CADERNOS UniFOA 20 ANOS
Edição 56 | 2026
e-ISSN: 1982-1816
Data de submissão: 16/11/2025 Data de Aprovação: 18/05/2026

Análise de ferramentas de teste de *software* em ambientes ágeis e *pipeline* integração contínua/ entrega contínua

Analysis of software testing tools in agile environments and continuous integration/continuous delivery pipeline

[1] Matheus Vinícius Ferreira Pinheiro
[1] Alysson Filgueira Milanez
[1] Reudismam Rolim de Sousa

1 Universidade Federal Rural do Semi-Árido, Mossoró, Rio Grande do Norte.

Resumo:

Este trabalho analisa e compara ferramentas de teste de *software*, com foco em sua adequação a metodologias ágeis e ao contexto de entrega contínua. A pesquisa foi estruturada em três fases: uma revisão bibliográfica sobre ferramentas de teste de unidade, integração, funcional, desempenho, segurança e usabilidade; uma implementação experimental do comportamento delas em cenários práticos; e uma análise comparativa por critérios, como facilidade de uso, desempenho e capacidade de integração. Os resultados indicaram que ferramentas, como o *Cypress* e *Postman*, são ideais para equipes ágeis, oferecendo agilidade e simplicidade, enquanto soluções mais robustas, como *LoadRunner* e *Burp Suite*, são mais adequadas para ambientes corporativos complexos. A análise revelou que não há uma única ferramenta ideal e a escolha deve considerar os requisitos do projeto e a composição da equipe. O estudo conclui que a combinação de ferramentas, integrada a *pipelines* de integração contínua/entrega contínua (CI/CD), é a abordagem mais eficaz para garantir cobertura abrangente e maior qualidade do *software*.

Palavras-chave:

Testes de *Software*, Metodologias Ágeis, Automação de Testes, Entrega Contínua.

Abstract:

This paper analyzes and compares software testing tools, with a focus on their suitability for agile methodologies and the continuous delivery context. We structured the research in three phases: a literature review of unit, integration, functional, performance, security, and usability testing tools; an experimental implementation of their behavior in practical scenarios; and a comparative analysis based on criteria such as ease of use, performance, and integration capabilities. The results indicated that tools such as Cypress and Postman are ideal for agile teams, offering agility and simplicity, while more robust solutions, such as LoadRunner and Burp Suite, are better suited for complex enterprise environments. The analysis revealed that there is no single ideal tool; the choice should consider both project requirements and team composition. Our study concludes that a combination of tools, integrated into Continuous Integration/Continuous Delivery (CI/CD) pipelines, is the most effective approach for ensuring comprehensive coverage and higher software quality.

Keywords:

Software Testing, Agile Methodologies, Test Automation, Continuous Delivery.

CC BY NC SA

1 INTRODUÇÃO

A qualidade, segurança e confiabilidade são preocupações centrais no desenvolvimento de *software*, sobretudo na adoção de metodologias ágeis e práticas de entrega contínua (Honorato, 2022) que aceleram a criação de funcionalidades, mas impõem desafios para verificar falhas e vulnerabilidades (Dias, 2023). Assim, as ferramentas de teste de *software* são essenciais na criação de produtos. No entanto, à medida que tecnologias e práticas de desenvolvimento emergem, questiona-se a eficiência delas em demandas cada vez mais complexas e dinâmicas (Maldonado *et al*., 2016).

Mesmo com várias ferramentas de teste no mercado, a literatura apresenta lacunas na análise de testes para o desenvolvimento ágil e contínuo. Honorato (2022) e Dias (2023) apontam que, embora as ferramentas tenham evoluído, poucos trabalhos avaliam sua eficiência e adequação em diferentes testes, como segurança, usabilidade, funcionalidade, integração e desempenho. Além disso, com o ritmo acelerado das inovações tecnológicas, muitas revisões sistemáticas não abordam ferramentas recentes ou exploram sua aplicabilidade prática em ambientes de entrega contínua (Dias, 2023).

Esta pesquisa visa preencher essa lacuna, investigando a eficiência das ferramentas de teste de *software*. O estudo se concentra na análise de como essas ferramentas são aplicadas em diversos aspectos, desde a correção funcional até a segurança e usabilidade do sistema. A partir dessa análise, pretende-se identificar pontos fortes e limitações das ferramentas, contribuindo para o âmbito acadêmico e a prática profissional. Assim, a problemática central reside em entender a eficiência das ferramentas de testes em ambientes ágeis e de entrega contínua e como elas podem ser aprimoradas ou aplicadas de forma a garantir mais qualidade, segurança e usabilidade dos sistemas modernos.

Ao responder a essas questões, espera-se avançar na área e fornecer diretrizes para o uso eficaz dessas ferramentas. Para isso, foi feita uma análise crítica da eficiência delas no contexto ágil e de entrega contínua. A pesquisa busca identificar lacunas na eficiência e melhorias, contribuindo para a tomada de decisão no uso das ferramentas.

2 REFERENCIAL TEÓRICO

Esta seção apresenta os principais conceitos e teorias que embasam esta pesquisa.

2.1 Teste de software

Os *softwares* são adotados em diversas atividades, trazendo uma demanda crescente por qualidade e produtividade (Maldonado *et al*., 2016). Nesse cenário, os testes de *software* buscam reduzir a incidência de defeitos nos produtos finais, antes da entrega ao cliente (O'keeffe, 2012). Conforme Jorgensen (2013), o custo da ausência de testes de *software* pode ser significativo, impactando diretamente na qualidade do produto final, podendo também levar à insatisfação dos clientes (Honorato, 2022). A busca pela qualidade do produto perpassa os processos de verificação, validação e teste (VV&T), visando assegurar a qualidade em todas as etapas.

As ferramentas de teste desempenham um papel crucial nesse processo, desde as fases iniciais ao produto final, projetadas para diferentes finalidades de testes, como testes unitários, de integração, funcionais, de segurança e de regressão (Pressman; Maxim, 2016; Sommerville, 2011).

2.2 Tipos de teste

Conforme Maldonado *et al*. (2016), o processo de teste de *software* divide-se em quatro etapas principais: planejamento, projeto, execução e avaliação. Essas etapas são fundamentais para a qualidade do *software*, aplicadas em três metodologias principais de teste: caixa branca, caixa preta e caixa cinza (Maldonado *et al*., 2016).

O teste de caixa branca trabalha com o código-fonte do componente e avalia aspectos como teste de condição, fluxo de dados, ciclos e caminhos lógicos, para verificar o seu comportamento interno (Pressman; Maxim, 2016; Maldonado *et al*., 2016). O teste de caixa preta analisa o comportamento do sistema para encontrar problemas, tais como funções incorretas ou ausentes, erros de *interface*, erros nas estruturas de dados, etc. (Pressman e Maxim, 2016). Os testes de caixa cinza combinam características dos testes de caixa branca e caixa preta, acessando documentos, tais como requisitos e banco de dados, permitindo criar testes mais completos (Acharya; Pahdya, 2021).

Dentre os tipos de teste, o de unidade concentra-se nas unidades menores do programa, como funções, procedimentos, métodos ou classes (Delamaro; Maldonado; Jino, 2016). Já no teste de integração, os módulos ou componentes são combinados e testados em conjunto (Pressman; Maxim, 2016). Os testes funcionais consideram o programa ou sistema como uma "caixa-preta", para verificar se os requisitos foram implementados e se o sistema executa suas funções corretamente (Delamaro; Maldonado; Jino, 2016). De outro modo, os testes de desempenho e carga avaliam a capacidade e eficiência de sistemas em diferentes condições operacionais (Sommerville, 2011). O teste de segurança verifica se os mecanismos de proteção do sistema vão de fato protegê-lo (Beizer, 1984). Já o teste de usabilidade, avalia a interação entre o sistema e o usuário, verificando a eficiência, a eficácia e a satisfação do usuário ao utilizar a aplicação (Delamaro; Maldonado; Jino, 2016). Por sua vez, os testes de regressão asseguram que novas alterações não introduzem defeitos (Pressman; Maxim, 2016).

2.3 Ferramentas para teste de unidade

O *JUnit*, um *framework* para testes unitários em Java, permite a criação de classes de teste (*class driver*), para exercitar os métodos da classe-alvo. Cada classe de teste é isolada por métodos com asserções para verificar os resultados esperados (Rocha, 2003). O *NUnit*[2] é um *framework* de teste de unidade para o *.NET*, que permite escrever e executar testes em *C#* e *VB.NET*. Semelhante ao *JUnit*, ele simplifica a criação de testes unitários para verificar o comportamento esperado das classes e métodos. O *PyUnit*[3] é um *framework* de teste de unidade para *Python*, integrado à biblioteca padrão da linguagem, necessitando apenas importar a biblioteca *unittest*.

2.4 Ferramentas para teste de integração

Postman[4] é uma plataforma para testes de *Application Programming Interface* (API), sendo um ambiente simples e integrado para a criação, envio e análise de requisições *HTTP*. A *interface* gráfica permite configurar parâmetros, cabeçalhos e corpo de requisições para chamadas a APIs. O *SoapUI*[5] é uma ferramenta Java para testar serviços *Web*. SOAP (*Simple Object Access Protocol*) é um padrão que usa *XML* para formatar dados, conhecido por sua rigidez e segurança, ideal para transações complexas. *Rest-Assured* facilita o desenvolvimento de testes automatizados para APIs *REST*, suportando a validação de requisições *HTTP*

2 https://www.c-sharpcorner.com/article/introduction-to-nunit-testing-framework/

3 https://www.devmedia.com.br/teste-unitario-com-pyunit/41233

4 https://enotas.com.br/blog/postman/

5 https://www.devmedia.com.br/soapui-testes-de-web-services-rapido-e-descomplicado/37461

com *JSON*. *REST* (*Representational State Transfer*) é uma arquitetura que utiliza métodos *HTTP* (*GET, POST, PUT, DELETE*) para comunicação *Web*, oferecendo simplicidade e flexibilidade na integração de sistemas. O *Rest-Assured* funciona como um cliente, oferece opções de validação, incluindo cabeçalhos (*Headers*), corpo da requisição (*Body*) e código de estado (*Status Code*). A ferramenta utiliza a sintaxe *GHERKIN* (*Given, When, Then*), dividindo o código em seções para demonstrar seu comportamento (Souza, 2024).

2.5 Ferramentas para teste funcional

O *Selenium*[6] é um *framework* gratuito para testes funcionais de aplicações *Web*. Permite a execução de testes em uma variedade de navegadores e plataformas, oferecendo uma abordagem flexível e eficaz para a validação de funcionalidades de sistemas *Web*. Além disso, o *Selenium* suporta várias linguagens de programação, como *Java, C#, Python* e *Ruby* (Holmes; Kellogg, 2006). *Cypress* é uma ferramenta de testes *end-to-end* de aplicações *Web,* utilizando *JavaScript*, permitindo a criação, depuração e execução de testes no navegador, facilitando a configuração e eliminando o uso de pacotes adicionais (Mwaura, 2021). Além disso, *Cypress* suporta testes em múltiplos navegadores e pode ser integrado a *pipelines* de CI/CD, permitindo a execução de testes em ambientes diversos e facilitando a manutenção de um fluxo de desenvolvimento contínuo e ágil (Mwaura, 2021). *Appium*[7] é uma ferramenta de teste de aplicativos móveis que utiliza o protocolo *WebDriver*. Esse protocolo define a comunicação entre navegadores e aplicativos usando uma linguagem comum. O *Appium* suporta as plataformas móveis *iOS, Android, Windows* e *Mac*. Com abordagem *write once, run anywhere*, a escrita dos testes é feita em uma única linguagem de programação, como *Ruby, Java, Node.js, PHP, C#, Clojure* e *Perl* para execução em diferentes plataformas.

2.6 Ferramentas para teste não funcional

O *Apache JMeter* é uma ferramenta baseada em *Java* para testar e medir o comportamento funcional de aplicativos cliente/servidor, o desempenho e a carga de aplicativos *Web* (Halili, 2008). Sua *interface* gráfica permite configurar testes, monitorar recursos e gerar relatórios, facilitando a identificação de problemas e gargalos de desempenho. Sua flexibilidade a torna versátil para várias necessidades de teste, permitindo a integração com outras ferramentas e *scripts* personalizados (Halili, 2008). O *Gatling*[8] é uma ferramenta de código aberto para testar o desempenho e a carga. É escrita em *Scala*, sendo possível escrever código diretamente na aplicação. É comumente usado para simular grandes quantidades de tráfego em aplicações *Web*. *Gatling* se destaca por sua capacidade de realizar testes complexos e escaláveis com alta eficiência. Foi desenvolvido com a biblioteca *Akka* e suporta protocolos como *HTTP, WebSockets* e *Jakarta Message Service* — JMS. O *LoadRunner*[9] é uma ferramenta muito utilizada para testes de carga, desempenho e estresse, permitindo simular múltiplos usuários simultâneos em um sistema para avaliar seu comportamento sob condições de carga realistas. Com *scripts* que imitam os usuários, coleta dados sobre o desempenho, identificando gargalos no sistema. A ferramenta é compatível com plataformas e tecnologias, incluindo *Web*, *mobile* e aplicativos empresariais, e suporta uma abordagem em nuvem, permitindo testes escaláveis e flexíveis em diferentes ambientes de desenvolvimento.

2.7 Ferramentas para teste de segurança

O *OWASP ZAP* (*Zed Attack Proxy*)[10] é uma ferramenta de testes de segurança de aplicações *Web*, criada pelo *Open Web Application Security Project* (OWASP). O *ZAP* identifica vulnerabilidades em aplicações *Web*, oferecendo funcionalidades, como análise passiva e ativa, varredura de portas, *fuzzing* e interceptação de trá-

6 https://hnz.com.br/selenium-saiba-o-que-e-suas-vantagens-e-como-utilizar/

7 https://testgrid.io/blog/appium-testing-tutorial/

8 https://www.loadview-testing.com/pt-br/blog/teste-de-carga-gatling-como-fazer-testes-distribuidos-e-exemplos/

9 https://www.loadview-testing.com/pt-br/comparar/loadrunner-vs-loadview/.

10 https://community.revelo.com.br/owasp-zap-ferramenta-de-seguranca-de-aplicativos-da-web/

fego *HTTP/HTTPS*. *Burp Suite*[11] é um conjunto de ferramentas de testes de segurança, com foco em testes de penetração e segurança de sistemas *Web*. Oferece funcionalidades que permitem analisar a comunicação entre o cliente e o servidor, detectar vulnerabilidades e avaliar a resistência da aplicação contra possíveis ataques. O *Nessus* é um *scanner* de vulnerabilidades utilizado na identificação de falhas de segurança em sistemas, redes e aplicações, auxiliando na detecção de configurações incorretas, falhas de *software* e outros pontos de exposição em ambientes computacionais (Tenable, [s. d.]). Em estudos comparativos, a ferramenta também é analisada ao lado de soluções como *OpenVAS* e *Nmap Scripting Engine*, especialmente em processos de avaliação de riscos e automação da análise de vulnerabilidades (Chalvatzis; Karras; Papademetriou, 2019).

2.8 Ferramentas para teste de usabilidade

UsabilityHub[12] é uma plataforma *on-line* com ferramentas para testes de usabilidade focados em *design*. Essas ferramentas coletam *feedback* de usuários reais sobre diversos aspectos de *interfaces*, incluindo *layouts*, logotipos e materiais de marketing. Entre as ferramentas disponíveis no *UsabilityHub,* estão o *Five Second Test*, o *Click Test*, o *Question Test*, o *Navigation Test* e o *Preference Test*.

2.9 Ferramentas para teste de regressão

O *TestNG*[13] é um *framework* para testes utilizado em *Java*, inspirado no *JUnit*, com funções avançadas e flexíveis para criar e executar testes. Anotações como *@Test, @BeforeMethod* e *@AfterMethod* permitem configurar e gerenciar eficientemente o ciclo de vida dos testes. O *pytest*[14] é um *framework* de teste para *Python*, conhecido pela simplicidade na criação de testes. A sintaxe intuitiva e o suporte a testes unitários e funcionais (incluindo testes de regressão) facilitam escrever e executar testes, permitindo foco na lógica de teste ao invés da configuração do *framework*.

2.10 Os testes em ambientes ágeis

Nos ambientes ágeis, a automação de testes é crucial para que novas entregas sejam feitas com rapidez e segurança (Maldonado *et al*., 2016). O uso de ferramentas automatizadas reduz custos e melhora a qualidade do *software* (Honorato, 2022). A natureza iterativa e incremental das metodologias ágeis exige que os testes acompanhem o ritmo acelerado das entregas. Nesse contexto, a integração de testes contínuos ao *pipeline* de desenvolvimento permite detectar falhas mais cedo, facilitar a refatoração e promover maior confiança em modificações frequentes (Maldonado *et al*., 2016). Além disso, a colaboração entre desenvolvedores, testadores e demais membros da equipe é fundamental para a cobertura e a eficiência dos testes em cada *sprint* (Piovesan, 2018).

2.11 Metodologias ágeis e entrega contínua

As metodologias ágeis revolucionaram o desenvolvimento de *software,* ao priorizar entregas incrementais, colaboração e flexibilidade frente a mudanças (Sommerville, 2011). Promovem ciclos de desenvolvimento rápidos e adaptáveis, favorecendo a entrega constante de funcionalidades em pequenos incrementos. Nesse contexto, práticas como a integração contínua (*Continuous Integration* - CI) e a entrega contínua (*Continuous Delivery* - CD) tornaram-se fundamentais (Shahin; Babar; Zhu, 2017) para que o *software* esteja em um estado funcional, pronto para ser entregue ao cliente, permitindo uma rápida identificação e correção de defeitos e um fluxo contínuo de testes automatizados, para assegurar a qualidade do código (Maldonado *et al*., 2016).

11 https://community.revelo.com.br/burp-suite-parte-i/

12 https://brasil.uxdesign.cc/3-maneiras-rápidas-e-eficientes-de-testar-a-usabilidade-do-seu-produto-parte-i-22251c187eb2

13 https://www.geeksforgeeks.org/testng-tutorial/

14 https://didatica.tech/tudo-sobre-a-biblioteca-pytest-aprenda-na-pratica/

3 TRABALHOS RELACIONADOS

Embora a literatura avance com ferramentas de teste, há desafios não solucionados, como a dificuldade do ajuste rápido ao desenvolvimento ágil. Além disso, muitas ferramentas novas, como *Cypress* e *RestAssured*, possuem uma documentação nova, e os estudos sobre elas são limitados. A maioria das pesquisas sobre ferramentas de teste foca em soluções antigas e estabelecidas no mercado, como *JUnit* e *Selenium*, com documentação mais robusta, criando uma lacuna de como integrar o *Cypress* e *RestAssured* em CI/CD e testes em ambientes distribuídos, dificultando sua adoção em fluxos de trabalho ágeis. Delamaro, Maldonado e Jino (2016) e Pressman e Maxim (2016) sugerem que muitas ferramentas atuais não conseguem acompanhar a velocidade das mudanças no desenvolvimento ágil. Além disso, Rajapakse *et al*. (2022) apontam que a integração de práticas e ferramentas de segurança em ambientes *DevOps* ainda representa um desafio, especialmente pela necessidade de equilibrar velocidade de entrega, automação e controle de vulnerabilidades.

Os trabalhos discutidos contribuem para entender e usar ferramentas de teste em diferentes contextos, mas muitos deles se limitam a abordagens teóricas ou a análises pontuais de ferramentas, sem comparar categorias distintas (Hamilton, 2004; Rocha, 2003; Holmes; Kellogg, 2006; Amankwah *et al*., 2020; Mwaura, 2021). Em especial, são raros os estudos que conectam essas ferramentas a um fluxo completo de desenvolvimento ágil e entrega contínua, contexto em que a automação de testes se torna ainda mais crítica para garantir qualidade e velocidade nas entregas (Honorato, 2022; Dias, 2023).

Adicionalmente, muitos trabalhos não abordam de forma integrada diferentes testes — como testes de unidade, integração, funcionais, desempenho, segurança e usabilidade — tampouco analisam a interoperabilidade entre as ferramentas e os *pipelines* de CI/CD, fator fundamental em ambientes *DevOps* modernos. Como discutido por Giannecchini (2018), a integração contínua é importante, ao possibilitar reduzir falhas ao entregar o *software* para o cliente.

Diferentemente destes, esta pesquisa propõe uma análise sistemática, fundamentada e aplicada, para preencher essas lacunas ao comparar diversificadas ferramentas por critérios práticos, como facilidade de uso, curva de aprendizagem, integração com *pipelines*, etc. A proposta responde a demandas apontadas por autores como Delamaro, Maldonado e Jino (2016), que destacam a carência de estudos práticos na escolha consciente de ferramentas de teste; e Pressman e Maxim (2016), que apontam desafios da engenharia de *software* frente à diversidade e evolução acelerada das ferramentas de suporte.

Além disso, este trabalho reforça o papel de ferramentas de segurança e usabilidade, frequentemente tratadas de forma secundária nos fluxos de desenvolvimento contínuo, propondo uma visão holística e alinhada à adoção de metodologias ágeis (Rajapakse *et al*., 2022). Assim, oferece um panorama atualizado e pragmático da adequação das ferramentas analisadas, contribuindo para a tomada de decisão técnica e para a definição de estratégias de qualidade em projetos de *software*. A Tabela 1 apresenta um comparativo entre os trabalhos relacionados e a proposta deste estudo.

Tabela 1 - Comparação entre trabalhos relacionados e o presente trabalho

Aspecto	Trabalhos relacionados	Proposta
Testes de Unidade	Delamaro et al. (2016), Jorgensen (2013)	Sim
Testes Funcionais	Holmes e Kellogg (2006), Mwaura (2021)	Sim
Testes de Segurança	Chalvatzis, Karras e Papademetriou (2019); Rajapakse et al. (2022); Pressman e Maxim (2016) (parcial)	Sim
Integração CI/CD	Dias (2023)	Sim
Análise Prática	O'Keeffe (2012)	Sim
Testes Diversos	Pressman e Maxim (2016), Honorato (2022), Delamaro et al. (2016)	Sim

Fonte: Elaborada pelos autores (2026)

4 METODOLOGIA

A metodologia adotada foi estruturada em três fases: pesquisa bibliográfica, implementação experimental e análise comparativa.

Na primeira fase, realizou-se uma pesquisa bibliográfica a partir de livros, artigos científicos, dissertações, trabalhos acadêmicos e documentações técnicas, contemplando as principais categorias de testes de *software*: testes unitários, testes de integração, testes funcionais, testes de desempenho e carga, testes de segurança e testes de usabilidade

Na segunda fase, foi realizada a implementação experimental das ferramentas selecionadas. Os testes foram executados, em sua maioria, em ambiente local, utilizando o editor *Visual Studio Code*, em um computador com sistema operacional *Windows 11*, processador *AMD Ryzen 5 2600* e 16 GB de memória RAM. O repositório utilizado para organização dos códigos, arquivos e configurações dos testes foi disponibilizado publicamente no *GitHub*, sob o nome TCC_Eng.-Software, com estrutura organizada por ferramentas e categorias de teste, incluindo diretórios como *Appium*, *C#/NUnit*, *Cypress*, *Java/JUnit*, *Postman*, *Python/PyUnit*, *RestAssured*, *Selenium* e *SoapUI*. O próprio repositório apresenta, como objetivo, analisar técnicas e ferramentas de teste de *software* aplicadas à segurança, usabilidade, testes funcionais, integração e desempenho, com foco na qualidade e segurança ao longo do ciclo de desenvolvimento e entrega contínua.

A implementação experimental contemplou a configuração e execução de ferramentas de diferentes categorias. Para os testes unitários, foram utilizados *JUnit*, *NUnit* e *PyUnit*, aplicados à validação de unidades de código. No caso do *JUnit*, por exemplo, foram elaborados casos de teste para verificar operações básicas, como soma, subtração, multiplicação, divisão e tratamento de divisão por zero. A eficiência foi observada a partir do tempo de execução, da ausência de falhas inesperadas e da capacidade da ferramenta de indicar corretamente os resultados esperados.

Para os testes de integração, foram utilizados *Postman*, *SoapUI* e *RestAssured*, com o objetivo de validar a comunicação entre componentes, especialmente por meio de chamadas a APIs e serviços. Nessa etapa, observaram-se critérios como facilidade de configuração das requisições, clareza dos resultados, tempo de resposta e identificação de falhas na comunicação entre serviços.

A avaliação funcional foi realizada com *Selenium*, *Cypress* e *Appium*, aplicados à automação de interações com interfaces *web* e *mobile*. Embora cada ferramenta apresente uma forma própria de configuração,

buscou-se manter padrões semelhantes de interação, contemplando os mesmos fluxos de navegação e elementos da interface sempre que possível.

Para a análise de desempenho e carga, foram consideradas ferramentas como *JMeter*, *Gatling* e *LoadRunner*, com o propósito de avaliar o comportamento da aplicação sob diferentes condições de uso. Foram analisados aspectos como tempo médio de resposta, estabilidade da aplicação, capacidade de lidar com execuções sequenciais e comportamento diante de múltiplas requisições.

A verificação de segurança foi conduzida com *OWASP ZAP*, *Burp Suite* e *Nessus*, ferramentas voltadas à identificação de vulnerabilidades em aplicações, sistemas e ambientes *web*. A análise considerou a capacidade de detecção de falhas, clareza dos relatórios, facilidade de configuração das varreduras e classificação dos riscos identificados.

Por fim, a avaliação de usabilidade contemplou as ferramentas *UsabilityHub* e *UserTesting*, com foco na experiência do usuário durante a interação com as interfaces. Nessa etapa, foram observados critérios como clareza da navegação, facilidade de compreensão das funcionalidades e percepção geral sobre a interface analisada.

Na terceira fase, as ferramentas foram comparadas a partir dos seguintes critérios: facilidade de instalação e configuração, facilidade de uso, clareza da documentação, tempo médio de execução, qualidade dos relatórios gerados, capacidade de identificação de falhas, flexibilidade, customização e compatibilidade com fluxos de desenvolvimento ágil e integração contínua/entrega contínua.

Para reduzir a subjetividade da análise comparativa, os critérios qualitativos utilizados nas tabelas foram classificados com base em uma escala operacional de quatro níveis de adequação: Nível 1 — baixa adequação, Nível 2 — adequação moderada, Nível 3 — boa adequação e Nível 4 — alta adequação. Essa escala foi definida a partir da observação prática das ferramentas durante a instalação, configuração, execução dos testes, estabilidade, clareza dos relatórios, eficiência dos resultados e adequação ao contexto de desenvolvimento ágil e integração contínua. Dessa forma, as classificações não representam medidas absolutas, mas uma avaliação comparativa dentro do ambiente experimental adotado neste estudo.

A Tabela 2 apresenta os critérios de avaliação operacional das ferramentas utilizadas, organizados em uma escala de adequação que varia de Nível 1 (baixa adequação) ao Nível 4 (alta adequação), conforme descrito no texto. Cada nível foi atribuído com base nas observações práticas realizadas no cenário experimental adotado.

Tabela 2 - Classificação dos Níveis de Adequação Operacional das Ferramentas Utilizadas — Os critérios de avaliação são organizados em uma escala de Nível 1 (baixa adequação) ao Nível 4 (alta adequação).

Nível	Classificação	Critério operacional
Nível 1	Baixa adequação	Apresenta limitações relevantes de instalação, configuração, uso, estabilidade, documentação ou interpretação dos resultados, dificultando sua aplicação prática no cenário analisado.
Nível 2	Adequação moderada	Apresenta funcionamento aceitável, mas exige maior esforço técnico, adaptação, configuração ou conhecimento prévio para uso adequado.
Nível 3	Boa adequação	Apresenta funcionamento satisfatório, com boa estabilidade, resultados claros e limitações pontuais que não comprometem a aplicação prática.
Nível 4	Alta adequação	Apresenta melhor desempenho dentro da categoria analisada, com maior facilidade de uso, estabilidade, eficiência, clareza dos resultados e aderência ao cenário de teste.

Fonte: Elaborada pelos autores (2026)

Ressalta-se que o nível geral de adequação atribuído a cada ferramenta não corresponde a uma média aritmética dos critérios avaliados. A classificação final considerou a aderência da ferramenta ao cenário experimental analisado, sua aplicabilidade prática, suas limitações operacionais e sua compatibilidade com o tipo de teste em questão.

A avaliação da facilidade de uso considerou três aspectos principais: simplicidade na instalação, clareza da documentação e curva de aprendizado inicial. O desempenho foi analisado a partir do tempo médio de execução, da estabilidade ao longo das execuções e da capacidade de lidar com testes sequenciais. Já a eficiência, foi avaliada considerando a relação entre esforço de configuração, qualidade dos resultados obtidos e aplicabilidade prática da ferramenta no contexto de desenvolvimento de *software*.

5 RESULTADOS E DISCUSSÕES

Nesta seção, são apresentados os resultados práticos, comparando as ferramentas em diversos critérios e discutindo as implicações dos resultados obtidos.

Os testes, os *scripts* e arquivos de configuração estão disponíveis no repositório do projeto no *GitHub*[15] , visando assegurar a reprodutibilidade dos experimentos e permitir a consulta dos procedimentos realizados e contribuir para a validação dos resultados. A execução dos testes experimentais foi baseada nas ferramentas descritas na Seção 2.

5.1 Testes de unidade

De acordo com a Tabela 3, as ferramentas *JUnit*, *NUnit* e *PyUnit* apresentaram os resultados dos testes de uma calculadora, envolvendo operações aritméticas básicas (soma, subtração, multiplicação, divisão e tratamento de erro para divisão por zero). Foram executados entre 5 e 10 casos de teste por operação em cada ferramenta, caracterizando um cenário de porte pequeno a médio devido à simplicidade e à quantidade reduzida de testes, resultando em um baixo tempo de execução.

15 https://github.com/matheusvfp/TCC_Eng.-Software

Tabela 3 - Resultados dos Testes de Unidade

Ferramenta	Tempo de execução médio	Resultado dos casos de teste	Nível de adequação
JUnit	0,03 s	Casos executados com sucesso	Nível 3 — Boa adequação
NUnit	0,03 s	Casos executados com sucesso	Nível 4 — Alta adequação
PyUnit	0,01 s	Casos executados com sucesso	Nível 4 — Alta adequação

Fonte: Elaborada pelos autores (2026)

Os testes foram realizados no ambiente de desenvolvimento *Visual Studio Code*, uniformizando a análise, sendo escolhido pela ampla adoção por desenvolvedores[16] .

O *JUnit*, o *NUnit* e o *PyUnit* apresentaram desempenho semelhante quanto à execução dos casos de teste, com baixo tempo de processamento e resultados consistentes. Entretanto, foram observadas diferenças quanto à configuração inicial das ferramentas. O *JUnit*, embora maduro e amplamente utilizado, exigiu maior atenção na preparação do ambiente Java, o que elevou o esforço de configuração em comparação às demais ferramentas. Por esse motivo, foi classificado como Nível 3 — Boa adequação. O *NUnit* e o *PyUnit*, por apresentarem configuração mais direta no cenário experimental adotado e execução satisfatória dos testes, foram classificados como Nível 4 — Alta adequação.

5.2 Testes de integração

O *Postman* destacou-se pela acessibilidade e pela interface gráfica intuitiva, sendo adequado para a criação, execução e validação de requisições em APIs REST, especialmente em testes manuais e exploratórios. O *SoapUI* apresentou maior robustez em testes envolvendo serviços *SOAP*, embora tenha demonstrado maior curva de aprendizado e maior esforço de configuração. Já o *RestAssured,* destacou-se pela maior aderência a cenários de automação, por permitir a construção de testes diretamente em Java e sua integração com ferramentas, como *JUnit*, *Maven* e *Gradle*.

Na avaliação dos testes de integração, foram considerados três critérios principais: facilidade de configuração, clareza dos resultados e suporte à automação em fluxos de integração contínua e entrega contínua. Conforme apresentado na Tabela 4, o *Postman* apresentou boa adequação devido à simplicidade de uso e à clareza de sua interface. O *SoapUI* também foi classificado com boa adequação, principalmente pela robustez em testes de serviços, apesar da configuração menos intuitiva. O *RestAssured* recebeu a maior classificação por sua compatibilidade com testes automatizados e *pipelines* de CI/CD, embora demande maior conhecimento técnico para implementação.

Tabela 4 - Resultados dos Testes de Integração

Ferramenta	Facilidade de configuração	Clareza dos resultados	Suporte à automação/CI/CD	Nível de adequação
Postman	Nível 4	Nível 4	Nível 3	Nível 3 — Boa adequação
SoapUI	Nível 2	Nível 3	Nível 2	Nível 3 — Boa adequação
RestAssured	Nível 2	Nível 3	Nível 4	Nível 4 — Alta adequação

Fonte: Elaborada pelos autores (2026)

16 https://survey.stackoverflow.co/2024

Em contextos de CI/CD, o *RestAssured* destacou-se como a ferramenta mais adequada para testes de integração, ainda que exija conhecimento prévio em programação. Por se tratar de uma biblioteca de testes em Java, permite automação completa e integração com ferramentas amplamente utilizadas em projetos de *software*, como *JUnit*, *Maven* e *Gradle*. Essa característica favorece sua aplicação em ambientes de desenvolvimento ágil e integração contínua, nos quais a validação automatizada de APIs contribui para a identificação antecipada de falhas. Além disso, a adoção do Java como linguagem de implementação reforça a aplicabilidade do *RestAssured* em contextos corporativos, considerando que o Java permanece entre as linguagens de programação mais utilizadas por desenvolvedores, conforme dados do *Stack Overflow Developer Survey 2024* (Stack Overflow, 2024).

5.3 Testes funcionais

De acordo com a Tabela 5, o *Selenium*, o *Cypress* e o *Appium* foram avaliados quanto à facilidade de configuração e uso, desempenho na execução e aderência ao cenário funcional analisado. O *Selenium* demonstrou boa adequação para automação de aplicações *web*, embora tenha exigido maior atenção na configuração e apresentado limitações em interfaces mais dinâmicas. O *Cypress* destacou-se pela execução rápida dos testes, pelo *feedback* visual e pela integração com projetos baseados em JavaScript, apresentando maior aderência ao cenário avaliado. Já o Appium, mostrou-se relevante por sua aplicação em testes de interfaces móveis, porém sua configuração foi mais complexa e sua avaliação ocorreu em um contexto de página *web* adaptada, o que limitou parcialmente a análise de seu potencial em aplicações móveis nativas.

Tabela 5 - Resultados dos Testes Funcionais

Ferramenta	Facilidade de configuração e uso	Desempenho na execução	Aderência ao cenário avaliado	Nível de adequação
Selenium	Nível 3	Nível 3	Nível 3	Nível 3 — Boa adequação
Cypress	Nível 4	Nível 4	Nível 4	Nível 4 — Alta adequação
Appium	Nível 2	Nível 3	Nível 3	Nível 3 — Boa adequação

Fonte: Elaborada pelos autores (2026)

Nos testes funcionais, o *Cypress* apresentou o melhor desempenho comparativo, especialmente pela facilidade de configuração, execução rápida, *feedback* visual e integração nativa com JavaScript, o que favoreceu sua classificação como Nível 4 — alta adequação. O *Selenium* também se mostrou eficaz para automação de aplicações *web*, sendo uma ferramenta madura e amplamente utilizada, porém apresentou maior complexidade de configuração e algumas limitações em aplicações com interfaces mais dinâmicas, sendo classificado como Nível 3 — boa adequação. O *Appium*, por sua vez, destacou-se pela versatilidade em testes de interfaces móveis, com compatibilidade com diferentes linguagens de programação e plataformas. Entretanto, sua configuração mostrou-se mais complexa e demorada, além de ter sido aplicado em um contexto de página *web* adaptada, o que limitou parcialmente a avaliação de seu potencial em aplicações móveis nativas. Por esse motivo, também foi classificado como Nível 3 — boa adequação.

5.4 Testes de desempenho

Conforme demonstrado na Tabela 6, as ferramentas *JMeter*, *Gatling* e *LoadRunner* foram avaliadas quanto à capacidade de simulação de carga, flexibilidade dos cenários e facilidade de configuração e uso. O *JMeter* apresentou boa flexibilidade para criação e personalização de cenários de teste, porém sua interface gráfica e sua configuração podem dificultar a adoção por usuários iniciantes. O *Gatling* destacou-se pelo

melhor equilíbrio entre desempenho, clareza dos *scripts* e capacidade de simulação de múltiplas requisições, sendo classificado como Nível 4 — alta adequação. Já o *LoadRunner,* apresentou elevada robustez e maior potencial de escalabilidade, mas sua complexidade de uso e seu custo tornam sua aplicação mais adequada a ambientes corporativos de grande porte, justificando sua classificação como Nível 3 — boa adequação, no contexto experimental deste estudo.

Tabela 6 - Resultados dos Testes de Desempenho e Carga

Ferramenta	Capacidade de simulação de carga	Flexibilidade dos cenários	Facilidade de configuração e uso	Nível de adequação
JMeter	Nível 3	Nível 4	Nível 2	Nível 3 — Boa adequação
Gatling	Nível 4	Nível 3	Nível 3	Nível 4 — Alta adequação
LoadRunner	Nível 4	Nível 4	Nível 2	Nível 3 — Boa adequação

Fonte: Elaborada pelos autores (2026)

Nos testes de desempenho e carga, o *JMeter* mostrou-se eficaz na simulação de múltiplos usuários e na personalização dos cenários, mas apresentou menor facilidade de uso devido à interface menos intuitiva e à necessidade de configuração mais cuidadosa. O *Gatling* apresentou desempenho superior no cenário analisado, com *scripts* mais claros e boa capacidade para simulação de carga, embora exija conhecimento técnico para melhor aproveitamento. O *LoadRunner* destacou-se pela robustez, escalabilidade e suporte a cenários mais complexos, sendo indicado para grandes corporações. No entanto, seu custo e sua complexidade operacional reduzem sua adequação em contextos acadêmicos, experimentais ou em equipes com menor maturidade técnica.

5.5 Testes de segurança

O *OWASP ZAP* destacou-se pela interface gráfica acessível e pela facilidade de configuração, permitindo a realização de varreduras automatizadas em aplicações *web* de forma mais intuitiva. Essa característica favorece sua utilização por usuários menos experientes em testes de segurança, especialmente em contextos acadêmicos ou em equipes em fase inicial de adoção de práticas de segurança. O *Burp Suite* apresentou maior robustez e funcionalidades avançadas, sendo mais adequado para análises aprofundadas e testes de penetração, embora demande maior conhecimento técnico. Já o *Nessus,* mostrou-se relevante para análises de vulnerabilidades em ambientes de rede e infraestrutura, complementando os testes voltados a aplicações *web*. A comparação entre as ferramentas está sintetizada na Tabela 7.

Tabela 7 - Resultados dos Testes de Segurança

Ferramentwa	Capacidade de detecção	Clareza dos relatórios	Acessibilidade operacional	Nível de adequação
OWASP ZAP	Nível 3	Nível 3	Nível 4	Nível 4 — Alta adequação
Burp Suite	Nível 4	Nível 4	Nível 2	Nível 3 — Boa adequação
Nessus	Nível 3	Nível 3	Nível 2	Nível 3 — Boa adequação

Fonte: Elaborada pelos autores (2026)

Nos testes de segurança, o *OWASP ZAP* apresentou o melhor equilíbrio entre recursos disponíveis, facilidade de configuração e acessibilidade operacional, sendo classificado como Nível 4 — alta adequação no cenário analisado. O *Burp Suite* demonstrou maior profundidade técnica e maior capacidade para análises avançadas, porém sua curva de aprendizado e a complexidade de uso justificam sua classificação como Nível 3 — boa adequação. O *Nessus*, por sua vez, mostrou-se adequado para identificação de vulnerabilidades em infraestrutura, servidores e ambientes de rede, complementando os testes voltados a aplicações web. Entretanto, sua adoção em contextos institucionais ou corporativos pode depender de licença comercial/profissional, já que a versão gratuita possui limitações de uso. Por essa razão, embora apresente boa capacidade técnica, sua acessibilidade operacional foi classificada como Nível 2, mantendo sua avaliação geral como Nível 3 — boa adequação.

5.6 Testes de usabilidade

O *UsabilityHub* foi avaliado pela sua capacidade de fornecer *feedback* rápido sobre aspectos específicos da interface, como *layouts*, preferências visuais e fluxos de navegação. Já o *UserTesting,* foi analisado pela possibilidade de oferecer uma avaliação mais aprofundada da experiência do usuário, com relatórios mais detalhados e *feedbacks* qualitativos. A comparação entre as ferramentas considerou a velocidade de obtenção dos resultados, a profundidade da análise e a facilidade de aplicação no cenário experimental adotado, conforme apresentado na Tabela 8.

Tabela 8 - Resultados dos Testes de Usabilidade

Ferramenta	Velocidade de Feedback	Profundidade da Análise	Facilidade de aplicação	Nível de adequação
UsabilityHub	Nível 4	Nível 2	Nível 4	Nível 3 — Boa adequação
UserTesting	Nível 3	Nível 4	Nível 3	Nível 4 — Alta adequação

Fonte: Elaborada pelos autores (2026)

Nos testes de usabilidade, o *UsabilityHub* mostrou-se adequado para avaliações rápidas e pontuais, especialmente em fases iniciais de design, quando o objetivo é obter retornos sobre elementos visuais, escolhas de *layout* e fluxos simples de navegação. No entanto, sua análise tende a ser menos aprofundada, quando comparada a ferramentas que permitem maior exploração da experiência do usuário. Por esse motivo, foi classificado como Nível 3 — boa adequação. O *UserTesting*, por sua vez, apresentou maior profundidade analítica, permitindo uma compreensão mais detalhada da interação do usuário com a interface, embora demande maior organização para aplicação e análise dos dados. Assim, foi classificado como Nível 4 — alta adequação no cenário avaliado.

5.7 Eficiência das ferramentas

A análise comparativa da eficiência das ferramentas demonstrou que cada categoria de teste apresenta níveis distintos de adequação, conforme o contexto de aplicação, a facilidade de configuração, a clareza dos resultados e a aderência a ambientes de desenvolvimento ágil e integração contínua. Nos testes de unidade, o *NUnit* e o *PyUnit* apresentaram maior adequação no cenário experimental, devido à configuração mais direta, baixo tempo de execução e execução satisfatória dos casos de teste. O *JUnit* também apresentou resultados consistentes, porém exigiu maior atenção na preparação do ambiente Java, o que justificou sua classificação em nível inferior em relação às demais ferramentas da categoria.

Nos testes de integração, o *Postman* mostrou-se adequado para testes manuais e exploratórios de APIs REST, pela facilidade de uso e clareza da interface. O *SoapUI* demonstrou maior robustez para testes envolvendo serviços SOAP, embora com maior curva de aprendizado. O *RestAssured* apresentou maior aderência

a ambientes automatizados, especialmente por permitir integração com ferramentas como *JUnit*, *Maven* e *Gradle*, sendo mais indicado para fluxos de CI/CD.

Em relação aos testes funcionais, o *Cypress* apresentou maior eficiência no cenário analisado, com execução rápida, *feedback* visual e integração com projetos baseados em JavaScript. O *Selenium* manteve-se como uma alternativa consistente para automação *web*, embora demande maior configuração em interfaces mais dinâmicas. O *Appium* destacou-se pela versatilidade em testes móveis, mas sua configuração mais complexa e o uso em uma página *web* adaptada limitaram parcialmente sua avaliação.

Nos testes de desempenho e carga, o *JMeter* apresentou boa flexibilidade na criação de cenários, mas sua interface menos intuitiva pode dificultar a adoção por usuários iniciantes. O *Gatling* demonstrou melhor equilíbrio entre capacidade de simulação de carga, clareza dos *scripts* e desempenho no cenário experimental. O *LoadRunner* mostrou-se robusto e escalável, porém seu custo e complexidade operacional reduzem sua adequação em contextos acadêmicos ou em equipes com menor maturidade técnica.

Nos testes de segurança, o *OWASP ZAP* apresentou maior acessibilidade operacional, sendo adequado para usuários menos experientes e para testes iniciais de segurança em aplicações *web*. O *Burp Suite* demonstrou maior profundidade técnica, mas exige conhecimento especializado. O *Nessus* mostrou-se relevante para identificação de vulnerabilidades em infraestrutura e rede, embora sua adoção em ambientes institucionais possa depender de licença comercial, o que representa uma limitação prática.

Por fim, nos testes de usabilidade, o *UsabilityHub* mostrou-se eficiente para avaliações rápidas de elementos visuais, *layouts* e fluxos simples de navegação. O *UserTesting*, por sua vez, permitiu uma análise mais aprofundada da experiência do usuário, com maior riqueza de dados qualitativos. Assim, os resultados indicam que a eficiência das ferramentas não deve ser avaliada de forma isolada, mas em função do objetivo do teste, do perfil da equipe, dos recursos disponíveis e do nível de automação desejado.

5.8 Limitações e recomendações

Apesar dos resultados obtidos, observaram-se limitações relacionadas à complexidade de configuração, à curva de aprendizado, ao custo de adoção e à necessidade de conhecimento técnico em algumas ferramentas. Soluções como *RestAssured*, *Gatling*, *Burp Suite* e *Appium* apresentaram maior exigência técnica, o que pode dificultar sua adoção por equipes iniciantes ou por ambientes com baixa maturidade em automação de testes. Além disso, ferramentas como *LoadRunner* e *Nessus*, embora robustas, podem envolver barreiras relacionadas a licenciamento, custo e aplicação em contextos corporativos mais estruturados.

Outra limitação observada refere-se à integração entre diferentes tipos de testes. A combinação de testes automatizados, testes de segurança, testes de desempenho e avaliações de usabilidade exige planejamento, padronização dos ambientes e definição clara dos critérios de análise. Em especial, a integração com *pipelines* de CI/CD depende da compatibilidade técnica das ferramentas, da organização dos *scripts* e da capacidade da equipe em manter os testes atualizados ao longo do ciclo de desenvolvimento.

Recomenda-se que a escolha das ferramentas considere a fase do desenvolvimento, os objetivos do projeto, o perfil técnico da equipe, os recursos disponíveis e o nível de automação desejado. Ferramentas como Postman, Cypress e OWASP ZAP mostraram-se mais acessíveis para equipes em fase inicial ou com foco em agilidade e facilidade de uso. Já RestAssured, Gatling, Burp Suite, LoadRunner e Nessus tendem a ser mais adequadas a contextos que exigem maior robustez, automação, escalabilidade ou análise técnica aprofundada.

Por fim, recomenda-se a adoção combinada de ferramentas, de acordo com a finalidade de cada tipo de teste, a fim de ampliar a cobertura da avaliação e fortalecer a qualidade do software. A integração gradual dessas ferramentas a fluxos de CI/CD pode contribuir para a identificação antecipada de falhas, a redução de retrabalho e a melhoria contínua do processo de desenvolvimento. No entanto, essa integração deve ser realizada de forma planejada, considerando as limitações operacionais, o custo de manutenção e a capacidade técnica da equipe responsável pelos testes.

6 CONCLUSÃO

Este trabalho destacou a importância da seleção criteriosa de ferramentas de teste de *software*, considerando as características, limitações e necessidades de cada projeto. A análise comparativa evidenciou que as ferramentas avaliadas apresentam diferentes níveis de adequação, conforme o tipo de teste, o grau de complexidade da configuração, a clareza dos resultados, a facilidade de uso, a capacidade de automação e a compatibilidade com ambientes de desenvolvimento ágil e integração contínua.

Os resultados demonstraram que não há uma ferramenta única capaz de atender, de forma superior, a todas as categorias de teste analisadas. Ferramentas como *PyUnit*, *NUnit*, *Postman*, *Cypress* e *OWASP ZAP* apresentaram maior acessibilidade operacional no cenário experimental adotado, sendo mais adequadas a equipes em fase inicial ou a contextos que demandam simplicidade e agilidade. Por outro lado, soluções como *RestAssured*, *Gatling*, *Burp Suite*, *LoadRunner* e *Nessus,* mostraram maior robustez técnica, mas também exigiram maior conhecimento especializado, configuração mais cuidadosa ou recursos adicionais, como licenciamento e infraestrutura apropriada.

A análise também reforçou a relevância da automação de testes e da integração com práticas de CI/CD, especialmente em ambientes que buscam entregas contínuas, redução de falhas e melhoria da qualidade do *software*. Nesse sentido, a escolha das ferramentas deve considerar não apenas seu desempenho técnico, mas também o perfil da equipe, o estágio do projeto, os recursos disponíveis e a capacidade de manutenção dos testes ao longo do ciclo de desenvolvimento.

Como trabalhos futuros, sugere-se a ampliação do escopo da pesquisa para incluir ferramentas emergentes, inclusive aquelas baseadas em Inteligência Artificial, bem como a aplicação de métricas quantitativas mais detalhadas, como cobertura de testes, taxa de falhas identificadas, tempo médio de resposta, número de requisições processadas e esforço de configuração. Também se recomenda investigar a aplicação dessas ferramentas em ambientes reais de DevOps e *pipelines* de entrega contínua, a fim de validar sua eficiência em contextos mais amplos e próximos da prática profissional.

REFERÊNCIAS

ACHARYA, Shivani; PANDYA, Vidhi. Bridge between Black Box and White Box – Gray Box Testing Technique. **International Journal of Electronics and Computer Science Engineering**, v. 2, n. 1, p. 175-185, 2012. Disponível em: https://www.academia.edu/32153458/Bridge_between_Black_Box_and_White_Box_Gray_Box_Testing_Technique. Acesso em: 27 abr. 2026.

STACK OVERFLOW. **2024 Developer Survey: Technology**. 2024. Disponível em: https://survey.stackoverflow.co/2024/technology. Acesso em: 27 abr. 2026.

BEIZER, B. **Software Testing Techniques**. 2. ed. New York: Van Nostrand Reinhold, 1990. Disponível em: https://dl.acm.org/doi/10.5555/79060. Acesso em: 27 abr. 2026.

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. **Introdução ao Teste de Software**. 2. ed. [S. l.]: GEN LTC, 2016. Disponível em: https://minhabiblioteca.com.br/catalogo/livro/78492/introdu-o-ao-teste-de-software/. Acesso em: 27 abr. 2026.

DIAS, Jailson da Costa. **Teste de software com IA: um mapeamento sistemático da literatura**. 2023. 65 p. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro de Informática, Universidade Federal de Pernambuco, Recife, 2023. Disponível em: https://repositorio.ufpe.br/bitstream/123456789/50401/1/TCC%20Jailson%20da%20Costa%20Dias.pdf. Acesso em: 27 abr. 2026.

GIANNECCHINI, Nilo. **Integração contínua com aplicação de testes de regressão**. 2018. Dissertação (Mestrado em Tecnologia) – Faculdade de Tecnologia, Universidade Estadual de Campinas, Limeira, 2018. Disponível em: https://repositorio.unicamp.br/acervo/detalhe/1021914. Acesso em: 27 abr. 2026.

HALILI, E. H. **Apache JMeter:** A Practical Beginner's Guide to Automated Testing and Performance Measurement for Your Websites. [S. l.]: Packt Pub Ltd, 2008. Disponível em: https://unov.tind.io/record/34368?ln=ru. Acesso em: 27 abr. 2026.

HAMILTON, B. NUnit Pocket Reference. [S. l.]: O'Reilly Media, Inc., 2004. Disponível em: https://www.oreilly.com/library/view/nunit-pocket-reference/9780596007393/. Acesso em: 27 abr. 2026.

HOLMES, Antawan; KELLOGG, Marc. **Automating functional tests using Selenium. In:** AGILE 2006. Washington, DC: IEEE, 2006. p. 270-275. DOI: 10.1109/AGILE.2006.19. Disponível em: https://ieeexplore.ieee.org/document/1667589. Acesso em: 27 abr. 2026.

HONORATO, Viviane Silva. **A importância dos testes manuais e automatizados em sistemas críticos perante um cenário pandêmico e remoto**. 2022. 21 f. Trabalho de Conclusão de Curso (Artigo) – Departamento de Computação, Universidade Estadual da Paraíba, Campina Grande, 2022. Disponível em: https://repositorio.uepb.edu.br/items/7965cb63-197c-4aff-8f71-53d79959211d. Acesso em: 27 abr. 2026.

JORGENSEN, P. C. **Software Testing:** A Craftsman's Approach. 4. ed. Boca Raton: CRC Press, 2013. Disponível em: https://www.oreilly.com/library/view/software-testing-4th/9781466560680/. Acesso em: 27 abr. 2026.

CHALVATZIS, Ilias; KARRAS, Dimitrios A.; PAPADEMETRIOU, Rallis C. Evaluation of Security Vulnerability Scanners for Small and Medium Enterprises Business Networks Resilience towards Risk Assessment. In: **IEEE International Conference on Artificial Intelligence and Computer Applications**. Dalian: IEEE, 2019. p. 52-58. DOI: 10.1109/ICAICA.2019.8873438. Disponível em: https://ieeexplore.ieee.org/document/8873438. Acesso em: 27 abr. 2026.

RAJAPAKSE, Roshan N.; ZAHEDI, Mansooreh; BABAR, Muhammad Ali; SHEN, Haifeng. Challenges and solutions when adopting DevSecOps: a systematic review. **Information and Software Technology**, v. 141, p. 106700, 2022. DOI: 10.1016/j.infsof.2021.106700. Disponível em: https://www.sciencedirect.com/science/article/pii/S0950584921001543?via%3Dihub. Acesso em: 27 abr. 2026.

TENABLE. **Tenable Nessus Documentation**. [S. l.], [s. d.]. Disponível em: https://docs.tenable.com/Nessus.htm. Acesso em: 27 abr. 2026.

SHAHIN, Mojtaba; BABAR, Muhammad Ali; ZHU, Liming. Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. IEEE Access, v. 5, p. 3909-3943, 2017. DOI: 10.1109/ACCESS.2017.2685629. Disponível em: https://ieeexplore.ieee.org/document/7884954. Acesso em: 27 abr. 2026.

MALDONADO, José Carlos; BARBOSA, Ellen Francine; VINCENZI, Auri Marcelo Rizzo; DELAMARO, Márcio Eduardo; SOUZA, Simone do Rocio Senger de; JINO, Mario. **Introdução ao teste de software**. São Carlos: ICMC-USP, 2004. Disponível em: https://web.icmc.usp.br/SCATUSU/RT/Notas_Didaticas/nd_65.pdf. Acesso em: 27 abr. 2026.

MWAURA, W. **End-to-End Web Testing with Cypress:** Explore techniques for automated frontend web testing with Cypress and JavaScript. [S. l.]: Packt Publishing, 2021. Disponível em: https://www.oreilly.com/library/view/end-to-end-web-testing/9781839213854/. Acesso em: 27 abr. 2026.

O'KEEFFE, Juan Franciso Fonseca. **Análise de fatores de impacto no erro de estimativa de esforço e de duração em projetos de software**. 2012. 92 f. Dissertação (Mestrado em Administração e Negócios) – Faculdade de Administração, Contabilidade e Economia, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, 2012. Disponível em: https://tede2.pucrs.br/tede2/bitstream/tede/5640/1/438587.pdf. Acesso em: 27 abr. 2026.

AMANKWAH, Richard; CHEN, Jinfu; KUDJO, Patrick Kwaku; TOWEY, Dave. An empirical comparison of commercial and open-source web vulnerability scanners. **Software: Practice and Experience**, v. 50, n. 9, p. 1842-1857, 2020. DOI: 10.1002/spe.2870. Disponível em: https://onlinelibrary.wiley.com/doi/full/10.1002/spe.2870. Acesso em: 27 abr. 2026.

PIOVESAN, Ana Claudia. **Framework para testes ágeis de software: uma proposta exploratória**. 2018. 187 f. Dissertação (Mestrado em Engenharia de Produção e Sistemas) – Programa de Pós-Graduação em Engenharia de Produção e Sistemas, Universidade Tecnológica Federal do Paraná, Pato Branco, 2018. Disponível em: https://repositorio.utfpr.edu.br/jspui/bitstream/1/3439/1/PB_PPGEPS_M_Piovesan%2C%20Ana%20Claudia_2018.pdf. Acesso em: 27 abr. 2026.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de Software:** Uma abordagem profissional. 8. ed. Porto Alegre: AMGH Editora, 2016. Disponível em: https://books.google.com.br/books/about/Engenharia_de_Software_8%C2%AA_Edi%C3%A7%C3%A3o.html?hl=pt-BR&id=wexzCwAAQBAJ. Acesso em: 27 abr. 2026.

ROCHA, H. Desenvolvimento guiado por testes com JUnit. In: Congresso Brasil Software Week, 2003, São Paulo. **Anais** [...] São Paulo, 2003. Disponível em: https://www.inf.ufpr.br/andrey/ci221/JUnit_Fenasoft.pdf. Acesso em: 27 abr. 2026.

SOMMERVILLE, Ian. **Software Engineering**. 9. ed. Boston: Addison-Wesley, 2010. Disponível em: https://books.google.com.br/books/about/Software_Engineering.html?id=l0egcQAACAAJ. Acesso em: 27 abr. 2026.

SOUZA, Fernando Ribeiro de. **Testando APIs REST com Postman e Rest Assured: um relato de experiência com o Sistema EducAPI**. 2021. Trabalho de Conclusão de Curso (Licenciatura em Ciência da Computação) –

Centro de Ciências Aplicadas e Educação, Universidade Federal da Paraíba, Rio Tinto, 2021. Disponível em: https://repositorio.ufpb.br/jspui/bitstream/123456789/29049/1/FernandoRibeirodeSouza_TCC.pdf. Acesso em: 27 abr. 2026.