UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Nayara de Oliveira Faustino

Processo automatização de teste para aplicações WEB - Estudo de caso

Uberlândia, Brasil

2025

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Nayara de Oliveira Faustino

Processo automatização de teste para aplicações WEB - Estudo de caso

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Ronaldo Castro de Oliveira

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2025

Nayara de Oliveira Faustino

Processo automatização de teste para aplicações WEB - Estudo de caso

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, maio de 2025:

Prof. Dr. Ronaldo Castro de Oliveira
Orientador

Christiane Regina Soares Brasil
Convidado(a)

Marcelo Zanchetta do Nascimento
Convidado

Uberlândia, Brasil
2025

Resumo

No atual cenário do desenvolvimento de software, observa-se uma crescente demanda por entregas rápidas, funcionais e voltadas à experiência do usuário. No entanto, muitos projetos priorizam produtividade e funcionalidades em detrimento da qualidade, negligenciando etapas fundamentais, como o processo de testes. Essa realidade é ainda mais evidente em empresas que não possuem uma equipe especializada em garantia de qualidade (Quality Assurance) , onde os testes de software são, muitas vezes, inexistentes ou atribuídos exclusivamente aos desenvolvedores, gerando sobrecarga, retrabalho e riscos à confiabilidade do produto final. Dados do mercado de Tecnologia da Informação revelam que uma parcela significativa das empresas ainda não conta com analistas de testes em seus times, e também não compreende a importância da aplicação de testes desde as fases iniciais do ciclo de vida do desenvolvimento. A ausência de práticas estruturadas de verificação e validação pode acarretar falhas em produção, baixa usabilidade, insegurança e custos elevados de manutenção, fatores que comprometem tanto a experiência do usuário quanto a reputação da empresa, ao longo deste trabalho, serão explorados os principais tipos de testes aplicáveis ao contexto web, os benefícios da automação e um passo a passo para a introdução gradual desse processo em equipes de desenvolvimento, considerando desde a concepção do produto até a integração dos testes aos ciclos de entrega contínua. A proposta visa não somente ensinar como estruturar o processo de testes, mas também promover uma mudança de cultura organizacional, enfatizando a importância da qualidade como parte essencial da entrega de software. Por fim, o objetivo central deste estudo é dar visibilidade ao processo de testes dentro das empresas, especialmente naquelas que são desenvolvidas aplicações web e ainda não compreendem o papel estratégico dos testes automatizados. Espera-se, com isso, contribuir para a disseminação de boas práticas de qualidade de software e para o fortalecimento da área de QA no mercado de tecnologia.

Palavras-chave: Qualidade de software, Automação Web, Ferramentas de testes, Quality-Assurance.

Lista de ilustrações

Lista de abreviaturas e siglas

QA	Quality Assurance
TI	Tecnologia da Informação
IDE	Integrated Development Environment
SM	Scrum Master
PO	Product Owner
UX	User Experience Designer
IU	User Interface Designer
TL	Tech Lead
DEV	Desenvolvedor
E2E	End-to-End
FRONT	Frontend
BACK	Backend

Sumário

1 Introdução

Nos dias atuais, a área de desenvolvimento de software concentra-se na entrega de produtos de qualidade de forma rápida e eficiente. No entanto, de pouco adianta disponibilizar soluções que não gerem valor ao usuário final, devido a erros e inconsistências na experiência de uso.

Com frequência, usuários se deparam com falhas, defeitos ou comportamentos inesperados em sites e aplicativos, independentemente do setor ao qual pertencem. Tais problemas, aliados a interfaces com baixa usabilidade ou excessivamente complexas, resultam na frustração e, muitas vezes, no abandono da jornada proposta pela aplicação. A atuação do profissional de garantia da qualidade *(QA Quality Assurance)* tem o potencial de evitar esses e outros problemas. Sua responsabilidade é promover a qualidade em todas as etapas do desenvolvimento, desde a definição dos requisitos e a elaboração das histórias de usuário até a entrega final, aplicando técnicas e estratégias de testes alinhadas ao escopo e ao objetivo do produto.(PARVEEN REX BLACK, 2018)

Ao começar os testes ainda nas fases iniciais do projeto, mesmo antes do início do desenvolvimento, possibilita-se identificar e mitigar falhas com maior agilidade e menor custo. Quanto mais cedo o processo de testes é incorporado ao ciclo de vida do software, mais simples, rápida e econômica tende a ser a correção de problemas. Essa abordagem contribui diretamente para o aumento da vida útil da aplicação e para a entrega de soluções mais confiáveis e alinhadas às expectativas dos usuários.(FILHO; RIOS, 2003)

O processo de testes deve ser levado a sério, pois um defeitos pode gerar prejuízos severos a imagem, valor e confiança de uma marca em relação aos seus usuários. Um ótimo exemplo do quão caro pode custar a ausência de testes é o de um banco digital que, após uma falha na atualização de uma funcionalidade, teve o prejuízo estimado em 9 milhões de reais. Segundo nota no jornal Estadão (GONSALVES, 2024), o erro consistiu em não validar o valor de saldo disponível, permitindo que o saque no valor de até 1000 reais fosse efetuado por seus usuários mesmo sem ter esse valor em conta. Esse erro poderia ter sido detectado usando a técnica de teste de valor limite, que consiste em análise de valor, sendo esta uma extensão da técnica de partição de equivalência, com foco nos valores extremos de um intervalo, pois erros tendem e podem ocorrer nesses pontos (BEIZER, 2003). Essa é uma ferramenta muito útil em sistemas financeiros.

Tal episódio só deixa claro o quanto a ausência de testes é preocupante, pois técnicas e tipos de testes distintos, de acordo com a necessidade da aplicação, devem ser planejados e executados em todos os níveis de desenvolvimento do produto até ser disponibilizado em produção. Essa situação exemplifica o quanto os testes são necessários,

pois, de acordo com (DIJKSTRA, 1971), o teste de programa pode ser usado para mostrar a presença de defeitos, mas nunca para mostrar sua ausência. Para os usuários do produto do banco digital que apresentou o erro em saldos e cobranças indevidas fica a insegurança quanto ao produto que gerou dúvidas sobre a confiabilidade da marca, que precisou se reposicionar, informar que tem profissionais capacitados para manter a qualidade, relatar o erro e, em paralelo, notificar que a equipe estava correndo atrás de uma solução rápida. No entanto, o custo maior está relacionado à baixa confiabilidade, visto que usuários e acionistas podem pensar que, se há erros desse porte, qual a segurança terão ao investir nessa empresa?

E não basta somente aplicar a correção do defeitos encontrado, mais uma vez o banco digital deixou escapar falhas em seu processo de desenvolvimento e qualidade, pois com a correção desse primeiro defeitos, aparentemente gerou um outro na mesma jornada, relacionada a saldo do usuário, como relata o jornal Estadão, no dia 08/11/2024: "Na manhã desta sexta-feira (8), os clientes agora relatam um novo problema nas redes sociais: a cobrança de compras antigas em duplicidade"(ROCHA, 2024).

Esses e outros exemplos podem ser encontrados com facilidade, o que só reforça a importância de implementar o processo de testes, conduzido por quem sabe aplicar as técnicas e estratégias. Assim, o QA se mostra ainda mais fundamental no desenvolvimento de software. O analista de qualidade é quem desempenha o papel de ser o agente que dissemina a qualidade entre a squad, por ter a visão de projeto mais apurada e por acompanhar o produto desde antes da etapa de desenvolvimento, idealizando um produto sólido, capaz de alcançar a melhor performance dentro do possível e do que foi desenhado por seus stakeholders. A visibilidade do profissional de qualidade vem crescendo cada vez mais, pois a atuação deste já não está mais em questão e todos sabem da importância para o produto. Com esse crescimento na área, várias comunidades com o foco em QA estão surgindo e cada dia mais se fazem necessários eventos, divulgação e conhecimento para esses analistas pensarem de maneira cíclica, como um analista de qualidade deve pensar. É essa chave que pode fazer toda a diferença no sucesso do produto.

A qualidade do software tem sido uma das principais preocupações nas práticas modernas de desenvolvimento, especialmente diante do aumento da complexidade das aplicações e da exigência por entregas mais rápidas e com menos erros. A literatura especializada destaca os testes de software como uma etapa essencial para garantir a confiabilidade, a performance e a usabilidade dos sistemas. Segundo (PRESSMAN; MAXIM, 2021), o teste de software é uma atividade sistemática que busca identificar falhas em um sistema, permitindo que os defeitos sejam corrigidos antes que o produto seja entregue ao usuário final.

Este estudo de caso pretende, principalmente, investigar a importância da automação de testes em aplicações web, utilizando as ferramentas Selenium WebDriver e

JUnit, com foco na contribuição desses testes para a garantia da qualidade de software em ambientes ágeis.

1.0.1 Motivação

O presente estudo foi motivado por uma crescente pressão sobre as equipes de desenvolvimento para realizar entregas com agilidade. No cenário atual, onde há uma corrida constante para lançar produtos no mercado antes da concorrência, o tempo destinado ao desenvolvimento é cada vez mais reduzido o que frequentemente leva à negligência das etapas de teste.

Nesse ambiente altamente competitivo, empresas de diferentes setores precisam entregar soluções de software com maior frequência e em prazos mais curtos, sem comprometer aspectos cruciais como estabilidade, desempenho e confiabilidade. No entanto, ao priorizar apenas a velocidade da entrega e relegar a qualidade a segundo plano, corre-se o risco de disponibilizar aplicações frágeis, com baixa usabilidade e alta propensão a falhas.

Diante desse cenário do mercado de tecnologia, empresas de todos os segmentos precisam entregar soluções de software com maior frequência em menor tempo, sem comprometer a estabilidade, performance e confiabilidade, tentando entregar aplicações robustas sem grande foco na qualidade. Ignorado esses passos, é difícil obter um produto com boa usabilidade, acarretando um produto sem qualidade. Esse novo modelo de mercado de urgência exige processos de desenvolvimento rápidos, iterativos e com ciclos curtos, onde não há espaço para os testes no planejamento, visando outra prioridade no processo de vida útil do produto. Desde que Myers escreveu seu livro sobre testes, várias pesquisas de melhorias vêm sendo realizadas na área de QA, por este motivo hoje é tido como senso comum que teste não pode mais ser um apêndice sem prioridade nos processos de desenvolvimento.(MYERS GLENFORD J; BADGETT, 2011)

É nesse ponto, no qual a automação de testes de software ganha protagonismo, que o QA entra na jogada para demonstrar o que de fato é qualidade de software, provocando discussões com foco em melhorias para a aplicação, com uma visão diferente de negócios e desenvolvedores. Dessa forma, os detalhes são pensados e visualizados de forma distinta, nada habitual para encontrar mais inconformidades em algo desenvolvido por outras mãos. Essa visibilidade de melhorias e a expertise de promover a qualidade é o diferencial do QA no meio do processo de desenvolvimento, visto que a automação não apenas reduz o tempo necessário para validar funcionalidades, mas proporciona maior segurança nas entregas, melhora a cobertura dos testes e contribui diretamente para a detecção precoce de defeitos.

A motivação central deste estudo está nas dificuldades que muitas equipes en-

frentam ao implementar o processo de testes, mesmo quando utilizam boas práticas de desenvolvimento. Em muitos casos, falhas críticas são descobertas em etapas avançadas, elevando o custo da correção e impactando negativamente os prazos e a qualidade percebida pelos usuários. Isso evidencia a necessidade de incorporar processos de Quality Assurance mais estruturados e integrados, desde o planejamento até a entrega final.

A automação de testes é um processo eficiente, pois, na etapa de validação de garantia de qualidade, um dos fatores levados em consideração é cobertura e diversificação dos testes executados de acordo com a aplicação. Testes no backend que verificam se a lógica do servidor está funcionando corretamente e testes no frontend que garantem que a interface do usuário funciona como esperado também podem ser executados de maneira automatizada, mitigando erros e melhorando a usabilidade, segurança e confiabilidade da entrega, além desses, têm-se ferramentas como Selenium WebDriver dentre tantas outras que são open-source.(JASON, 2004) São essas comunidades como Selenium Brasil, 4ALL Testes e mais outras comunidades com conteúdo de apoio bem estruturado, que possibilitam otimizar o dia a dia da equipe de testes, permitindo uma curva de aprendizado rápida. A automação, de acordo com algumas ferramentas, emite relatórios com imagens e detalhes dos defeitos encontrados, facilitando o mapeamento desses para o desenvolvedor aplicar a correção de forma mais assertiva.

Os principais tipos e estratégias de testes vão ser explorados nesse estudo para viabilizar a compreensão das atividades de um QA e a sua importante contribuição na atuação nos novos moldes de equipes ágeis de desenvolvimento de software. Essa ideia, no contexto web, torna-se ainda mais relevante devido à diversidade de navegadores e dispositivos que permitem o acesso a essas aplicações; nesse caso, a automação tem um papel fundamental no processo, pois testes de ponta a ponta ou popularmente conhecido como *(E2E end-to-end)* são executados em questão de segundos, mesmo em grandes jornadas, em que um tester executando testes manuais poderia demorar horas para chegar a mesma conclusão.

Assim, será apresentado o ciclo de vida do software, destacando as etapas de testes, os estilos e os níveis de aplicação de testes automatizados. Também serão abordados os benefícios desses testes para garantir a qualidade da aplicação ao longo de todo o projeto, especialmente em projetos que adotam metodologias ágeis, onde os testes são realizados a cada fase de entrega para reduzir erros conforme se aproxima a conclusão. Vale lembrar que os testes indicam a presença, e não a ausência, de falhas e defeitos (MYERS GLENFORD J; BADGETT, 2011).

1.1 Objetivos Geral e Específicos

O presente estudo tem como objetivo analisar o papel estratégico do analista de qualidade no ciclo de vida do desenvolvimento ágil de software, especialmente sob a metodologia Scrum e Kanban, que permite discutir a relevância do analista de qualidade como agente ativo na entrega contínua de valor. Na realidade, observa-se uma fusão entre as duas metodologias ágeis: Scrumban, uma fusão da estrutura do Scrum, que inclui planejamento e reuniões frequentes, com a adaptabilidade do Kanban, que utiliza quadros visuais e um limite para tarefas em execução. Assim, o foco é analisar essa metodologia unificadora, com ênfase na flexibilidade e adaptabilidade do processo de desenvolvimento (SEMERANO; OLIVEIRA, 2024).

1.1.1 Objetivos Específicos

A seguir, explicitam-se os objetivos específicos, a começar pelo objetivo de avaliar os benefícios e desafios da automação de testes web em projetos ágeis, considerando seus impactos no desempenho das equipes, na entrega contínua de valor e na sustentabilidade do processo de desenvolvimento.

O segundo objetivo visa apresentar um estudo de caso prático que demonstre a implementação de um processo de automação de testes em uma empresa de desenvolvimento de software, evidenciando os resultados obtidos, as dificuldades enfrentadas e as lições aprendidas ao longo da aplicação. Ainda, busca-se analisar como a inserção de testes automatizados desde as fases iniciais do desenvolvimento pode melhorar a detecção de falhas, reduzir o retrabalho e fortalecer a cultura de qualidade nas equipes ágeis.

Por fim, tem-se por objetivo investigar a contribuição das metodologias ágeis Scrum e Kanban (ou, sua combinação, Scrumban) para a integração eficaz da automação de testes web ao processo de desenvolvimento, destacando como essas abordagens favorecem a adaptação contínua, a colaboração entre os times e a melhoria incremental do produto (SOUZA ANDERSON FERREIRA ALVES, 2024).

1.1.2 Método de Pesquisa

Este trabalho adota a metodologia de estudo de caso com caráter exploratório, por ser uma abordagem eficaz para investigar fenômenos contemporâneos em seu contexto real. Essa escolha metodológica permitiu a realização de conversas e entrevistas informais com profissionais envolvidos no processo de desenvolvimento de software, proporcionando uma compreensão mais ampla e detalhada da realidade observada. A opção por essa abordagem se justifica pela necessidade de analisar, de forma contextualizada, a ausência de práticas consolidadas de garantia da qualidade em equipes de desenvolvimento, bem

como identificar os tipos de testes aplicados e a percepção sobre a importância do QA no ciclo de desenvolvimento de software.

O estudo se baseia em materiais publicados em livros, periódicos científicos, revistas especializadas e bases acadêmicas, como o Portal de Periódicos da CAPES, Google Acadêmico, entre outras fontes relevantes. A partir desse referencial, busca-se destacar a importância dos testes de software, com ênfase na automação voltada para aplicações web. Portanto, o estudo de caso possibilita compreender na prática como ferramentas de automação, como o Selenium WebDriver, podem ser inseridas na rotina de trabalho do QA. Além disso, permite analisar os impactos da automação na agilidade e eficiência da execução de testes em larga escala, como os testes de regressão, contribuindo para a melhoria contínua do processo de desenvolvimento de software.

1.2 Organização do trabalho

O estudo será desenvolvido em mais quatro capítulos, para além desta introdução, referenciados conforme a descrição abaixo:

Capitulo 2: referencial teórico, com base relevante para a compreensão do estudo, incluindo trabalhos correlatos para fundamentar concretamente o tema abordado.

Capitulo 3: como o foco principal desse estudo de caso é a automação de testes em aplicações web, será apresentada a estrutura dos testes e a sua implementação, de acordo com o ciclo de vida dos testes. Esta seção explica a combinação das ferramentas Selenium WebDriver e JUnit.

Capítulo 4: desenvolvimento da proposta de implantação do processo de testes de software, utilizando metodologia ágil e os principais tipos de testes aplicados a sistemas web.

Capítulo 5: conclusão do trabalho, a fim de dar visibilidade para a importância de propagar a qualidade de software, destacando as considerações finais, a validação do estudo de caso e a contribuição para as empresas, contando com recomendações e sugestões para possíveis implementações futuras.

1.2.1 Considerações finais

O presente estudo aborda a importância da garantia da qualidade no desenvolvimento ágil de software, com ênfase na automação de testes em aplicações web por meio das ferramentas Selenium WebDriver e JUnit. Diante da pressão por entregas rápidas e frequentes, muitas equipes negligenciam os testes, o que resulta em falhas graves e prejuízos financeiros, como ilustrado no caso de um banco digital. O papel do analista de qualidade se mostra essencial ao promover a excelência desde as fases iniciais do ciclo de

vida do software, contribuindo para a detecção precoce de defeitos, a redução de retrabalho e a entrega de soluções mais confiáveis e alinhadas às expectativas dos usuários. Utilizando a metodologia de estudo de caso com caráter exploratório, a pesquisa busca demonstrar os impactos positivos da automação de testes em ambientes ágeis, especialmente com a adoção de metodologias como Scrum, Kanban e Scrumban. Além disso, destaca-se a relevância das comunidades e ferramentas open-source para a evolução da prática de QA, bem como a necessidade de consolidar processos estruturados de testes como parte integrante do desenvolvimento moderno.

2 Revisão Bibliográfica

O capítulo de revisão bibliográfica tem como objetivo garantir a base do estudo de forma sólida, apresentando conceitos e princípios de testes de software para automação web. Ao longo deste estudo, serão abordados tópicos técnicos para facilitar a compreensão do passo a passo na criação da automação, a fim de demonstrar os ganhos com as boas práticas empregadas na qualidade de software.

2.1 Testes de Software

2.1.1 O que é teste de software?

O teste de software é uma atividade crítica para a validação e verificação da qualidade, permitindo detectar falhas e assegurar que os requisitos do sistema sejam efetivamente atendidos. O processo de analisar um artefato de software se dá ao ser executada uma sequência de passos para detectar as diferenças entre o que foi planejado vs. o que foi desenvolvido (DELAMARO; JINO; MALDONADO, 2013).

Testar um software é o processo de executar um sistema com a intenção de encontrar erros. Para entender essa prática, é importante compreender os conceitos básicos (PRESSMAN; MAXIM, 2021):

- Erro é definido como uma ação humana que pode gerar um resultado incorreto considerando o que foi planejado para o software, frequentemente causada por um engano na compreensão ou implementação.
- Defeito é definido como uma imperfeição no código ou artefato do software, causada por um erro, que pode gerar falhas no comportamento do sistema.
- Falha ocorre quando um defeito é executado e o sistema se comporta de maneira inesperada, resultando em uma saída incorreta ou falha de funcionamento.

Nesta fase inicial, é necessário compreender também a diferença entre caso de teste e cenário de teste.

2.1.2 Cenário de Teste

É a descrição de uma situação específica que será validada para garantir que o sistema funcione como o esperado. Um cenário de teste geralmente inclui uma sequência

de passos, ações de entrada e resultados esperados, visando validar o comportamento do sistema diante de uma determinada condição definida na escrita do cenário.

Um exemplo de cenário presente na grande maioria das aplicações é o login, que representa a primeira ação para se ter acesso ao produto. Esse cenário pode ser descrito informando, passo a passo, as ações realizadas e os resultados esperados.

Cenário: Login com credenciais válidas - Dado que o usuário está na tela de login, quando ele informa usuário e senha corretos e, então, faz o clique no botão logar, ele deve ser redirecionado para a página principal.

Esse é um cenário positivo ou popularmente dito como cenário feliz, pois o resultado esperado dele é um login realizado com sucesso. Poderia haver N cenários apenas na funcionalidade de login, como cenário login com credenciais inválidas ou falha na conexão do usuário; é importante validar qual a mensagem de erro o usuário vai receber no caso de uma indisponibilidade de rede ou do serviço.

Aqui, na descrição do cenário, foi usado o BDD (Behavior-Driven Development - Desenvolvimento Guiado por Comportamento) como uma abordagem para unir o lado técnico (desenvolvedores e testadores) e o lado de negócios composto por analistas (responsáveis por levantar e detalhar requisitos do sistema), POs Product Owners (papel do Scrum responsável por priorizar e definir o que deve ser desenvolvido para gerar valor ao negócio) e stakeholders (partes interessadas no projeto, como clientes, gestores e usuários finais). O BDD propõe que os requisitos e comportamentos esperados do sistema sejam descritos em uma linguagem natural e compreensível para todos os envolvidos, independentemente de conhecimento técnico. Trata-se de uma técnica de desenvolvimento ágil que orienta a construção do software com base no comportamento esperado do sistema a partir da perspectiva do usuário, utilizando uma sintaxe estruturada composta por três partes: Dado o contexto inicial, Quando a ação executada, e Então o resultado esperado. Essa abordagem melhora a comunicação entre áreas, alinha expectativas e reduz falhas de entendimento sobre os requisitos.

Essa linguagem comum facilita a comunicação e o alinhamento de expectativas entre as partes, reduzindo ruídos de interpretação e garantindo que o software desenvolvido atenda realmente às necessidades do negócio. Além disso, ao escrever os cenários com base em comportamentos observáveis, o BDD ajuda a documentar claramente o que o sistema deve fazer, promovendo uma cultura de colaboração contínua e foco no valor entregue ao usuário final.

2.1.3 Caso de Uso

É uma descrição mais ampla que representa como um ator (usuário ou sistema externo) interage com o sistema para atingir um objetivo específico. Casos de uso fazem

parte da modelagem de requisitos e ajudam a entender o que o sistema precisa fazer do ponto de vista do usuário (ALMEIDA et al., 2008).

Exemplo: Caso de Uso: Realizar Login.

Ator: Usuário.

Objetivo: Acessar a área restrita do sistema.

Fluxo Principal: O usuário acessa a tela de login; insere usuário e senha; o sistema valida as informações; redireciona para a página principal.

Fluxos alternativos: Usuário esquece a senha (entra no fluxo de recuperação), senha inválida, indisponibilidade de rede etc.

Casos de uso são mais completos, por detalharem fluxos principais, alternativos, pré-condições e pós-condições. Abaixo, uma maneira simples de visualizar as diferenças entre caso de uso e cenários de testes.

ITEM	CENÁRIO DE TESTE	CASO DE USO
Objetivo	Validar um comportamento específico	Descrever a interação usuário-sistema
Foco	Testes	Requisitos e funcionalidades
Linguagem	Mais prática e focada no teste	Mais analítica e de especificação
Exemplo	Testar login com sucesso	Realizar login no sistema

Figura 1 – Comparativo entre caso de uso e cenário de testes

Fonte: (RIOS, 2010)

2.1.4 Behavior Driven Development BDD

O BDD (Behavior-Driven Development- Desenvolvimento Guiado por Comportamento) é uma prática criada por Daniel Terhorst North, no início dos anos 2000, para melhorar o Test Driven Development. A ideia era ajudar programadores em equipes ágeis a escreverem testes e código focados no comportamento esperado do software, evitando mal entendidos. Com o tempo, o BDD evoluiu para também incluir análise e testes de aceitação automatizados (SMART; MOLAK, 2023).

O formato (Dado/Quando/Então) foi desenvolvido para descrever cenários de forma clara e executável. O modelo foi influenciado pelo conceito de linguagem ubíqua, do livro "Domain-Driven Design", de Eric Evans.(TERHORST-NORTH, 2024) O BDD

é válido como uma documentação viva. Os cenários descritos nesse modelo servem como uma documentação que sempre reflete o estado real do sistema, e o mesmo pode ser apresentado com documentação para novos membros da equipe, a fim de conhecer o produto e entender a jornada do usuário. No estudo, não será utilizado o BDD, tendo em vista que ele foi apresentado apenas como opção para a fase de planejamento de testes, onde é amplamente utilizado por negócios durante a escrita das atividades que passam por testes.

2.2 Principais tipos de testes

O processo de teste de software compreende diversas abordagens que visam verificar a conformidade de um sistema com os requisitos estabelecidos. A escolha do tipo de teste depende do objetivo da verificação ou validação, do estágio do ciclo de vida do software e do nível de criticidade da aplicação.A seguir na figura 2 é possível entender o processo que acontece durante o desenvolvimento de software, os testes se dividem em duas abordagens fundamentais: verificação e validação. A verificação ocorre nas fases iniciais e tem como objetivo garantir que o produto está sendo construído de acordo com os requisitos técnicos, padrões e documentações definidos ou seja, foca em "construir o produto certo". Já a validação é realizada nas etapas finais e busca confirmar se o produto atende às necessidades reais dos usuários, validando funcionalidades em ambiente de execução, ou seja, se estamos "construindo o produto corretamente". Abaixo, destacam-se os principais tipos de testes relacionados a cada uma dessas abordagens.

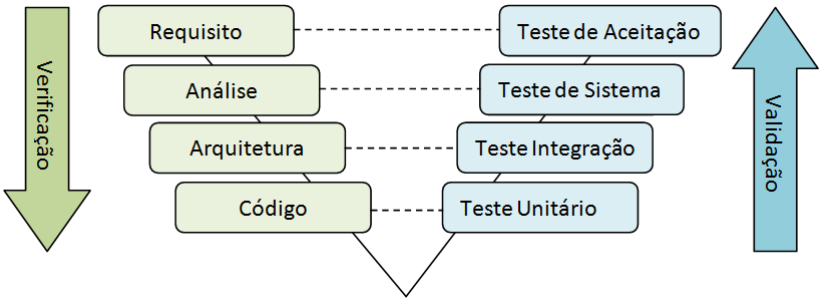

Figura 2 – Tipos de testes

Fonte: (RIOS, 2010)

A seguir, são apresentados os principais tipos de testes:

Esse tipo de teste tem como foco verificar a menor parte testável do software, funções ou métodos. É geralmente realizado por desenvolvedores durante a fase de codificação. Visa garantir que cada unidade funcione corretamente de forma isolada e auxilia na identificação em estágio inicial. Em resumo, consiste em validar dados válidos e inválidos via I/O (entrada/saída) (MYERS GLENFORD J; BADGETT, 2011).

2.2.1 Teste de Integração

Os testes de integração têm como objetivo avaliar a comunicação entre diferentes módulos ou unidades do sistema. Esse tipo de teste visa detectar falhas na interação entre componentes, como erros de interface e inconsistências nos dados compartilhados (BEIZER, 2003).

2.2.2 Teste de Sistema

Envolve a verificação do sistema em sua totalidade, validando se ele atende aos requisitos funcionais e não funcionais. Engloba testes de performance, segurança, usabilidade e compatibilidade, como principais estratégias de teste (FILHO, 2003).

2.2.3 Teste de Aceitação

É o teste realizado para validar o sistema com base nas expectativas do cliente ou usuário final. Geralmente é feito no final do ciclo de desenvolvimento, antes da entrega final, podendo ser conduzido com base em critérios de aceitação definidos previamente (LEITE; VIANA et al., 2025).

2.2.4 Teste de Regressão

Prática realizada no fechamento do "pacote da versão". Ao finalizar o desenvolvimento da "user story", é executado um teste cobrindo todos os cenários da aplicação para garantir que a nova implementação, alterações ou correções no código não introduzam novos defeitos, ou impactem negativamente funcionalidades já existentes. Esses testes são geralmente automatizados.

Ao iniciar uma sprint, há um pacote de atividades planejadas, que será desenvolvido de acordo com o capacite da equipe. A atividade passa por desenvolvimento, enquanto isso, o QA está descrevendo os cenários de testes e levantando as massas de dados para iniciar a execução; ao finalizar, emite um relatório de testes para ficar evidenciado a jornada que está programada para ser entregue. A última etapa é o fechamento do pacote, que gera uma única versão com todas as atividades desenvolvidas de forma fracionada por versões de módulos da aplicação, para um novo ciclo de testes que valida que a nova implementação não gerou novos defeitos em partes da jornada que não sofreram alterações. Os testes regressivos são executados com frequência ao longo das entregas contínuas (NETO; CLAUDIO, 2007).

2.2.5 Testes manuais

Testes manuais de software são um tipo de verificação nos quais o analista ou testador executam os testes sem o uso de scripts automatizados, ou ferramentas. O próprio profissional interage de forma direta com o sistema, executando um conjunto de cenários definidos, manualmente. Ele simplesmente acessa a aplicação e faz as validações conforme os cenários desenhados para verificar se ela funciona conforme o esperado. Esta é uma técnica muito utilizada devido à facilidade de execução, pois só requer o aprendizado do produto; no entanto, não tem escalabilidade, por exemplo, uma aplicação com mais de 100 cenários levará muito tempo para concluir a execução do plano de testes.

Em testes realizados por humanos e sem ferramentas de automação, é comum envolver testes exploratórios sem script ou baseados em roteiro, considerando o conhecimento dos analistas e/ou testador que está executando os testes. Os testes manuais também são úteis para validar aspectos como usabilidade, aparência *(UI) User Interface Designer* e comportamento em situações menos previsíveis, sendo comum em fases iniciais do desenvolvimento ou quando o sistema ainda muda com frequência.

2.2.6 Testes automatizados

Os testes automatizados são muito importantes no modelo de desenvolvimento de software atual. Eles possibilitam o processo mais rápido de testes, confiável e organizado, já que permitem repetir os testes sempre que necessário em fração de segundos, a depender do número de cenários a serem executados. Em termos de escalabilidade, rapidez e eficiência, o teste automatizado, se comparado ao teste manual, tem uma performance melhor.

Ao usar testes automatizados, a tendência é que haja a redução dos custos e do tempo dos projetos, deixando o desenvolvimento mais fluido. Além disso, essa prática facilita a identificação de erros logo no começo, considerando que os testes automatizados foram implementados, a princípio, em paralelo com a fase de desenvolvimento. Ou seja, os testes de unidade podem ser automatizados por desenvolvedores antes de disponibilizar uma versão para testes executados por QA de forma mais detalhada.

O analista de qualidade pode aproveitar para validar os testes unitários com um tipo de teste que verifica se cenários críticos atendem às expectativas, como o teste mutante, que pode confirmar a qualidade dos testes já implementados, uma vez que sua aplicação sofre alterações constantes e, em alguns momentos do projeto, os desenvolvedores não mantêm os testes unitários existentes. Essa atitude pode gerar uma falsa sensação da cobertura de testes, pois os cenários que não têm manutenção podem somente atrapalhar a validação inicial. Portanto, o analista de qualidade precisa reforçar a importância do teste unitário implementado por desenvolvedores e acompanhar essa tarefa, essencial em equipes que usam métodos ágeis e entregam atualizações o tempo todo (MYERS

GLENFORD J; BADGETT, 2011).

A popularização desse tipo de teste está atrelada às metodologias ágeis, pois a partir dos anos 2000, o Scrum passou a ser a metodologia mais utilizada em projetos de software, e desde então as equipes exigem entregas contínuas e feedback rápido. Nesse modelo, a automação começou a ser considerada essencial, pois com o desenvolvimento incremental, o número de ciclos de testes é maior, e o foco da automação é garantir qualidade nas entregas frequentes.(BECK et al., 2001)

A automação de testes de software tem se mostrado cada vez mais estratégica para aumentar a produtividade e a confiabilidade dos sistemas em ambientes ágeis. (PINHEIRO; VALENTIM; VINCENZI, 2015) realizou um estudo comparativo entre a execução de testes manuais e testes de aceitação automatizados em uma aplicação web. Em tal estudo, ficou claro o quanto a automação traz ganhos para o andamento do ciclo de testes, o que reduz o tempo de execução e permite maior reutilização de casos de usos, fatores críticos para a manutenção da qualidade diante da crescente complexidade das aplicações web. O estudo evidencia que a automação, apesar do esforço inicial de implementação, proporciona reduções expressivas de custo e esforço manual a médio e longo prazo, especialmente em testes de regressão. Além disso, considera-se que a automação melhora a rastreabilidade e a documentação dos testes, características essenciais em projetos modernos.

2.3 Garantia de qualidade (QA)

De acordo com (RIOS, 2010), um bom processo de testes está atrelado a definição precisa dos objetivos idealizados, para que seja possível visualizar os entregáveis de qualidade. Garantia de Qualidade - QA: a garantia da qualidade é preventiva, porque ela se refere ao conjunto de atividades planejadas e desenhadas dentro do sistema de qualidade, com o objetivo de gerar confiança de que os requisitos de qualidade serão atendidos. Essas atividades são preventivas e focam na melhoria dos processos de desenvolvimento, visando evitar que defeitos ocorram no produto final.

Controle de Qualidade - QC: o controle de qualidade segue o método de detecção, ao envolver a execução de atividades e testes para identificar defeitos no produto já desenvolvido, o que foca em execução de técnicas e atividades operacionais utilizadas para cumprir os requisitos de qualidade do produto. Essas atividades são corretivas e têm o foco na identificação e correção de defeitos no produto final, garantindo que esse atenda aos padrões de qualidade estabelecidos.

É comum existir dúvida quanto ao que é processo de controle de qualidade e o que pode ser atribuído como responsabilidades de garantia de qualidade, visto que esses são termos que têm definições e usabilidade distintas,o método de qualidade assim como na

Aspecto	Garantia de Qualidade (QA)	Controle de Qualidade (QC)
Foco	Processos e prevenção	Produto final e correção
Tipo	Proativo	Reativo
Envolve	Planejamento, norma, melhoria contínua	Testes e inspeções
Objetivo	Evitar defeitos	Encontrar defeitos

Figura 3 – Garantia de qualidade x Controle de qualidade

Fonte: (RIOS, 2010)

figura 3 pode ser dividido em duas sessões. O controle de qualidade (QC) é uma parte no processo da garantia de que o produto de software atenda aos padrões previamente definidos e aos requisitos funcionais e não funcionais estabelecidos. Enquanto isso, o QA, sendo a garantia de qualidade, tem o objetivo de prevenir as falhas no produto, estabelecendo processos, validando e identificando erros e defeitos na tentativa de avaliar a qualidade. Para facilitar a compreensão, é necessário adotar esse tipo de pensamento: enquanto o QA é proativo e orientado a processos, buscando prevenir problemas de qualidade; o QC é reativo e orientado a produtos, focando na detecção e correção de defeitos (ADERSON.RIOS, 2007).

2.4 Metodologias ágeis

Metodologias ágeis surgiram a partir da necessidade de desenvolver software de maneira mais leve, rápida e centrada nas pessoas. A definição prática do desenvolvimento ágil foi estabelecida com a criação do Manifesto Ágil, em 2000, o qual reúne valores e princípios voltados à colaboração, entrega contínua e foco no cliente. Essas metodologias são baseadas no desenvolvimento iterativo, com equipes auto-organizadas e multifuncionais, incentivando inspeção frequente, adaptação, alinhamento com os objetivos do negócio e aplicação de boas práticas de engenharia para garantir entregas rápidas e com qualidade (WILLI, 2014).

2.4.1 Scrum

Scrum é um modelo ágil de gestão de projetos cujo objetivo é tornar as equipes de desenvolvimento mais produtivas, promovendo alta performance e entregas constantes. Trata-se de uma estrutura voltada para o desenvolvimento, entrega e sustentação de produtos complexos, baseada em ciclos iterativos e incrementais. Neste estudo, o foco recai sobre essa metodologia devido à sua ampla aceitação e aplicação no setor de tecnologia. O

Scrum se popularizou significativamente no Brasil, sendo adotado por diversas empresas de software em razão de sua eficácia na organização de equipes e no aumento da eficiência dos processos de desenvolvimento.

No que diz respeito ao desenvolvimento de software, o Scrum foi adaptado para aprimorar as etapas e permitir o desenvolvimento de software com passos predeterminados, incluindo papéis e etapas bem definidos, como a definição das cerimônias para a manutenção do produto, facilitando a comunicação entre todos os envolvidos no projeto e minimizando falhas na comunicação (WAZLAWICK, 2019).

2.4.2 Papéis no Scrum

No contexto do Scrum, um dos frameworks ágeis mais utilizados no desenvolvimento de software e em outras áreas, o termo papéis se refere às funções bem definidas que cada integrante do time assume para garantir o funcionamento eficaz do processo ágil. Diferente de modelos tradicionais de gestão, onde as responsabilidades são muitas vezes distribuídas de forma hierárquica ou genérica, no Scrum cada papel tem objetivos, responsabilidades e limites de atuação claramente estabelecidos, abaixo são descritos: (DIMES, 2014)

2.4.2.1 PO - Product Backlog

O PO é o dono do produto, responsável por garantir que o time entregue valor ao negócio, gerenciando o Product Backlog (lista de funcionalidades e requisitos), definindo quais são as prioridades com base no valor para o cliente e representando os interesses do cliente e stakeholders. Dentro do possível, o PO está sempre disponível para tirar dúvidas do time sobre os requisitos. Um exemplo: imagine que o QA encontrou um defeitos no final da release e emitiu um relatório de testes não favorável em relação à subida da funcionalidade, com um erro não impeditivo na jornada. De um lado, têm-se os desenvolvedores que são a favor de não represar código e realizar a subida para produção, disponibilizando assim a funcionalidade com um erro; enquanto os QAs são contra seguir dessa forma. Aqui entra o PO, demonstrando para ambas as partes qual será o real valor, entregue ao usuário, de uma funcionalidade com um erro não impeditivo na jornada, e trazendo a definição de seguir ou não adiante, apresentando um plano de correção de forma prioritária para permitir, em alguns casos, de fato entregar algo para um número restrito de usuários em paralelo da correção do erro.

2.4.2.2 Scrum Master

É o responsável por garantir que o time siga as práticas e ritos da metodologia Scrum corretamente no dia a dia, facilitando a comunicação. Quando a equipe confia em ter a comunicação com os demais membros e equipes centralizadas no SM Scrum Master,

isso é um ponto positivo, pois o mesmo tem a função de remover impedimentos que atrapalhem o time a desenvolver suas respectivas atividades.Um bom exemplo prático de uma parte da atuação do Scrum Master: acontece no segundo dia de uma Sprint. Durante a daily Scrum, um dos desenvolvedores relata estar com dificuldades para acessar uma API (Interface de Programação de Aplicações). Uma API é um conjunto de regras, protocolos e definições que permite a comunicação entre diferentes sistemas ou componentes de software, funcionando como uma ponte que conecta aplicações distintas de forma segura e padronizada. No caso citado, trata-se de uma API externa, fundamental para a implementação de uma funcionalidade planejada para essa Sprint. Percebendo o impedimento, o Scrum Master atua imediatamente, entrando em contato com a equipe responsável pela API para entender a origem da falha e obter uma estimativa de resolução. Paralelamente, colabora com o desenvolvedor para reorganizar suas tarefas, redirecionando seu esforço para outra funcionalidade do backlog, otimizando assim sua produtividade enquanto o problema persiste.

Além disso, o Scrum Master promove uma reunião entre os times envolvidos para esclarecer o impacto do bloqueio na Sprint, alinhar expectativas e tentar antecipar uma solução. Essa atuação proativa garante que todos os envolvidos estejam cientes da prioridade do problema e contribui para que o time mantenha o foco nas entregas, mesmo diante de obstáculos, assegurando a fluidez do processo e a continuidade do progresso dentro da Sprint.

2.4.2.3 - Time de Desenvolvimento

O time de desenvolvimento é composto por Dev's, QAs, UX/UI e TL. Esses profissionais são responsáveis pela entrega técnica e funcional de um produto ao final de cada sprint. Essa entrega é representada por um incremento do produto, ou seja, uma parte da aplicação que esteja pronta para ser disponibilizada ao cliente ou usuário final. Para isso, o time deve garantir que esse incremento atenda aos critérios definidos na atividade, previamente acordada pela equipe durante as cerimônias ágeis e sessões de planejamento. Essa definição inclui requisitos como: o código estar testado, integrado, revisado e validado de acordo com os padrões de qualidade.

Esse time é geralmente composto por diferentes perfis profissionais que colaboram entre si para garantir a qualidade, usabilidade e entrega contínua do produto.

2.4.2.4 DEV - Desenvolvedor

O desenvolvedor de software é o profissional que transforma requisitos em soluções tecnológicas, criando e mantendo sistemas web, mobile ou desktop. Suas principais atividades envolvem analisar demandas, escrever e testar códigos, corrigir erros, integrar

funcionalidades, documentar o projeto e colaborar em reuniões ágeis para garantir entregas com qualidade, desempenho e segurança.

2.4.2.5 QA - Quality Assurance

É o profissional responsável por garantir a qualidade do software. Atua desde a definição dos critérios de aceitação até a execução de testes (manuais e/ou automatizados), prevenindo defeitos, validando funcionalidades e promovendo a melhoria contínua do processo de desenvolvimento.

2.4.2.6 UX - User Experience Designer

Responsável por projetar a experiência do usuário com base em estudos de comportamento, usabilidade e acessibilidade. Seu foco é garantir que a aplicação seja intuitiva, agradável e eficaz na resolução das necessidades do usuário.

2.4.2.7 UI - User Interface Designer

Atua na parte visual do produto, sendo responsável pela interface do usuário. Cria elementos gráficos, define a identidade visual e garante a consistência estética da aplicação, em alinhamento com as boas práticas de design e com os padrões de UX.

2.4.2.8 TL - Tech Lead

O Tech Lead, ou líder técnico, é o profissional que orienta tecnicamente o time de desenvolvimento. Ele atua na tomada de decisões arquiteturais, revisão de código, padronização técnica e na mediação entre as demandas do time e os objetivos do projeto, garantindo que a solução técnica seja viável e sustentável.

2.4.3 Etapas do Scrum, eventos

2.4.3.1 Sprint

Período fixo (geralmente de 1 a 4 semanas), no qual um incremento do produto é desenvolvido. Esse tempo pode ser definido durante a etapa inicial do planejamento. A literatura informa um período de 1 a 4 semanas, mas, em grande maioria, as equipes definem um período de 3 semanas para planejar o backlog, desenvolver, testar, homologar, entregar e disponibilizar em ambiente produtivo aquela fração do produto.(WAZLAWICK, 2019)

2.4.3.2 Sprint Planning

Reunião no início da Sprint, onde o time define o que será feito e como será feito. Nessa cerimônia, ocorre a definição e alinhamento em torno do escopo e da meta da sprint.

O PO apresenta os itens priorizados, isto é, as "user story"escritas muitas vezes com a notação do BDD. O time analisa e escolhe as que acredita que consegue entregar com base na sua capacidade.

2.4.3.3 Daily Scrum (Daily Meeting)

Reunião diária de 15 minutos, onde o time compartilha o que fez, o que fará e se há impedimentos. Caso exista algum impedimento, o agilista irá entender melhor o assunto no pós-daily, junto ao analista, e criar um plano de ação para resolver o impedimento a fim de liberar o analista e o mesmo conseguir voltar a tocar a atividade.

2.4.3.4 Sprint Review

Reunião de demonstração no fim da Sprint, em que o time apresenta o que foi entregue ao PO e stakeholders. Nessa cerimônia, todas as frentes apresentam o que foi possível entregar dentro do que estava planejado (UX, desenvolvedores e QA's). Quando acontece de "escorregar atividade", isto é, caso haja algum impedimento externo que impactou na conclusão de uma atividade, ele será priorizado na sprint seguinte e uma justificativa é apresentada junto ao tech lead para que todo o time fique ciente sobre o fato.

2.4.3.5 Sprint Retrospective

Reunião de reflexão e melhoria contínua. A equipe identifica o que funcionou bem e o que pode melhorar. Na retrospectiva, é possível utilizar ferramentas para medir os pontos que foram positivos e o que tem oportunidade para aplicar melhorias. O objetivo dessa cerimônia é ter um "termômetro"de como o time está e já sair com um "to-do"que deve ser priorizado na próxima sprint.

2.4.4 Artefatos do Scrum

No universo ágil, os artefatos do Scrum emergem como ferramentas cruciais que impulsionam a transparência, o alinhamento e o desenvolvimento contínuo de projetos. Criados e aprimorados durante a jornada, esses instrumentos personificam os princípios do Manifesto Ágil: colaboração próxima, adaptabilidade e entrega consistente de soluções funcionais.(WAZLAWICK, 2019)

Os protagonistas desse ecossistema são:

2.4.4.1 Product Backlog (Backlog do Produto)

Trata-se de uma lista priorizada e em constante evolução de todas as funcionalidades, requisitos, melhorias e correções que precisam ser implementadas no produto.

O Product Backlog é gerenciado pelo Product Owner e reflete tudo o que agrega valor ao negócio. Ele é dinâmico e deve ser continuamente refinado (processo conhecido como backlog grooming) para garantir que os itens estejam bem definidos e prontos para serem trabalhados nas próximas sprints.

2.4.4.2 Sprint Backlog (Backlog da Sprint)

É o conjunto de itens selecionados do Product Backlog que serão desenvolvidos durante uma sprint. Junto a esses itens, o Sprint Backlog inclui um plano detalhado de como o time pretende alcançar a entrega do incremento planejado. Esse artefato promove a auto-organização da equipe, ao permitir que os membros definam e acompanhem seu próprio progresso em direção ao objetivo da sprint.

2.4.4.3 Incremento

O incremento representa a soma de todos os itens do Product Backlog concluídos durante uma sprint, somado aos incrementos das sprints anteriores. Para estar apto à entrega, esse incremento precisa atender à Definition of Done, ou seja, deve estar completamente desenvolvido, testado, documentado e pronto para ser disponibilizado ao cliente, seguindo os critérios de qualidade estabelecidos pela equipe. Esse artefato reforça o princípio ágil de entregar software funcional com frequência e valor percebido pelo cliente.

2.4.4.4 Definition of Done (DoD)

A Definicao of Done é um conjunto de critérios objetivos que determinam quando um item do backlog está realmente concluído. Esses critérios podem incluir: codificação finalizada, testes automatizados ou manuais executados, validação funcional, documentação atualizada, revisão de código e integração ao ambiente de produção. A DoD promove qualidade técnica e transparência, assegurando que todas as entregas atendam a um padrão mínimo aceito por toda a equipe.

Esses artefatos, ao serem bem gerenciados, garantem que o time mantenha o foco em entregar valor contínuo, adaptando-se rapidamente às mudanças e mantendo um fluxo sustentável de desenvolvimento pilares fundamentais do Manifesto Ágil.

2.5 Testes ao longo do Ciclo de Vida de Desenvolvimento de Software

2.5.1 Processo de testes

O processo de testes de software é uma etapa fundamental dentro do ciclo de desenvolvimento de software, sendo responsável por garantir que o produto atenda aos re-

quisitos funcionais e não funcionais esperados, além de proporcionar uma boa experiência ao usuário. Ao iniciar esse processo, é essencial compreender a natureza do produto, seu propósito e, principalmente, a jornada do usuário. Esse entendimento direciona a construção de uma estratégia de testes eficaz, capaz de identificar falhas críticas, prevenir erros e agregar valor ao produto final, na figura 4 ás etapas são apresentadas.

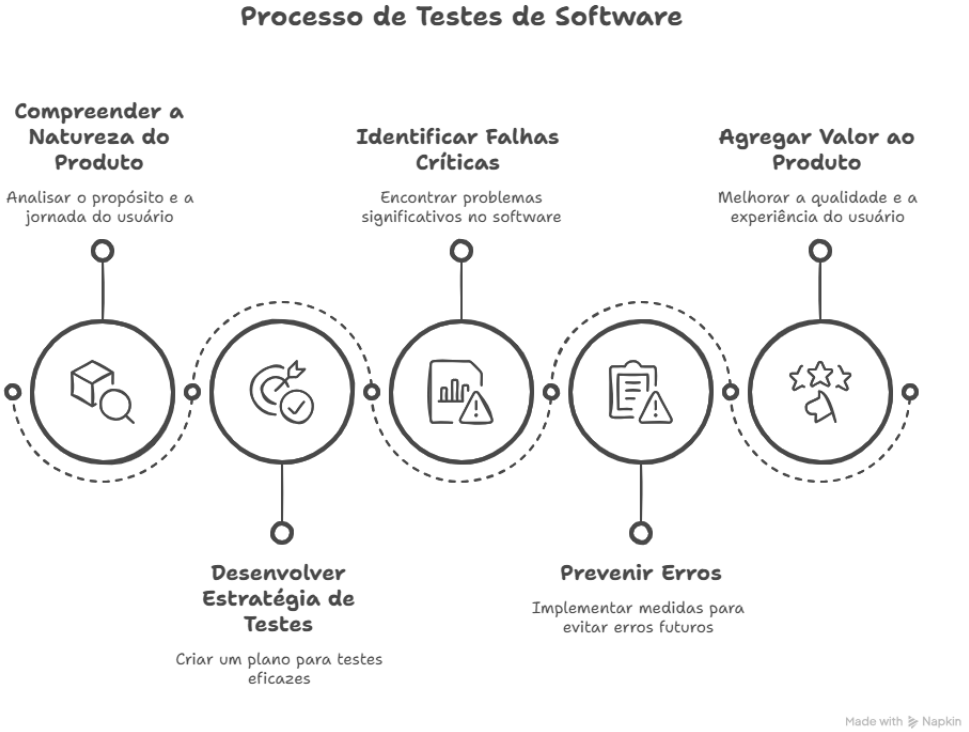

Figura 4 – Processo de Testes de Software

Fonte: (RIOS, 2010)

A seguir, são detalhadas as etapas principais do processo de testes.

2.5.1.1 Definição de Requisitos

Esta etapa inicial consiste em levantar, compreender e documentar as necessidades do cliente, usuários finais e stakeholders. Os requisitos funcionais (o que o sistema deve fazer) e não funcionais (como o sistema deve se comportar, como desempenho, segurança, etc.) servem como base para a construção dos casos e cenários de testes. Quanto mais clara for essa definição, maior a chance de sucesso dos testes.

2.5.1.2 Planejamento dos Testes

Com os requisitos definidos, inicia-se o planejamento de testes, no qual são determinadas a estratégia de testes, os objetivos, o escopo, os recursos disponíveis, os critérios

de entrada e saída, as ferramentas que serão utilizadas, bem como os riscos envolvidos. Também são definidos os papéis e responsabilidades da equipe e o cronograma de execução. Neste momento, é importante considerar o tipo de aplicação (web, mobile, API, etc.) para escolher as melhores abordagens e tecnologias.

2.5.1.3 Escrita dos Casos e Cenários de Testes

Após o planejamento, é realizada a escrita dos casos de teste de forma macro e dos cenários de testes com maior detalhamento. Nessa etapa, define-se também o escopo dos testes E2E , com foco nos fluxos críticos da aplicação, ou seja, aqueles que, se falharem, impactam diretamente a experiência do usuário.

2.5.1.4 Execução dos Testes

Com os cenários definidos e as ferramentas configuradas, inicia-se a execução dos testes, tanto manuais quanto automatizados, conforme o planejamento. Durante essa fase, os resultados são registrados, incluindo evidências e capturas de tela que comprovam o comportamento do sistema. Defeitos e falhas são reportados à equipe de desenvolvimento com clareza e detalhamento técnico, para serem corrigidos eficientemente.

2.5.1.5 Monitoramento e Controle

Ao longo da execução, é essencial monitorar o andamento dos testes por meio de indicadores como taxa de sucesso, número de defeitos identificados, tempo de execução e cobertura de testes. Esse acompanhamento permite ajustes no planejamento, identificação de gargalos e suporte à tomada de decisão de forma ágil e baseada em dados reais. Ferramentas de gestão e integração contínua são extensivamente utilizadas nesta fase para gerar relatórios automáticos.

2.5.1.6 Encerramento dos Testes

Finalizada a execução e validando que os critérios de aceitação foram cumpridos, inicia-se o encerramento formal do processo. Essa fase envolve revisar os objetivos alcançados, documentar lições aprendidas, gerar relatórios finais e coletar feedbacks dos stakeholders e usuários. Além disso, é importante garantir que os testes automatizados sejam integrados, possibilitando sua reutilização em futuras regressões ou novas funcionalidades.

2.5.1.7 O ciclo de vida de desenvolvimento de software e boas práticas de teste

É composto por diferentes fases que se organizam da seguinte maneira: quatro etapas sequenciais, que seguem uma estrutura em cascata; e duas etapas paralelas, que ocorrem de forma contínua ao longo do processo. O modelo adotado para ser apresentado

pelos autores é baseado na abordagem TMap (Test Management Approach), escolhida por conta da sua clareza didática e a facilidade de visualização de cada fase. Cada uma dessas etapas envolvem atividades específicas, gerando produtos e, assim, resultando na produção de documentos que registram e formalizam o andamento dos testes. Além disso, os autores destacam que, com base em dados do mercado, cada fase possui uma representatividade percentual estimada dentro do ciclo, o que permite uma visão mais estratégica do esforço de teste. Essa representação é formalizada no modelo 3P x 3E, que busca equilibrar planejamento, preparação e execução com estimativas, esforço e evidências, tornando-se uma ferramenta útil para o gerenciamento da qualidade em projetos de software.

As etapas podem ser compreendidas conforme a definição atribuída a cada uma abaixo:

Os procedimentos iniciais pretendem a validação dos requisitos de negócio, que serão a base para o desenvolvimento das regras que o produto a ser desenvolvido deve obedecer, visando garantir que os requisitos levantados nessa etapa sejam suficientes para um desenvolvimento sólido que não apresente ambiguidade. É de suma importância que o levantamento de requisitos seja documentado com riqueza de detalhes, admitindo também todos os recursos necessários para o desenvolvimento do produto.

Durante o planejamento, são escolhidas as estratégias, recursos e cronogramas necessários para a execução dos testes, como uma estimativa. Nessa etapa, a elaboração do plano de teste com as definições de todo o processo é realizado com base no que já foi desenhado por negócios, assim, a equipe de testes consegue criar cenários macros da aplicação antes mesmo do início do desenvolvimento, escolhendo ferramentas a ser utilizadas e quais tipos de testes aplicar. Ainda, são identificados os riscos associados ao processo de teste e estabelecidos os critérios de entrada e saída para as diferentes fases do ciclo de vida de teste.

O planejamento adequado permite que a equipe de testes alinhe suas atividades com os objetivos do projeto, assegurando que os testes sejam realizados de forma sistemática e que os resultados obtidos sejam confiáveis e úteis para a tomada de decisões.

No último "p", está a preparação, que não é menos importante, pelo contrário, é a etapa que objetiva preparar o ambiente de testes, permitindo assim a execução do passo anterior. Após escolher ferramentas e ter a dimensão do projeto para direcionar capacite dos analistas, inicia a criação do ambiente de testes. Essa fase deve ocorrer em paralelo com outras fases do produto, como especificação e execução.

As próximas etapas do ciclo são o coração de todo o processo. São os três Es: especificação, execução e entrega, de acordo com (FILHO; RIOS, 2003).

A especificação consiste em detalhar os casos e cenários de testes, com a finalidade de revisar o que foi descrito e planejado, para alcançar um nível maior de cobertura e

detalhe de cada cenário idealizado para ser executado na fase de testes.

Já a execução, que muitos julgam como a parte mais importante, é somente a ponta do iceberg, após camadas de fases para permitir a execução de fato. Ao executar um cenário de testes, consideramos conhecer o produto com o objetivo de encontrar erros, defeitos ou falhas para mitigar que esse tipo de comportamento chegue até o usuário final. Isso garante a qualidade e proporciona uma melhoria contínua, consoante o dinamismo do ciclo de desenvolvimento de software, incluindo testes. Outro ponto são as evidências e relatórios de testes, que devem ser disponibilizados ao final de cada tarefa executada

Para finalizar o ciclo de vida do processo de testes, apresenta-se a entrega, etapa na qual são apresentados os planos, as ferramentas utilizadas para uma melhor performance na execução dos testes, bem como uma estimativa baseada na capacidade do time e nas evidências documentais de todos os procedimentos executados, o que está dentro do parâmetro esperado e o que foi feito para facilitar a visualização dos resultados alcançados.

2.6 Testes Automatizados

O teste de software é um processo, isto é, uma sequência de passos para validar uma jornada, projetada para garantir que o código do computador faça o que foi projetado para fazer e, inversamente, que não faça nada não intencional (ANICHE, 2015). Iniciar uma automação de software visa ganhar escala, acompanhando entregas frequentes com garantia de qualidade. Ao executar um teste isolado do módulo e encontrar uma inconformidade na aplicação, logo é possível mapear o erro, encaminhar os dados e evidências dos testes para o desenvolvedor atuar na correção, permitindo uma tratativa pontual e mais assertiva.

Ao comparar os testes manuais com automatizados, é comum surgir uma fala errônea sobre a ideia de que os testes automatizados poderiam substituir os testes manuais, e a verdade é que ambos não são excludentes, dessa forma, um não anula o outro. Portanto, os testes automatizados não conseguem cobrir os cenários da aplicação em sua totalidade, por conta de alguns impedimentos da própria aplicação ou recursos do nativo, em casos de testes em mobile, e por esse motivo os testes manuais são de suma importância. Os dois métodos devem andar juntos no processo de testes.

A automação, aliada a projetos de desenvolvimento ágil, proporciona um ganho significativo para as organizações, uma vez que, a cada etapa, há um novo incremento que pode ser executado de forma automatizada, permitindo que um teste regressivo seja realizado quantas vezes forem necessárias para assegurar que novas entregas não causaram defeitos ou problemas em projetos já existentes.

2.7 Trabalhos Relacionados

O trabalho *Melhores Práticas e Ferramentas para Automação de Testes de Software com Selenium* 2025, investiga o uso das ferramentas de automação de testes, com ênfase no Selenium WebDriver e Selenium IDE, no contexto de desenvolvimento ágil. Os autores destacam que a automação de testes de software é essencial para garantir eficiência e qualidade em ciclos de desenvolvimento cada vez mais rápidos. Por meio de um estudo de caso prático, são apresentados passos estratégicos fundamentais para o sucesso, incluindo criação de roteiros, execução de testes, monitoramento de resultados e integração com pipelines de entrega contínua, além disso, o artigo analisa as características e benefícios das ferramentas de automação Selenium, destacando suas vantagens no contexto de testes para aplicações web. Também explora os principais desafios enfrentados pelas equipes de desenvolvimento ao implementar a automação de testes, como a manutenção de scripts e a adaptação a mudanças nas aplicações. Por fim, avalia os custos e benefícios da automação de testes, considerando o impacto financeiro e técnico no ciclo de desenvolvimento, e propõe boas práticas para a adoção eficiente da automação de testes em projetos de software. Este trabalho fornece uma visão abrangente sobre as práticas de automação de testes e oferece recomendações para sua implementação bem-sucedida no desenvolvimento de software (COSTA; SANTOS, 2025).

O artigo *Metodologias Ágeis: Explorando o Impacto do Scrum e do Kanban na Qualidade e Produtividade do Software* destaca como práticas ágeis, como Scrum e Kanban, contribuem para ciclos de desenvolvimento mais curtos, entregas incrementais e maior adaptabilidade a mudanças. No contexto de aplicações web, a automação de software, especialmente a automação de testes, se torna essencial para acompanhar a rapidez dessas entregas. A integração da automação em pipelines ágeis permite detectar falhas precocemente, garantir regressões controladas e manter a qualidade mesmo em ambientes de desenvolvimento contínuo. No presente estudo, foi possível trazer algo que está mais próximo do dia a dia, ou seja, a junção de partes do Scrum e Kanban, criando assim um ajuste de flexibilidade entre as duas metodologias (SOUZA ANDERSON FERREIRA ALVES, 2024).

Os testes manuais em aplicações web apresentam diversas limitações, como a dificuldade de repetir as execuções. Por exemplo, em um teste regressivo, existe um alto consumo de tempo, devendo-se considerar também que essa execução está suscetível a erros humanos. Para mitigar esses desafios, os autores Hanna, Amal Elsayed e Mostafa, propõem um framework de testes automatizados, voltado especificamente para aplicações web. O modelo desenvolvido visa padronizar e agilizar o processo de testes, melhorar a cobertura de cenários e facilitar o reuso dos scripts de teste. De acordo com (HANNA; ABOUTABL; MOSTAFA, 2018), esse tipo de teste se destaca por oferecer suporte à integração contínua, contribuindo para maior confiabilidade e eficiência no ciclo de desen-

volvimento, com testes planejados nesse ciclo, para obter melhores resultados durante o processo da criação do produto. A pesquisa evidencia que a automação é uma alternativa eficaz para lidar com a complexidade e a dinamicidade das aplicações modernas.

Entre os trabalhos relacionados à automação de testes em aplicações web, coloca-se em evidência o estudo de Pinheiro, Valentim e Vincenzi, que compara a execução manual de testes com a execução automatizada, utilizando as práticas de BDD e ferramentas como Selenium WebDriver e JUnit. Enquanto o trabalho apresentado neste TCC também adota Selenium e JUnit para a construção de um framework de automação, os autores propõem uma abordagem adicional no artigo, incorporando práticas de Page Object Model (POM) para garantir ainda maior manutenção do código e a integração com pipelines de integração contínua. Diferente do estudo de Pinheiro, que focou na comparação de custo/tempo, este trabalho busca ainda estruturar o framework, considerando padrões de design de testes e escalabilidade para equipes de QA. No estudo citado, demonstrou-se que, apesar do esforço inicial maior na implementação da automação, os testes automatizados apresentam vantagens a médio e longo prazo, especialmente em cenários com múltiplos ambientes e muitas repetições. Em amostras com alto volume de testes (como 1000 execuções ou mais), observou-se uma redução de até 48,8% no custo de execução quando comparado ao teste manual (PINHEIRO; VALENTIM; VINCENZI, 2015).

2.7.1 Considerações finais

Em conclusão, este capítulo de revisão bibliográfica apresentou os fundamentos essenciais para a compreensão dos testes de software voltados à automação web, abordando desde os conceitos básicos de erro, defeito e falha, até práticas modernas como BDD e automação de testes. Foram discutidas também as diferenças entre cenários de teste e casos de uso, além da importância dos diversos tipos de testes manuais e automatizados no ciclo de desenvolvimento, especialmente em ambientes ágeis. A distinção entre Garantia da Qualidade (QA) e Controle de Qualidade (QC) reforça a necessidade de processos bem definidos para prevenir e corrigir falhas. Dessa forma, o capítulo estabelece uma base teórica sólida para demonstrar como as boas práticas em testes contribuem diretamente para a melhoria da qualidade do software.

3 Ferramentas para automação de testes em sistemas web

A automação de testes de software para aplicações web representa um ganho significativo em termos de eficiência, cobertura e confiabilidade no processo de validação de sistemas. Diferentemente de testes manuais, que demandam grande esforço humano e são suscetíveis a falhas, a automação permite a execução repetitiva e consistente de casos de teste, especialmente útil em cenários de regressão, testes de compatibilidade entre navegadores e validação de fluxos complexos de navegação.

3.1 Selenium

O Selenium é uma suíte de ferramentas open-source amplamente utilizada para automação de testes em navegadores web. Desenvolvido inicialmente pela ThoughtWorks, o Selenium consolidou-se como padrão de mercado, sendo adotado tanto por startups quanto por grandes corporações em seus pipelines de integração contínua. Sua versatilidade se deve à compatibilidade com diversas linguagens de programação, como Java, Python, CSharp e JavaScript e à sua arquitetura modular, composta por componentes como Selenium IDE, Selenium WebDriver e Selenium Grid.

O Selenium WebDriver, especialmente, representa um avanço significativo em relação ao antigo Selenium RC. Ele permite o controle direto dos navegadores por meio de APIs específicas, simulando interações humanas reais com precisão e estabilidade. Essa característica reduz a incidência de falsos positivos ou falhas de sincronização, problemas comuns em ferramentas de automação baseadas em polling ou manipulação indireta na massa de testes (THOORIQOH; ANNISA; YUHANA, 2021). Adicionalmente, o Selenium é altamente integrável com ferramentas de orquestração como Jenkins, Maven, Docker e plataformas de testes em nuvem como BrowserStack e SauceLabs, o que favorece a execução paralela e distribuída dos testes, ampliando significativamente a cobertura e a eficiência das execuções. Segundo estudo de (GAROUSI; ZHI, 2013), cerca de 71% das empresas entrevistadas utilizam o Selenium como principal ferramenta de automação funcional em ambientes de produção, atestando sua eficácia e confiabilidade.

O WebDriver é o componente mais fundamental e técnico da suíte Selenium. Sua função é agir como uma camada de abstração entre a aplicação de testes e os navegadores reais, utilizando drivers nativos para executar comandos com alto grau de precisão. Cada navegador possui um driver específico como ChromeDriver, GeckoDriver, EdgeDriver que se comunica diretamente com o browser por meio de comandos JSON Wire Protocol ou

W3C WebDriver Protocol (W3C2018).(JASON, 2004)

Essa abordagem proporciona maior controle e confiabilidade na automação, permitindo, por exemplo, a manipulação de múltiplas janelas, execução de scripts JavaScript, e testes de tempo de resposta. Além disso, o WebDriver suporta mecanismosrobustos de espera explícita e implícita, fundamentais para lidar com comportamentos assíncronos das aplicações modernas baseadas em frameworks como React, Angular e Vue.js. Ao permitir a execução remota via Selenium Grid, o WebDriver também viabiliza a automação em ambientes distribuídos, permitindo que testes sejam executados simultaneamente em diferentes combinações de sistemas operacionais, navegadores e resoluções de tela, o que é crucial em estratégias de cross-browser testing (KOSCIANSKI; SOARES, 2007).

Com isso, a automação se torna um pilar essencial para garantir qualidade e velocidade em ambientes ágeis e de entrega contínua, reduzindo custos, aumentando a produtividade da equipe de QA e elevando a confiança na aplicação entregue ao usuário final. Neste estudo, será utilizado o Selenium WebDriver com JavaScript, considerando esses dois parâmetros de acordo com a grande usabilidade no mercado e fácil implementação no processo de testes para uma equipe que deseja iniciar o ciclo de vida dos testes dentro do processo de desenvolvimento. De acordo com (PEIXOTO, 2018), o Selenium possui uma API simples e objetiva, sendo uma excelente porta de entrada para profissionais que desejam iniciar na automação de testes. Ainda, o fato de ser uma ferramenta open-source contribui para sua constante evolução por meio de atualizações frequentes, correções de defeitos e uma vasta oferta de materiais de apoio entre as comunidades de QA em todo o mundo, assim com suporte de fácil acesso o framework cresceu e tem a maior aceitação na comunidade de testes.

3.2 JUnit

O JUnit é um framework de testes unitários para Java que se integra de forma eficaz com o Selenium, servindo como esqueleto organizacional para a construção de suítes de testes automatizados. Criado por Erich Gamma e Kent Beck, o JUnit tornou-se padrão de fato para a implementação de testes em projetos Java, oferecendo uma sintaxe limpa, baseada em anotações como @Test, @BeforeAll, @AfterEach e @DisplayName.

Uma das grandes vantagens do JUnit é sua integração com sistemas de build automation como Maven e Gradle, o que permite sua execução automatizada em pipelines de CI/CD. Também é possível configurar estratégias de execução paralela, agrupamento de testes por categorias e geração de relatórios em formatos legíveis por humanos ou por sistemas de monitoramento.(BECHTOLD.STEFAN et al.,)

Em ambientes de automação de testes web, o JUnit permite validar com precisão os comportamentos esperados da aplicação, comparando resultados obtidos com valores

esperados por meio de asserções como assertEquals, assertTrue, assertThrows, entre outros. Segundo estudo de (DESPA, 2020), a adoção de frameworks de testes unitários como o JUnit melhora significativamente a cobertura de testes e a velocidade de identificação de regressões, sendo um dos pilares da cultura de Desenvolvimento Orientado a Testes (TDD). O JUnit também se destaca pela capacidade de contribuir com a melhoria contínua do código, ao permitir a criação de baterias de testes automatizados e a identificação precoce de falhas. Além disso, promove a padronização dos testes, favorecendo a manutenção e a evolução do sistema. Sua adoção é bastante comum em projetos que utilizam metodologias ágeis, como Scrum, integrando-se naturalmente ao fluxo de desenvolvimento ágil.

Outro aspecto relevante é a integração do JUnit com ferramentas de automação de testes, como o Selenium WebDriver, possibilitando a execução de testes de interface gráfica diretamente no navegador. Nesse contexto, o JUnit é responsável pelo controle das execuções e validações, reforçando seu papel central na automação de testes E2E. Sua principal função é permitir a verificação automatizada da funcionalidade de métodos e classes de forma rápida e estruturada. A cada ciclo de execução, o JUnit gera relatórios que indicam o sucesso ou a falha dos cenários testados, possibilitando a execução individual de testes ou a execução em lote de todo o conjunto. Dessa forma, os resultados obtidos podem ser facilmente compartilhados com a equipe de desenvolvimento, promovendo maior visibilidade sobre a qualidade do software. Em síntese, o uso do JUnit no processo de desenvolvimento de software representa uma prática consolidada e altamente recomendada, especialmente para projetos que buscam qualidade, segurança e agilidade na entrega de seus produtos.

3.3 Page Object Model

O Page Object Model (POM) é um padrão de projeto amplamente utilizado no desenvolvimento de testes automatizados para aplicações web. Ele consiste na criação de uma camada de abstração onde cada página ou componente significativo da aplicação é representado por uma classe separada. Nessa classe, são encapsulados tanto os elementos da interface quanto as ações possíveis sobre eles, promovendo uma clara separação entre a lógica de navegação e a lógica de teste. A manutenção de testes automatizados em ambientes dinâmicos requer uma arquitetura que minimize o acoplamento entre o código dos testes e a estrutura visual da aplicação. Nesse contexto, o padrão Page Object Model (POM) emerge como uma solução elegante e eficaz. Ele propõe que cada página ou componente da interface da aplicação seja representado por uma classe, contendo os elementos da interface e os métodos de interação encapsulados (FOWLER, 2020).

Esse padrão foi popularizado com o avanço do uso do Selenium WebDriver e é

associado principalmente a engenheiros da Selenium Community, como Simon Stewart, o criador do Selenium WebDriver, por mais que o conceito de separação de responsabilidades em testes já existisse antes em outros contextos de desenvolvimento de software.

O uso do Page Object Model traz diversos benefícios para projetos de automação de testes, visto que ele promove uma organização mais eficiente e modular do código, reduzindo a duplicação e facilitando a manutenção. Quando a interface da aplicação sofre alterações, é necessário atualizar apenas as classes correspondentes às páginas afetadas, e não todo o conjunto de testes. Além disso, o POM melhora a legibilidade e a reusabilidade dos testes, o que é essencial para equipes que trabalham com metodologias ágeis e integração contínua (CI/CD). A principal vantagem do Page Object é a centralização da lógica de interface, facilitando a manutenção e a reusabilidade dos testes. Em vez de repetir seletores e interações em múltiplos arquivos de teste, estas ações ficam concentradas em classes específicas, reduzindo a duplicidade e melhora a legibilidade. Alterações na interface, como mudança de IDs ou estrutura HTML, exigem ajustes em somente um local, minimizando o impacto nos testes e aumentando a estabilidade da suíte. Este padrão se integra naturalmente com o Selenium e o JUnit, formando uma estrutura de testes altamente coesa e modular. A adoção de POM reduz o tempo de manutenção de suítes de testes funcionais em projetos de médio e grande porte, tornando-se uma prática essencial para equipes de QA maduras e com foco em escalabilidade.

Ao apresentar *Melhorando a manutenibilidade de suítes de testes com o padrão de objeto de página: um estudo de caso industrial*,2013 um estudo de caso industrial que analisa como o uso do padrão Page Object pode melhorar a manutenibilidade de suítes de testes automatizados, especialmente em testes de interfaces gráficas de aplicações web, os autores descrevem a aplicação desse padrão em um projeto real de uma empresa de desenvolvimento de software. Eles demonstram que a separação entre lógica de teste e a estrutura da interface do usuário proporciona melhor organização do código, redução do esforço de manutenção e maior reutilização dos componentes de teste. O estudo evidencia que, ao encapsular a lógica da interface em objetos específicos, os testes se tornam mais robustos, legíveis e fáceis de atualizar diante de mudanças na interface do sistema (LEOTTA et al., 2013).

Esse trabalho reforça a importância de boas práticas de design em testes automatizados e fornece evidências empíricas sobre os benefícios do Page Object Pattern na automação de testes com ferramentas como Selenium.

3.3.1 Considerações finais

Como conclusão do capítulo, pode-se afirmar que a automação de testes em sistemas web, quando fundamentada no uso integrado de ferramentas como Selenium WebDriver, JUnit e no padrão de projeto Page Object Model (POM), representa um alicerce

robusto para garantir qualidade, eficiência e escalabilidade nos processos de validação de software. O Selenium destaca-se por sua versatilidade, compatibilidade com múltiplas linguagens e integração com diversas plataformas e ferramentas de orquestração, enquanto o JUnit organiza e executa testes com precisão, promovendo boas práticas como o TDD e favorecendo a integração contínua. Já o POM contribui significativamente para a manutenibilidade e legibilidade dos testes, ao encapsular a lógica de interface em estruturas reutilizáveis e desacopladas. Juntas, essas tecnologias formam uma arquitetura sólida e sustentável para a automação de testes em ambientes ágeis, reduzindo custos, aumentando a confiabilidade das entregas e promovendo uma cultura de qualidade contínua no desenvolvimento de aplicações web.

4 Desenvolvimento

4.1 Proposta

Para colocar em prática o processo de teste como uma sugestão de aprimoramento, é vantajoso seguir as diretrizes apresentadas neste estudo, uma vez que permite formar uma equipe de qualidade, integrando-a em todas as fases do projeto. Para alcançar tal realidade, foi idealizado um processo com suas etapas descritas a seguir.

Em primeiro lugar, deve-se avaliar o cenário atual, objetivando entender como o software é desenvolvido e entregue atualmente. Possíveis ações para esta etapa: Identificar se existem testes manuais ou automatizados; verificar se há falhas recorrentes em produção; e realizar o mapeamento com quem (se alguém) se testa o software hoje. No caso do exemplo, os testes são realizados por desenvolvedores.

Em seguida, deve-se definir os objetivos e o escopo dos testes, a fim de estabelecer metas claras para a qualidade. Ações para esta etapa: estabelecer quais tipos de testes são prioritários (ex: testes funcionais, regressão, UI); alinhar com os objetivos de negócio (por exemplo, reduzir retrabalho e aumentar a estabilidade).

Uma etapa importante tanto quanto executar a automação é escolher as ferramentas adequadas para criar o ambiente e executar os testes. O objetivo desta etapa é, então, montar um ambiente de testes funcional e escalável. Neste estudo de caso, o foco foi o Selenium WebDriver e o JUnit, devido a baixa curva de aprendizado e o fato de ser open-source, com comunidades fortes e engajadas, facilitando o aprendizado dos analistas, liberando-os mais rápido para atuar, com ferramentas comuns, como automação web: Selenium e JUnit; e execução de testes automatizados: JUnit + CI/CD (Jenkins, GitHub Actions, etc.). Assim, nesta etapa, é válido criar uma estratégia de testes junto ao time de negócios, intencionando combinar melhor os prazos e, portanto, formalizar como os testes serão conduzidos. Ações dessa etapa: criar um plano de testes com critérios de entrada, saída e cobertura mínima; definir uma pirâmide de testes (unitários, integração, UI); planejar como será a automação dos testes (cenários críticos, testes repetitivos).

A etapa de colocar tudo em prática consiste em implantar os testes de forma gradual e iterativa, com o objetivo de integrar a cultura de qualidade sem causar interrupções no fluxo de trabalho das equipes. Ações para a conclusão da implementação do processo de testes: começar com testes em funcionalidades mais simples e evoluir os cenários para cair em jornadas mais críticas ou com mais defeitos; introduzir testes automatizados nos pipelines de integração contínua; e envolver os desenvolvedores e testadores nas revisões de testes. Um bônus: promover a cultura da qualidade é uma possibilidade de deixar claro

que o QA não é o dono da qualidade, e sim o propagador de tal prática. A realização de treinamentos sobre testes e automação, a integração do QA nas reuniões ágeis (como planejamento e retrospectivas) e a valorização e documentação dos aprendizados de defeitos encontrados em produção são essenciais para garantir um trabalho de qualidade. Na imagem abaixo é apresentado o fluxograma do processo de testes proposto para o ciclo de testes.

Figura 5 – Processo Proposto

Fonte: Autora.

4.2 Contextualização

Existem empresas que adotam metodologias ágeis para gerenciar seus processos de desenvolvimento, mas que não possuem uma equipe dedicada à qualidade de software. Nesses casos, os testes costumam ser realizados pelos próprios desenvolvedores, com foco restrito aos testes unitários. Esse modelo pode acarretar problemas recorrentes de qualidade, uma vez que a prioridade é acelerar a entrega para o usuário final. No entanto, essa busca por velocidade pode gerar custos elevados, especialmente diante das exigências dos clientes em relação à qualidade dos demais produtos disponíveis no mercado.

Nesse contexto, uma proposta eficaz é integrar os testes automatizados no ciclo de vida do desenvolvimento, demonstrando a importância de incorporar práticas de qualidade desde as primeiras etapas. A implementação de um processo de testes estruturado, que funcione como uma etapa prévia ao desenvolvimento ou que, em alguns casos, possa ocorrer em paralelo ao desenvolvimento das funcionalidades, pode contribuir significati-

vamente para a saúde e sustentabilidade do produto. Para iniciar o processo de testes em uma empresa que desenvolve uma aplicação web, é essencial seguir uma abordagem estruturada para garantir que os testes sejam eficazes, escaláveis e integrados ao processo de desenvolvimento. Aqui será apresentado um passo a passo com as etapas necessárias para viabilizar esse processo de implantação.

4.3 Modelo de testes atual na empresa

A empresa analisada neste estudo de caso terá seu nome preservado por questões legais de proteção de dados. Para fins de referência, será denominada como empresa Nayafau, tratando-se de uma organização do segmento varejista e educacional. Atualmente, a Nayafau não conta com uma equipe dedicada de Quality Assurance (QA), impactando diretamente na qualidade do software desenvolvido. Todo o processo da aplicação web é conduzido exclusivamente pela equipe de desenvolvedores, que além de escreverem o código, assumem também a responsabilidade pelos testes; esses testes, no entanto, são realizados de maneira informal, sem planejamento prévio, critérios de aceitação bem definidos ou com a documentação adequada. Assim, em sua maioria, consistem em verificações manuais e superficiais, voltadas apenas para cenários básicos executados por seus próprios desenvolvedores, como é possível ver na imagem abaixo.

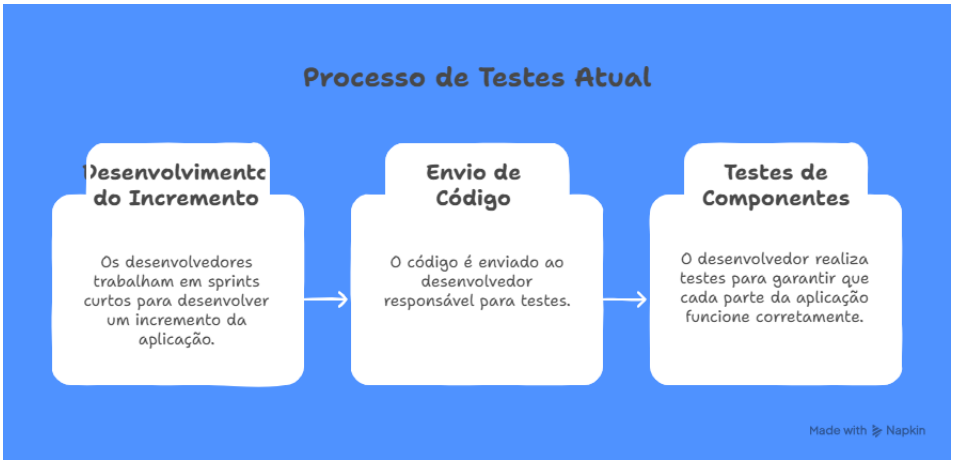

Figura 6 – Processos de Testes Atual

Fonte: Autora

A ausência de uma abordagem sistemática de testes tem gerado consequências significativas. Um dos principais problemas enfrentados é o alto índice de defeitos em produção, o que compromete a estabilidade e a confiabilidade da aplicação. Esses defeitos afetam diretamente a experiência do usuário final, gerando insatisfação, aumento do número de chamados no suporte técnico e perda de credibilidade da marca. Para mais, o retrabalho é recorrente, uma vez que funcionalidades precisam ser revisadas e corrigidas após já terem sido disponibilizadas ao público. Sem uma cultura de testes bem estabe-

lecida e sem automação para garantir validação contínua e confiável, a escalabilidade do sistema também se torna limitada. O crescimento e a evolução do produto digital se tornam mais lentos e arriscados, com maior propensão a falhas e custos operacionais elevados. Esse cenário reforça a necessidade urgente de implementação de práticas de qualidade, incluindo a introdução de testes automatizados no ciclo de desenvolvimento, a fim de melhorar a eficiência e assegurar a entrega de um software mais robusto e confiável.

4.4 Implementação do processo de testes na empresa

O primeiro passo é o entendimento do produto. Compreender é essencial para levantar os requisitos funcionais e não funcionais da aplicação, incluindo os fluxos de usuário, requisitos de desempenho, segurança, entre outros, com intuito de facilitar na definição do que deve ser testado e na determinação de quais tipos de testes são necessários, tais como:

- Testes funcionais (verificar se a aplicação funciona como esperado);

- Testes de usabilidade (foco na experiência do usuário);

- Testes de segurança (verificação contra vulnerabilidades);

- Testes de desempenho (tempo de resposta, carga);

- Testes de integração (com outros sistemas ou APIs).

A determinação das áreas críticas deve ser feita de forma precisa e detalhada, pois é neste estágio que se identificam as funcionalidades e partes mais críticas do aplicativo. Assim, é necessário mais cuidado nos testes, como login, pagamento e integração com sistemas externos. Isso minimizaria um mau funcionamento nos pontos principais da jornada, o que estará incluído no E2E, já que os cenários automatizados devem ser os principais pontos da jornada.

4.5 Configuração do Ambiente de Testes

Para iniciar o processo de automação de testes, é necessário configurar o ambiente. Para o estudo deste caso, toda a configuração e instalação de ferramentas são direcionadas para o Windows, enquanto o ambiente de desenvolvimento escolhido para a implementação dos testes foi a *(IDE) Integrated Development Environment*, IntelliJ IDEA, utilizando a linguagem de programação Java. Na sequência, são importadas as dependências e bibliotecas para a configuração do projeto, como JUnit, Maven, Selenium, todos contidos no arquivo POM do projeto.

- Selenium para automação de testes web;

- JUnit, para testes unitários e de integração;

- Versionamento de Código: Para controlar as versões de testes e integrar ao fluxo de desenvolvimento.

Primeiro passo é instalar o IntelliJ , a figura abaixo apresenta o layout da IDE escolhida:

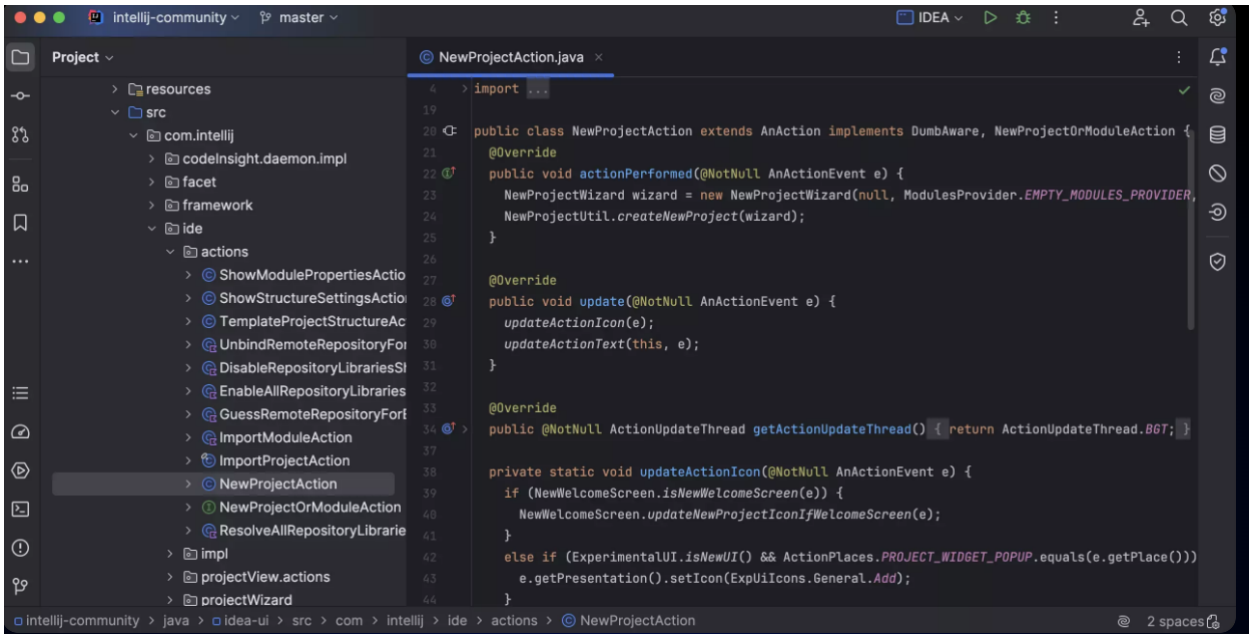

Figura 7 – IDE - IntelliJ Idea

FONTE: Extraído www.jetbrains.com

Em seguida, para a configuração do ambiente de testes, é necessário o download do Java SDK, o passo a passo da instalação do JDK e ás demais ferramentas abordadas nesse estudo podem ser encontradas no apêndice desse estudo. O JDK é um pré-requisito para permitir automatizar os testes integrados ao Selenium, uma vez que foi o framework escolhido para viabilizar o estudo, pois além de ser uma ferramenta open-source, ela também tem uma comunidade forte de usuários dispostos a oferecer suporte no avanço do conhecimento de novos usuários.

Ainda, é necessário realizar o download do Chrome driver para conseguir realizar a interação com o navegador. Entretanto, caso se queira utilizar outros navegadores, pode-se utilizar Mozilla Firefox, Microsoft Edge e Opera, que também são opções para validar o redirecionamento. Para selecionar o navegador mais eficiente, é interessante conhecer o público-alvo para obter dados e verificar qual o navegador mais acessa a sua aplicação e, assim, concentrar testes nele, garantindo uma experiência melhor. Em relação às bibliotecas, deve-se acessar o Maven Repository e realizar a instalação da versão mais recente para o JUnit. Esse repositório é basicamente uma biblioteca para auxiliar na escrita dos testes de unidade com Java, ao permitir testes de unidade voltados ao porte da aplica-

ção. Com o JUnit configurado como dependência no arquivo pom.xml, é possível ver na figura 8 que se tem uma estrutura de testes e uma estrutura de execução de testes muito abrangente, amparando na hora de escrever os testes automatizados do caso em questão para avaliar a interface gráfica da aplicação.

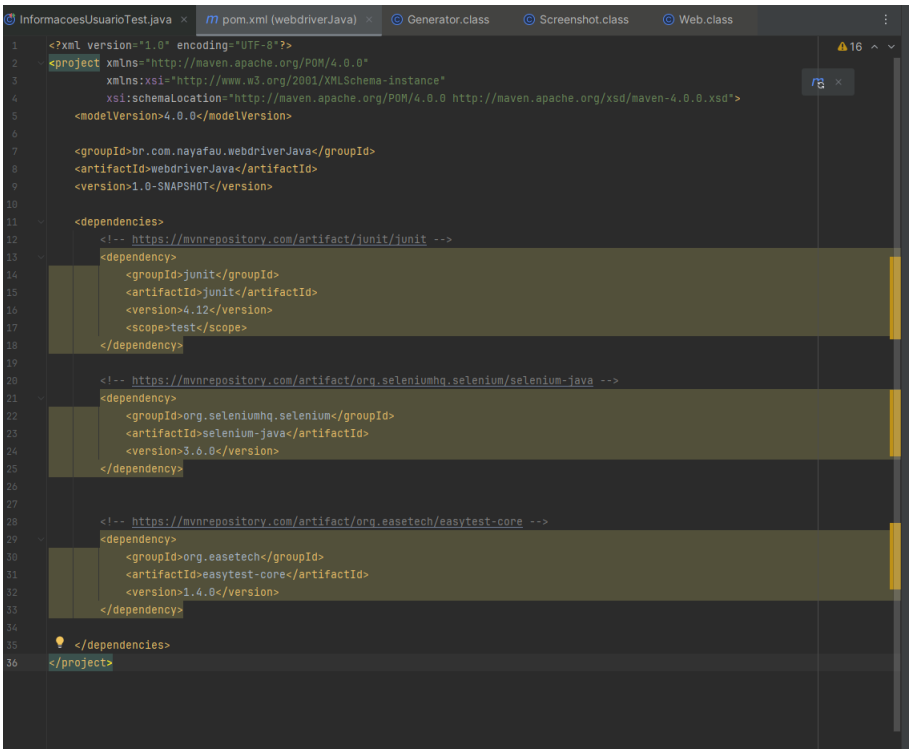

Figura 8 – Arquivo POM.XML

Fonte: autora

O Selenium se encontra no mesmo repositório é possível ver nas abaixo *figuras 9 e 10* . Para saber mais sobre as suas variações, é válido acessar o site tendo em vista que o mesmo reúne uma variedade de informações úteis sobre suas principais ferramentas: Selenium IDE, Selenium WebDriver e Selenium Grid. Além disso, oferece documentação, tutoriais, dicas e links relevantes. Tal website pode ser acessado pelo endereço: http://www.seleniumhq.org/.

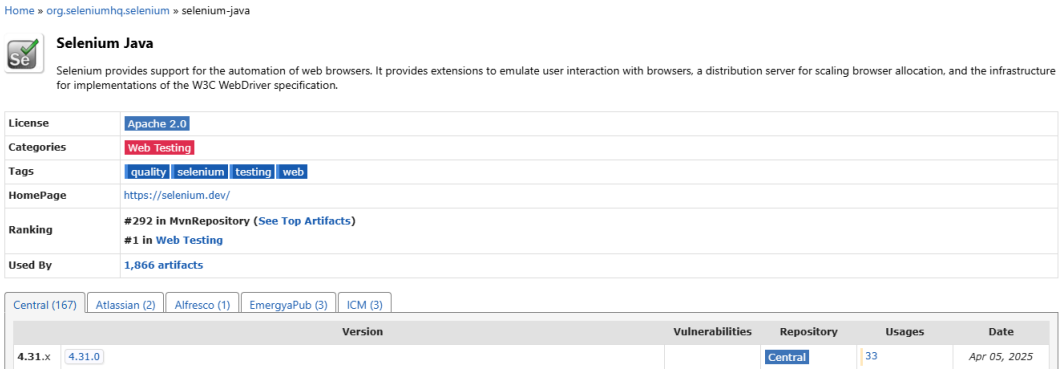

Figura 9 – Arquivo Selenium Java

Fonte: Extraído de https://mvnrepository.com/

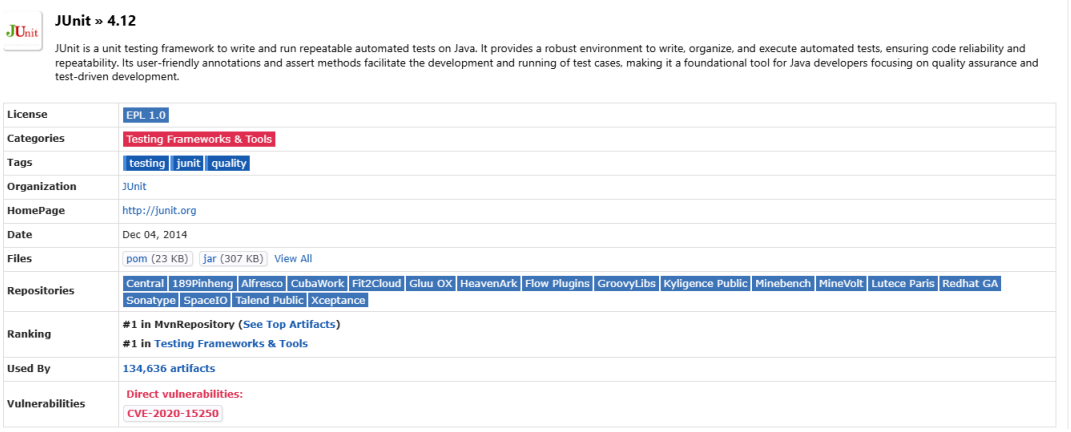

Figura 10 – Arquivo Junit

Fonte: Extraído de https://mvnrepository.com/

4.6 Desenvolvimento de Testes

Com o ambiente configurado, deve ser realizada a execução dos testes, considerando que no planejamento está contido na escrita dos cenários e, assim, permitindo criar uma jornada E2E para os testes. Vale reforçar que os testes manuais são insubstituíveis, mesmo com a automação não se deve descartar essa técnica de testes, pois a cobertura em alguns casos específicos é maior, ao permitir simular com exatidão a maioria dos cenários, assim como o usuário final.

Testes manuais iniciais: é recomendado começar com testes manuais para entender quais insumos e massas de dados devem ser criados para permitir a navegação na aplicação. Isso auxiliará na estimativa do tempo para execução dos cenários, por poder existir dependência de outras equipes dentro do mesmo ambiente para gerar a massa, ou pode ser necessário criar parte da massa no banco de dados, sendo este um passo subsequente para validar o que está ativo no ambiente em uma API, que valida a integração dos serviços

e que habilita a massa no ambiente. Esses passos são mais fáceis de serem realizados durante a execução manual. É fundamental validar que os cenários cobrem 100% a aplicação e identificar os fluxos críticos, pois isso também ajudará a identificar testes que podem ser automatizados mais tarde.

Automatização de Testes: após identificar os cenários críticos da jornada do usuário, é possível definir o escopo dos testes E2E (end-to-end) e escrever testes automatizados para esses casos. O artigo de (LEOTTA et al., 2013) destaca que a automação dos testes E2E é essencial para lidar com a complexidade dos sistemas modernos e acelerar ciclos de desenvolvimento. Os testes E2E são abordados como parte da execução de fluxos completos de usuários com base em modelos de comportamento do sistema.

A modelagem ajuda a gerar cenários realistas e relevantes para o teste E2E, garantindo uma melhor cobertura de caminhos críticos do sistema. O uso de ferramentas como Selenium Webdriver combinadas com JUnit, assim como abordado neste estudo, são ótimas escolhas para melhorar a cobertura e reduzir o esforço manual.

Testes de UI: é possível criar scripts de teste para interações com a interface (ex.: preenchimento de formulários, navegação entre páginas, testes de layout responsivo), realizados como framework Selenium WebDriver.

Testes de Integração (testes em diferentes navegadores e dispositivos): o Selenium foi escolhido para ser demonstrado neste estudo, por permitir realizar os testes em variados navegadores. Dessa maneira, é possível validar em layouts distintos a captação e chegada do usuário na sua aplicação, assim como realizar testes em diferentes navegadores e dispositivos para garantir a compatibilidade da aplicação.

4.6.1 Execução dos Testes

Ao iniciar uma automação, considera-se que os testes já foram escritos e executados manualmente. Uma fala muito comum, porém perigosa, é a de que os testes automatizados podem substituir os testes manuais, enquanto, na verdade, os testes manuais são uma garantia de que permite automatizar partes críticas da jornada para ganhar escalabilidade. O estudo de (THANT; TIN2, 2023) faz um comparativo entre testes manuais e testes automatizados, justamente com o foco de trazer dados que fortalecem a importância de uma abordagem híbrida, ao mostrar que testes manuais ainda são indispensáveis, principalmente para detectar falhas que exigem julgamento humano, enquanto os testes automatizados ajudam a ganhar velocidade e reduzir custos.

Os testes automatizados, ao serem incorporados no ciclo de desenvolvimento, permitem a execução de forma mais rápida, o que facilita na validação da garantia em projetos com metodologia ágil. Isso é possível e pode ser realizado em cada incremento, ou seja, todo incremento ao código pode ser testado de forma rápida, verificando, ao final da

etapa, se o regressivo foi executado com sucesso e se o incremento não gerou erros no já desenvolvido na aplicação. Na aplicação utilizada como exemplo, é possível definir cenários padrão de testes em uma aplicação web, como acesso ao navegador, login do usuário e preenchimento de informações adicionais. Esses testes seguem o chamado caminho feliz, ou seja, representam cenários positivos com o fluxo ideal de uso. Durante a automação com Selenium, é possível configurar a criação de uma pasta para armazenar as evidências geradas. Essas evidências ficam disponíveis na pasta "screenshot", onde são salvos os registros visuais dos testes executados, indicando se cada cenário teve o status sucesso ou falha.

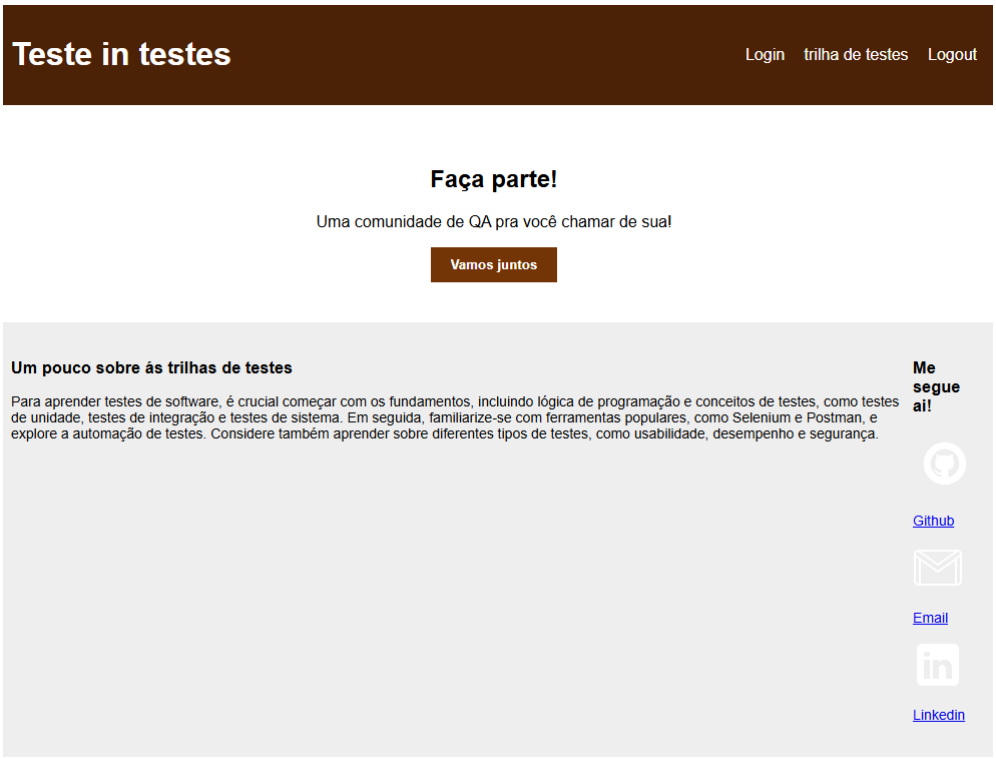

Figura 11 – Web site criado para exemplificar os testes

Fonte: A autora

Na figura 11, é possível ver uma webpage, customizada para ser uma comunidade educacional de QAs. Nessa webpage, é possível validar o cenário de login. No topo da tela na aplicação, têm-se as opções de realizar o clique em trilhas de testes, redirecionando para treinamentos com foco em automação de software e as ferramentas mais utilizadas. Ainda, vê-se o botão que guia para outra parte da jornada, como o "Vamos juntos", que redireciona para uma tela em que é possível criar uma nova atividade como meta de estudos. No canto direito da tela, é necessário validar o redirecionamento para cada um dos ícones, sendo eles Github, Email e LinkedIn.

A descrição dos cenários é simplificada devido à capacidade do JUnit de conduzir os testes de forma organizada e de fácil compreensão. Com o uso do trecho de código abaixo, é possível abrir o navegador automaticamente. Durante esse processo, uma nova aba é aberta e, no topo da tela, é exibida uma indicação de que o navegador está sendo manipulado por uma ferramenta de automação.

A seguir, demonstra-se uma maneira simples, que permite identificar os elementos na tela, comentando o passo a passo da jornada, com intenção de entender os cenários que precisam ser contemplados no escopo E2E. Após o acesso ao navegador e ao login do usuário, colocam-se os seguintes passos:

- Clicar no link que possui o texto "Login";
- Identificando o formulário de login;
- Digitar no campo com user "Login", que está dentro do formulário de id "signinbox", o texto "nayafau";
- Digitar no campo "Password", que está dentro do formulário de id "signinbox", o texto "123456"
- Clicar no link com o texto "Login".

A seguir é apresentado ás seguintes figuras: figura 12 – Acessando o navegador que possibilita ver o comportamento da abertura da tela do navegador e respectivamente a tela de login é apresentada duas vezes, sendo uma em branco e outra com os dados do usuário. *Figuras: 13 e 14*

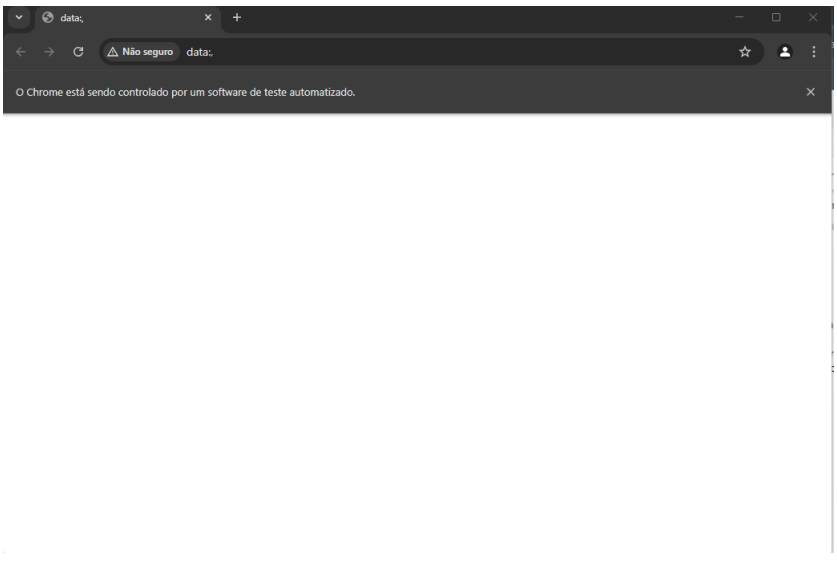

Figura 12 – Acessando o navegador

Fonte: A autora

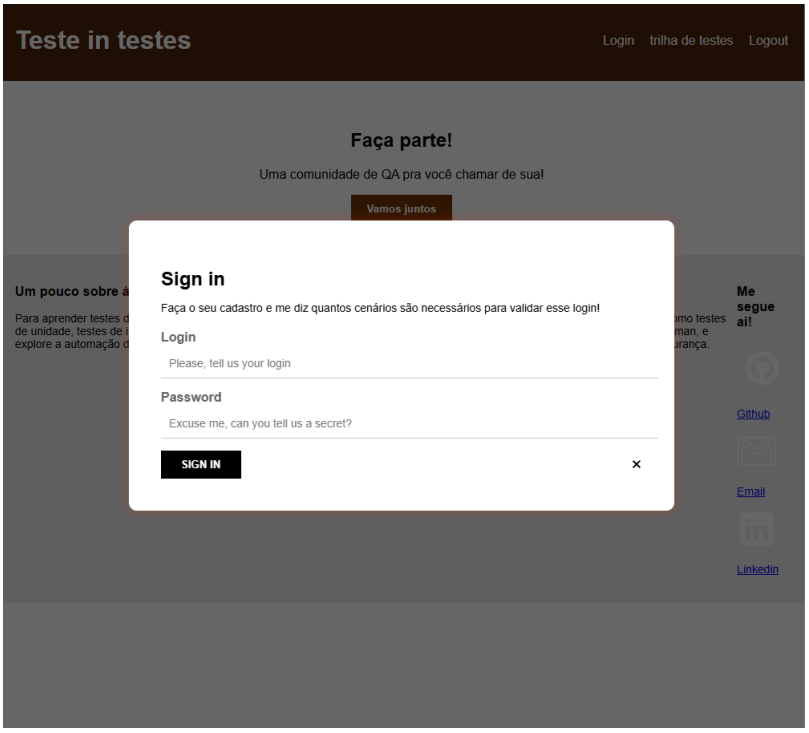

Figura 13 – Tela de login

Fonte: A autora

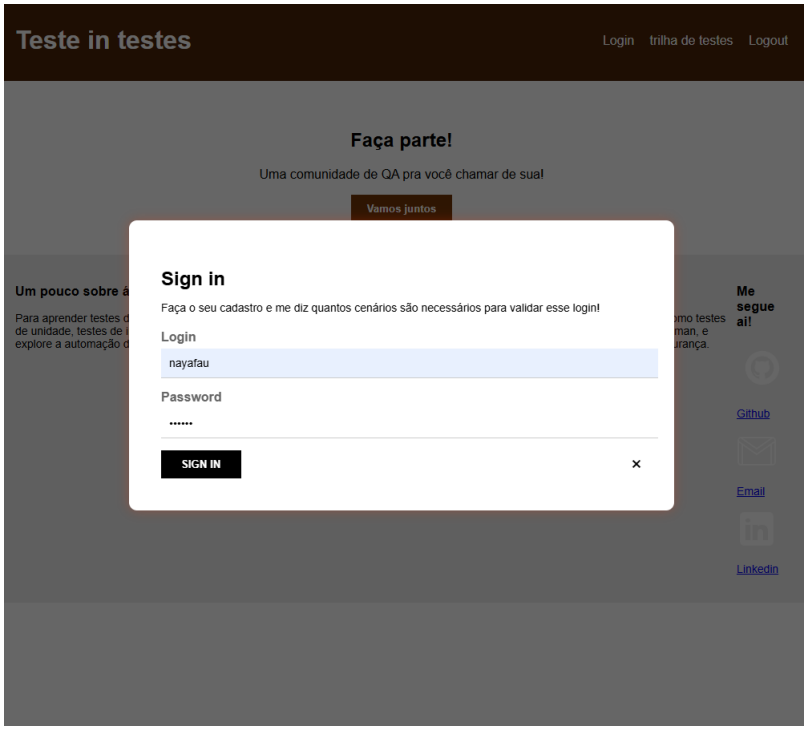

Figura 14 – Tela de login com os dados do usuário

Fonte: A autora

4.7 Import das bibliotecas do JUnit

Os métodos apresentados na figura 15 são as bibliotecas e esses são componentes padrão do JUnit, a biblioteca de testes mais comum no Java:

(@Before *@Antes*): Método executado antes de cada teste (útil para iniciar navegador, por exemplo).

(@After *@Depois*): Executado após cada teste (útil para fechar o navegador).

@Test: Indica que um método é um teste.

(@RunWith *"Rodar com" ou "Executar com")* : Usado para especificar um runner alternativo (como o DataDrivenTestRunner, que vem do EasyTest).

WebDriver: Interface principal do Selenium para controlar o navegador.

LoginPage: Provavelmente é uma classe criada pelo usuário, seguindo o padrão Page Object, que encapsula os elementos da página de login.

suporte.Web: Também deve ser uma classe do usuário, usada para iniciar o navegador (muito comum em projetos de automação).

import static org.junit.Assert.*: Importa todos os métodos de asserção do JUnit (assertEquals, assertTrue, etc.), usados para verificar os resultados dos testes.

InformacoesUsuarioPageObjectsTest.java | InformacoesUsuarioTest.java

```
package tests;

import static org.junit.Assert.*;

import org.easetech.easytest.annotation.DataLoader;
import org.easetech.easytest.annotation.Param;
import org.easetech.easytest.runner.DataDrivenTestRunner;
import org.junit.After;
import org.junit.Before;
import org.junit.Rule;
import org.junit.Test;
import org.junit.rules.TestName;
import org.junit.runner.RunWith;
import org.openqa.selenium.*;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.Select;
import org.openqa.selenium.support.ui.WebDriverWait;
import suporte.Generator;
import suporte.Screenshot;
import suporte.Web;
```

Figura 15 – Bibliotecas JUnit

Fonte: A autora

Essas assertivas contidas na imagem acima são fundamentais para garantir que o comportamento do software está correto em cada teste. Se a comparação falhar, o teste falha, indicando haver um erro no sistema ou no código testado.

4.8 Mapeando elementos

No trecho de código, apresentado na figura 16, tem-se o Before, notação que indica que o método deve ser executado antes de cada teste. Este bloco é executado antes de cada teste, normalmente usado para preparar o ambiente de teste, como abrir o navegador, configurar variáveis, carregar dados ou fazer login. Isso evita a repetição de código em cada teste e garante que tudo esteja pronto antes da execução do teste propriamente dito.

Agora, no passo seguinte, a notação @Test marca um método como um teste de fato. O JUnit executa todos os métodos anotados com @Test como testes automatizados. No método, é possível simular interações com a aplicação e usar assertivas para verificar se o comportamento está correto. A notação @After é usada para marcar um método que deve ser executado após cada método de teste ser executado. Isso geralmente é útil para limpar recursos, fechar conexões ou finalizar o navegador, no caso de testes com Selenium WebDriver.

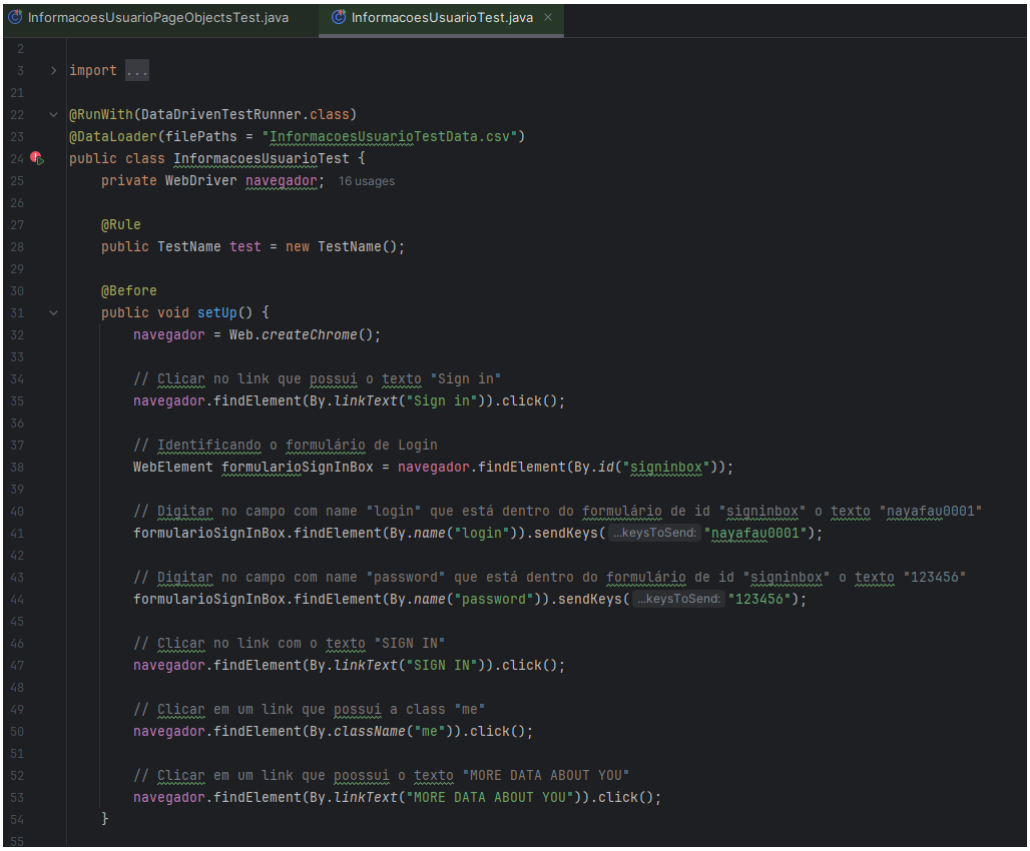

Figura 16 – Mapeando elementos

Fonte: A autora

Nesse momento, a automação já possibilita abrir o navegador, realizar o login e adicionar informações do usuário. Na tela a seguir, é possível interagir melhor com novos elementos, como, por exemplo na figura 19, na interface, que é um formulário modal que permite ao usuário adicionar uma nova tarefa com informações como título, mês, ano,

data limite, tempo estimado, descrição e status da tarefa (concluída ou não). O objetivo da tela é auxiliar o usuário a programar suas atividades e evitar a procrastinação.

O trecho de código a seguir executa um teste com o objetivo de simular um usuário preenchendo corretamente um formulário web de tarefas (como um planejador de atividades) e enviando os dados.

No teste, o script da imagem abaixo (figura 17) localiza e preenche todos os campos obrigatórios do formulário (como título da atividade, mês, ano, data limite, tempo estimado e descrição), escolhe a opção "não concluído"no menu suspenso de status e envia o formulário, ao clicar no botão de "Salvar". Por fim, faz-se uma verificação simples para garantir se o formulário ainda está visível após o envio, o que pode indicar que o processo ocorreu sem erros. A notação @Test indica que este é um caso de teste, e o método @After é executado após o teste terminar nesse caso, para fechar o navegador e limpar o ambiente, essa automação é útil para validar se o formulário da aplicação está funcionando corretamente, ajudando a detectar falhas antes da entrega do sistema ao usuário final. Na figura 18 é possível verificar o tempo total da execução apresentado na janela do Junit, com status de sucesso o com o tempo de 5 segundos.

```
    @Test  no usages
    public void testPreencherFormularioComSucesso() {
        navegador.findElement(By.name("taskTitle")).sendKeys( ...keysToSend: "Estudar testes com Selenium");

        // Como há múltiplos campos com name="taskTitle", é melhor encontrar os campos diretamente pela ordem
        navegador.findElements(By.name("taskTitle")).get(1).sendKeys( ...keysToSend: "05 - Maio");
        navegador.findElements(By.name("taskTitle")).get(2).sendKeys( ...keysToSend: "2025");

        navegador.findElement(By.name("dateLimit")).sendKeys( ...keysToSend: "2025-05-30");
        navegador.findElement(By.name("timeLimit")).sendKeys( ...keysToSend: "02:00");
        navegador.findElement(By.name("details")).sendKeys( ...keysToSend: "Planejar bem e não deixar para última hora!");

        Select comboStatus = new Select(navegador.findElement(By.name("done")));
        comboStatus.selectByValue("no");

        navegador.findElement(By.cssSelector("button[type='submit']")).click();

        // Aqui você poderia validar algum tipo de resposta, como uma mensagem ou redirecionamento
        // Por exemplo: verificar se o botão ainda existe
        Assert.assertTrue(navegador.findElement(By.tagName("form")).isDisplayed());
    }

    @After
    public void tearDown() {
        navegador.quit();
    }
}
```

Figura 17 – Código Adicionar novas informações

Fonte: A autora

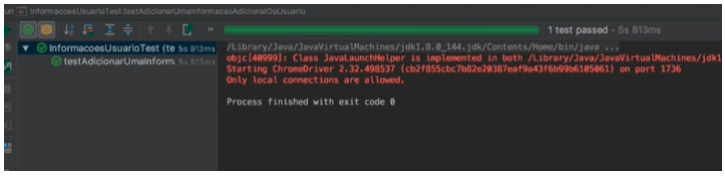

Figura 18 – JUnit sucesso

Fonte: A autora

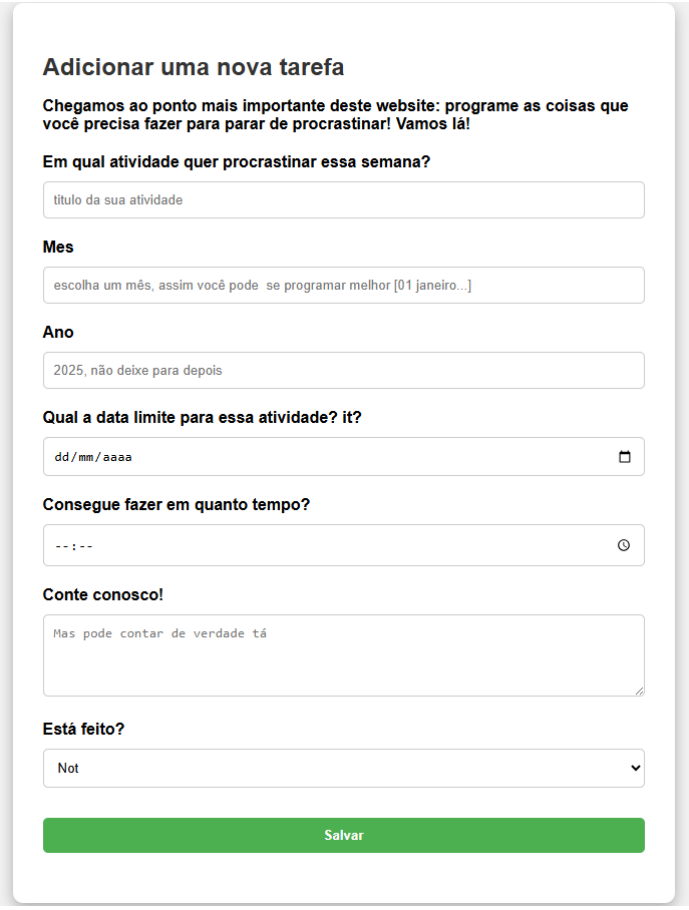

Figura 19 – Tela Adicionar novas informações

Fonte: A autora

4.9 Resultados

A automação de testes de software permitiu diversos benefícios estratégicos para a empresa. Considerando o cenário atual da Nayafau, empresa com foco em visibilidade para aprendizado de QAs, há uma redução significativa de retrabalho e falhas em produção, pois os testes automatizados ajudam a identificar erros logo nas fases iniciais do desenvolvimento. Isso evita que problemas cheguem ao ambiente do cliente, melhorando a qualidade do produto final.

Outro ganho importante é o aumento da confiança nas versões entregues. Quando

os testes são automatizados, possibilita-se garantir que os principais fluxos da aplicação estão funcionando corretamente após cada alteração, promovendo maior estabilidade e previsibilidade.

4.9.1 Considerações finais

Por fim, a automação permite que a empresa escale seu produto de maneira mais eficiente. Em vez de aumentar proporcionalmente o número de testadores conforme o sistema cresce, é possível cobrir mais cenários com menos esforço manual, tornando o processo mais sustentável e econômico a longo prazo. Logo, os desenvolvedores podem manter seu foco em apenas uma atividade, possibilitando, portanto, um desempenho melhor.

5 Conclusão

Este trabalho teve como propósito demonstrar, por meio de um estudo de caso prático, como implementar um processo de testes automatizados em empresas que ainda não adotam práticas estruturadas de garantia de qualidade (QA). Em um cenário cada vez mais competitivo no setor de tecnologia, a ausência de um processo de testes contínuo e eficiente tem se mostrado um problema recorrente, especialmente em pequenas e médias empresas, onde essa responsabilidade acaba sendo atribuída exclusivamente aos desenvolvedores. Essa abordagem, além de sobrecarregar a equipe técnica, compromete a qualidade das entregas, aumenta o retrabalho e eleva os riscos de falhas em produção.

O estudo apresentou uma proposta prática e acessível para iniciar o processo de testes automatizados, utilizando ferramentas open-source como Selenium WebDriver e JUnit. Com isso, foi possível comprovar que, mesmo empresas com recursos limitados, podem estruturar um processo de testes eficaz, integrado ao ciclo de desenvolvimento, sem grandes investimentos financeiros.

A proposta seguiu etapas claras: mapeamento do fluxo atual; escolha de ferramentas adequadas; padronização do código de testes; capacitação da equipe; criação de testes automatizados para os fluxos críticos; e integração ao pipeline de entrega contínua. Esses passos demonstraram, objetivamente, como é possível iniciar a jornada da automação de testes a partir do zero, com foco na melhoria da qualidade do software e na redução de falhas.

Além de apresentar as ferramentas e técnicas utilizadas, o estudo enfatizou os ganhos observados com a adoção da automação: diminuição de erros manuais; agilidade na identificação de defeitos; melhor aproveitamento da equipe de desenvolvimento; aumento da cobertura de testes e reuso de scripts em ciclos de regressão. Como uma proposta de continuidade para trabalhos futuros, recomenda-se expandir o escopo dos testes automatizados, incluindo validações de segurança e performance. A adoção gradual de uma cultura de testes desde a concepção do projeto também se mostrou essencial para sustentar a qualidade no longo prazo.

No contexto do curso de Sistemas de Informação, algumas disciplinas desempenharam um papel fundamental para a construção e desenvolvimento deste estudo, por oferecerem os conhecimentos teóricos e práticos necessários à compreensão e aplicação dos testes de software. A disciplina de Engenharia de Software se destacou por apresentar de forma estruturada todas as etapas do ciclo de vida de um sistema, desde a análise de requisitos e definição dos objetivos do negócio, até a modelagem, escolha da metodologia de desenvolvimento e a entrega final do produto. Esse conhecimento foi essencial para con-

textualizar a automação de testes dentro de um processo de desenvolvimento completo. A disciplina de Programação Orientada a Objetos contribuiu significativamente ao introduzir boas práticas de desenvolvimento e conceitos fundamentais como encapsulamento, herança e polimorfismo pilares importantes na criação de códigos limpos, reutilizáveis e estruturados, especialmente relevantes na construção de frameworks de testes. Já a disciplina de Banco de Dados foi indispensável por possibilitar a integração entre a lógica de programação e o armazenamento e manipulação de dados, permitindo a geração de massas de dados realistas e o suporte necessário à comunicação entre microserviços durante os testes automatizados. Dessa forma, essas três disciplinas do curso de Sistemas de Informação Engenharia de Software, Programação Orientada a Objetos e Banco de Dados foram essenciais para embasar técnica e conceitualmente a realização deste estudo de caso, evidenciando sua relevância na formação de profissionais capacitados para atuar com qualidade no desenvolvimento e validação de sistemas.

Para estudos futuros, sugere-se a investigação da aplicação de testes automatizados em ambientes com arquitetura baseada em microsserviços, onde os desafios de integração, comunicação entre serviços e deploy contínuo exigem abordagens mais avançadas e específicas. Além disso, destaca-se o potencial do uso de inteligência artificial na geração automática de casos de teste, o que pode aumentar significativamente a cobertura de testes e a detecção precoce de falhas, especialmente em sistemas de alta complexidade e dinamicidade. Ferramentas como o StackSpot, que auxiliam na modelagem e automação de casos de uso desde as fases iniciais do desenvolvimento, também representam uma oportunidade relevante de pesquisa, pois permitem alinhar testes automatizados com requisitos de negócio de forma mais estruturada e padronizada.

Conclui-se, portanto, que este estudo não somente discutiu a importância dos testes de software, mas forneceu um caminho viável para empresas que não têm esse processo implementado. Ao exemplificar como a automação pode ser iniciada e incorporada ao fluxo de trabalho, o trabalho contribui para dar visibilidade à área de QA e reforça que a qualidade não deve ser tratada como uma etapa opcional, e sim como parte integrante do desenvolvimento de software. Espera-se que este estudo contribua para dar visibilidade à importância do processo de testes automatizados, especialmente em aplicações web, promovendo uma cultura organizacional mais consciente e estruturada em relação à qualidade. Que mais empresas reconheçam o valor estratégico da área de QA e invistam em soluções que garantam entregas mais seguras, eficientes e com alto padrão de confiabilidade para seus usuários.

Referências

ADERSON.RIOS, E. B. **Base de conhecimento em teste de software**. Martins fontes. [S.l.]: Martins Fontes, 2007. v. 3. 0-264 p. ISBN 978-8599102893. Citado na página 23.

ALMEIDA, É. R. de; ABREU, B. T. de; MORAES, R.; MARTINS, E. Avaliação de um método para estimativa de esforço para testes baseado em casos de uso. **SBC**, p. 331–338, 2008. Citado na página 18. Disponivel em: https://doi.org/10.5753/sbqs.2008.15553

ANICHE, M. **Testes automatizados de software: Um guia prático**. [S.l.]: Editora Casa do Código, 2015. Citado na página 32.

BECHTOLD.STEFAN; BRANNEN, S.; LINK, J.; MERDES, M.; PHILIPP, M.; RANCOURT, J. de; STEIN, C. **Guia do usuário do JUnit 5**. Disponível em: <https://junit.org/junit5/docs/current/user-guide/>. Citado na página 36.

BECK, K.; BEEDLE, M.; BENNEKUM, A. V.; COCKBURN, A.; CUNNINGHAM, W.; FOWLER, M.; GRENNING, J.; HIGHSMITH, J.; HUNT, A.; JEFFRIES, R. et al. Manifesto for agile software development. **Manifesto for agile software developmen**, Snowbird, UT, 2001. Citado na página 22.

BEIZER, B. **Software Testing Techniques**. Dreamtech, 2003. 0–550 p. ISBN 9788177222609. Disponível em: <https://books.google.com.br/books?id=Ixf97h356zcC>. Citado 2 vezes nas páginas 9 e 20.

COSTA, J. O. I. da; SANTOS, V. K. F. dos. Explorando as melhores prÁticas e ferramentas para automaÇÃo de testes de software com selenium. **Revista ft**, v. 29, p. 53–54, 2025. ISSN 16780817. Disponível em: <https://revistaft.com.br/explorando-as-melhores-praticas-e-ferramentas-para-automacao-de-testes-de-software-com-selenium>. Citado na página 33. Disponivel em: https://doi.org/10.69849/revistaft/th102502210953

DELAMARO, M.; JINO, M.; MALDONADO, J. **Introdução ao teste de software**. [S.l.]: Elsevier Brasil, 2013. Citado na página 16.

DESPA, M. Software testing automation: Introduction and best practices. **International Journal of Advanced Computer Science and Applications (IJACSA)**, The Science and Information Organization, v. 11, n. 6, p. 357–362, 2020. Disponível em: <https://thesai.org/Downloads/Volume11No6/Paper_49-Software_Testing_Automation_Introduction_and_Best_Practices.pdf>. Citado na página 37.

DIJKSTRA, E. W. I. notes on structured programming. **I. Notes on Structured Programming**, 1971. Citado na página 10.

DIMES, T. **Scrum Essencial**. [S.l.]: Babelcube Inc., 2014. Citado na página 24.

FILHO, T. R. M.; RIOS, E. **Teste de software: o ponto de partida para qualquer projeto de teste de software é uma metodologia de testes consistente e adequada ao ambiente de desenvolvimento da empresa**. [S.l.]: Rio de Janeiro: Alta Books, 2003. Citado 2 vezes nas páginas 9 e 31.

FILHO, W. d. P. P. Engenharia de software: Fundamentos. **Métodos e Padrões-2a edição-LTC-2003**, 2003. Citado na página 20.

FOWLER, M. **Patterns of enterprise application architecture**. [S.l.]: Addison-Wesley, 2020. Citado na página 37.

GAROUSI, V.; ZHI, J. A survey of software testing practices in canada. **Journal of Systems and Software**, Elsevier, v. 86, n. 5, p. 1354–1376, 2013. Citado na página 35. Disponivel em: https://doi.org/10.1016/j.jss.2012.12.051

GONSALVES, W. **Quase R**$9milhões foram sacados em falha que viralizou como 'bug do Nubank' - Estadão. São Paulo, SP, Brasil : [s.n.], 2024. Disponível em : <>$. Citado na página 9.

HANNA, M.; ABOUTABL, A. E.; MOSTAFA, M.-S. M. Automated software testing framework for web applications. **International Journal of Applied Engineering Research**, v. 13, n. 11, p. 9758–9767, 2018. Citado na página 33.

JASON, H. **Selenium |About Selenium**. 2004. Disponível em: <https://www.selenium.dev/documentation/>. Citado 2 vezes nas páginas 12 e 36.

KOSCIANSKI, A.; SOARES, M. dos S. **Qualidade de Software-2ª Edição: Aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software**. [S.l.]: Novatec Editora, 2007. Citado na página 36.

LEITE, T.; VIANA, J. F. R. et al. **Testes de software: Conceitos e práticas para conquistar e manter a qualidade de software**. [S.l.]: Casa do Código, 2025. Citado na página 20.

LEOTTA, M.; CLERISSI, D.; RICCA, F.; SPADARO, C. Improving test suites maintainability with the page object pattern: An industrial case study. In: IEEE. **2013 ieee sixth international conference on software testing, verification and validation workshops**. [S.l.], 2013. p. 108–113. Citado 2 vezes nas páginas 38 e 47. Disponivel em: https://doi.org/10.1109/ICSTW.2013.19

MYERS GLENFORD J, S. C.; BADGETT, T. **The art of software testing**. [S.l.]: John Wiley & Sons, 2011. Citado 4 vezes nas páginas 11, 12, 19 e 22.

NETO, A.; CLAUDIO, D. Introdução a teste de software. **Engenharia de Software Magazine**, v. 1, p. 22, 2007. Citado na página 20.

PARVEEN REX BLACK, D. F. K. O. T. **Certified Tester Foundation Level Syllabus versão 4.0 International Software Testing Qualifications Board**. 2018. Disponível em: <https://bstqb.online/files/syllabus_ctfl_4.0br.pdf>. Citado na página 9.

PEIXOTO, R. **Selenium WebDriver: Descomplicando testes automatizados com Java**. [S.l.]: Editora Casa do Código, 2018. Citado na página 36.

PINHEIRO, V. d. S. F.; VALENTIM, N. M. C.; VINCENZI, A. M. R. Um comparativo na execuçao de testes manuais e testes de aceitaçao automatizados em uma aplicaçao web. In: SBC. **Simpósio Brasileiro de Qualidade de Software (SBQS)**. [S.l.], 2015. p. 260–267. Citado 2 vezes nas páginas 22 e 34. Disponivel em: https://doi.org/10.5753/sbqs.2015.15231

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de software-9**. [S.l.]: McGraw-Hill Brasil, 2021. Citado 2 vezes nas páginas 10 e 16.

RIOS, E. **Documentação de Teste de Software: Dissecando o padrão IEEE 829**. [S.l.]: Sao Paulo: Imagem, 2010. Citado 5 vezes nas páginas 18, 19, 22, 23 e 29.

ROCHA, B. **Bug no Nubank: clientes relatam cobranças duplicadas e saques "de graça"; veja o que diz o banco – Tempo Real – Estadão E-Investidor – As principais notícias do mercado financeiro**. São Paulo, SP, Brasil: [s.n.], 2024. Acessado em: 31 mar. 2025. Disponível em: <https://einvestidor.estadao.com.br/ultimas/bug-nubank-cobrancas-duplicadas-saques-de-graca/>. Citado na página 10.

SEMERANO, V. Z.; OLIVEIRA, L. F. D. O papel estratégico do analista de qualidade (qa) em equipes scrum, kanban e scrumban no desenvolvimento de software Ágil. **Advances in Global Innovation amp; Technology**, v. 2, n. 2, p. 32–45, mar. 2024. Disponível em: <https://revista.fateczl.edu.br/index.php/git/article/view/61>. Citado na página 13. Disponivel em: https://doi.org/10.29327/2384439.2.2-3

SMART, J. F.; MOLAK, J. **BDD in Action: Behavior-driven development for the whole software lifecycle**. [S.l.]: Simon and Schuster, 2023. Citado na página 18.

SOUZA ANDERSON FERREIRA ALVES, L. L. B. R. B. N. Gabriel Jardim Ribeiro de. Metodologias Ágeis: Explorando o impacto do scrum e do kanban na qualidade e produtividade do software. **Revista Multidisciplinar do Nordeste Mineiro**, v. 12, p. 2024, 11 2024. ISSN 2178-6925. Disponível em: <https://revista.unipacto.com.br/index.php/multidisciplinar/article/view/3060>. Citado 2 vezes nas páginas 13 e 33. https://doi.org/10.61164/rmnm.v12i2.3060

TERHORST-NORTH, D. **Desenvolvimento Orientado por Comportamento | Pepino**. 2024. Disponível em: <https://cucumber.io/docs/bdd/history>. Citado na página 18.

THANT, K. S.; TIN2, H. H. K. The impact of manual and automatic testing on software testing efficiency and effectiveness. **www.ijsronline.org**, 2023. ISSN ISSN 2583 – 2913. Disponível em: <https://www.ijsronline.org/issue/20230714-032703.942.pdf>. Citado na página 47.

THOORIQOH, H. A.; ANNISA, T. N.; YUHANA, U. L. Selenium framework for web automation testing: A systematic literature review. **Jurnal Ilmiah Teknologi Informasi**,

Sepuluh Nopember Institute of Technology, v. 19, n. 2, p. 65–76, 2021. Citado na página 35. Disponivel em: https://doi.org/10.12962/j24068535.v19i2.a1021

WAZLAWICK, R. **Engenharia de software: conceitos e práticas**. [S.l.]: Elsevier Editora Ltda., 2019. Citado 3 vezes nas páginas 24, 26 e 27.

WILLI, F. M. R. P. R. **Métodos Ágeis para Desenvolvimento de Software - Google Livros**. Bookman Editora, 2014. v. 1. ISBN 978-85-8260-208-9. Disponível em: <https://books.google.com.br/books?hl=pt-BR&lr=&id=8rQABAAAQBAJ&oi=fnd&pg=PR1&dq=metodologias+%C3%A1geis+&ots=I1shYn5ZL3&sig=afKxhavm3BttUutJjLJ86hxUkoY#v=onepage&q=metodologias%20%C3%A1geis&f=false>. Citado na página 23.

6 Apêndices

6.1 Downloads e dependências

Java JDK Link http://www.oracle.com/technetwork/pt/java/javase/downloads/jdk8-downloads-2133151.html

6.2 IntelliJ IDEA

https://www.jetbrains.com/idea/download/section=mac

6.3 JUnit https://mvnrepository.com/artifact/junit/junit/4.12

https://mvnrepository.com/artifact/junit/junit/4.12

6.4 Selenium WebDriver

https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java/3.6.0

6.5 ChromeDriver

https://sites.google.com/a/chromium.org/chromedriver/downloads

Outros Drivers

6.6 Mozilla

https://github.com/mozilla/geckodriver/releases (Firefox)

6.7 Internet Explorer

http://seleniumrelease.storage.googleapis.com/index.html?path=3.6 (Internet Explorer)