAÇÃO DE IMAGEM

Eu, AUGUSTO MARIANO PINHEIRO, brasileiro, casado, portador da Cédula de identidade RG nº. 1.449.960/SSP-ES, inscrito no CPF/MF sob nº 082.770.267-14, residente à Rua José Malta, nº.163, Fradinhos, Vitória, Espírito Santo, **AUTORIZO** o uso da imagem do software **Mesur** e o nome/imagem da empresa **FATTO CONSULTORIA E SISTEMAS LTDA** do CNPJ 02.434.797/0001-60, em todo e qualquer material entre imagens de vídeo, fotos e documentos, para ser utilizado no **trabalho de conclusão de curso** intitulado **Testes Funcionais de aplicações web com Behavior Driven Development** do responsável Paulo Victor Canal Panno, portador da Cédula de identidade RG n° 28.375.142-8, inscrito no CPF sob n° 150.567.937-01. A presente autorização é concedida a título gratuito, abrangendo o uso da imagem acima mencionada em todo território nacional.

Fica ainda **autorizada**, de livre e espontânea vontade, para os mesmos fins, a cessão de direitos da veiculação das imagens não recebendo para tanto qualquer tipo de remuneração.

Por esta ser a expressão da minha vontade declaro que autorizo o uso acima descrito sem que nada haja a ser reclamado a título de direitos conexos à imagem do software ou a qualquer outro, e assino a presente autorização em 02 vias de igual teor e forma.

Vitória, 09 de Março de 2022.

AUGUSTO MARIANO PINHEIRO:08277026714

Assinado de forma digital por AUGUSTO MARIANO PINHEIRO:08277026714
Dados: 2022.03.09 15:19:54 -03'00'

(Assinatura)