Testes Automatizados: Estratégias e Benefícios Para Qualidade de Software

Davison Alves da Silva[1]
Gryco Holanda (orientador)[2]

RESUMO

Este trabalho tem como objetivo apresentar as estratégias e benefícios dos testes automatizados para qualidade de software. Serão abordados conceitos fundamentais sobre testes de software, tipos de testes, metodologias e ferramentas de automação de testes. Os testes automatizados permitem a verificação sistemática e repetitiva do comportamento esperado da aplicação, contribuindo para a redução de custos e tempo de desenvolvimento. Durante o trabalho, serão apresentados os tipos de testes automatizados mais comuns, tais como testes unitários, de integração e ponta a ponta, além da utilização de frameworks para automação de testes. Serão discutidos também os desafios na implementação de testes automatizados, como a seleção adequada das ferramentas de automação, a manutenção dos testes automatizados ao longo do tempo e a flutuação de dados. Espera-se que este trabalho contribua para a disseminação do conhecimento sobre testes automatizados, incentivando sua adoção no contexto de desenvolvimento de software, além de colaborar para a formação de profissionais mais capacitados e conscientes da importância dos testes automatizados para qualidade do software.

Palavras-chave: Testes de Software. Automação de testes. Qualidade de Software.

1 INTRODUÇÃO

Produzir e manter um software com qualidade é um dos principais objetivos de uma equipe de desenvolvimento. Para atingir esse objetivo, diversas técnicas e estratégias são utilizadas, sendo os testes uma das mais importantes. Os testes têm como objetivo verificar se o software atende aos requisitos e especificações definidos, detectar falhas e garantir que a solução entregue ao cliente seja confiável e eficiente.

Nesse contexto, os testes automatizados se destacam por oferecer uma série de benefícios em relação aos testes manuais, como maior rapidez e precisão, redução de erros e repetitividade, além de permitir a execução de testes em diferentes plataformas e ambientes.

Com o avanço das tecnologias para automação de testes, as equipes de desenvolvimento têm à disposição uma ampla variedade de ferramentas e estratégias

[1] Acadêmico(a) do curso de Ciência da Computação da Anhanguera-Uniderp.
[2] Orientador(a). Docente do curso de Ciência da computação da Anhanguera-Uniderp.

para implementar testes automatizados. Isso permite que os testes sejam executados em diversos níveis, garantindo assim uma maior cobertura para o sistema.

Porém, é importante destacar que a implementação requer uma equipe capacitada e tempo adequado para escrita dos testes. É preciso ter em mente que os testes automatizados são uma parte fundamental do processo de desenvolvimento de software, mas não devem ser vistos como uma solução única para garantir a qualidade do produto final.

Este trabalho tem como objetivo apresentar um estudo sobre as estratégias e benefícios dos testes automatizados para qualidade do software, mostrando a importância de uma abordagem sistemática e bem planejada para a implementação dos testes, de forma a garantir uma maior eficiência e confiabilidade do software produzido.

2 METODOLOGIA

O presente trabalho adotou uma metodologia de revisão bibliográfica, que consiste em buscar, analisar e sintetizar as contribuições científicas de autores sobre um tema específico (SANTOS e CANDELORO, 2006). Com o objetivo de compreender o estado da arte em relação à automação de testes de software, a pesquisa utilizará diversas fontes, como livros, artigos científicos e sites especializados, dentre outros. As palavras-chave utilizadas na busca serão “Testes de Software”, “Automação de testes” e “Qualidade de Software”. A seleção dos artigos será realizada a partir de critérios pré-definidos, como relevância, originalidade e rigor metodológico. A partir da análise crítica dos resultados obtidos, foi possível a realização deste trabalho.

3 DESENVOLVIMENTO

3.1 NÍVEIS DE TESTE

Os níveis de teste são grupos de atividades de teste, com objetivos específicos e abordagens e responsabilidades específicas. Esses níveis estão relacionados com outras atividades dentro do ciclo de vida de desenvolvimento de software e são caracterizados por atributos como base de teste, objeto de teste e defeitos e falhas

típicas. Os quatro níveis de teste definidos no ISTQB são: teste de componentes, teste de integração, teste de sistema e teste de aceitação. Para cada nível de teste, é necessário um ambiente de teste adequado. (ISTQB, 2018).

3.1.1 Teste de unidade

De acordo com o ISTQB, o teste de unidade se concentra na validação e verificação de cada componente de forma isolada. É uma atividade de teste que verifica a funcionalidade, o comportamento e a interface do componente, garantindo que ele atenda às especificações e aos requisitos definidos. Ele é uma parte importante do processo de desenvolvimento de software, pois permite detectar e corrigir problemas em um estágio inicial, reduzindo os custos e os riscos do projeto.

Para Back-end podemos citar testes em funções ou métodos que realizam operações matemáticas e testes em funções ou métodos que transformam dados, como uma cadeia de caracteres (String) para JSON (Javascript Object Notation).

Para Front-end podemos citar testes de renderização, onde espera-se que o componente seja renderizado em tela e testes de interação onde valida-se por exemplo, se o botão exibi um modal após ser clicado.

3.1.2 Teste de integração

De acordo com Pressman (2006), o teste de integração é a fase do processo de teste de software que ocorre depois do teste de unidade e antes do teste do sistema. O objetivo do teste de integração é descobrir erros associados às interfaces e interações entre componentes do software. A realização do teste de integração ocorre a partir da junção de unidades ou módulos previamente testados em um grupo maior, formando subsistemas do software. A estratégia de testes de integração pode ser incremental ou não incremental. Na estratégia incremental, os subsistemas são testados individualmente e depois integrados de forma sequencial, enquanto que na estratégia não incremental, os subsistemas são integrados e testados em conjunto. O autor destaca ainda que o sucesso do teste de integração depende da qualidade dos testes de unidade, já que um erro não detectado nessa fase pode afetar todo o processo subsequente de testes.

Como exemplo podemos citar os testes em que validamos uma API, realizando chamadas a ela e verificando alguns pontos como status e corpo de resposta retornados.

3.1.3 Teste de sistemas

De acordo com Myers (2004), os testes de sistema são realizados para verificar se o software como um todo atende aos requisitos especificados. Esses testes são realizados no sistema inteiro, em contraste com os testes de componente e integração, que examinam partes menores do software. O objetivo dos testes de sistema é avaliar a conformidade do sistema com os requisitos não funcionais, como desempenho, segurança, usabilidade e compatibilidade com diferentes plataformas. Para realizar os testes de sistema, geralmente é necessário ter uma configuração de ambiente semelhante à produção, o que permite simular as condições reais de uso. Além disso, os testes de sistema podem ser realizados por equipes de testes para fornecerem uma perspectiva sobre sua qualidade.

3.1.4 Teste de aceitação

Segundo Beizer (1990), testes de aceitação são "testes formais com o objetivo de determinar se um sistema satisfaz seus critérios de aceitação ou não, e para permitir ao usuário determinar se aceita ou não o sistema". Esses testes são realizados pelos usuários finais ou representantes do cliente e têm como objetivo garantir que o sistema atenda às necessidades e expectativas dos usuários e/ou do cliente. Eles podem ser conduzidos em um ambiente de teste controlado ou no ambiente de produção simulado e podem incluir testes de funcionalidade, usabilidade, desempenho e outros aspectos críticos do sistema.

3.2 ESTRATEGIAS

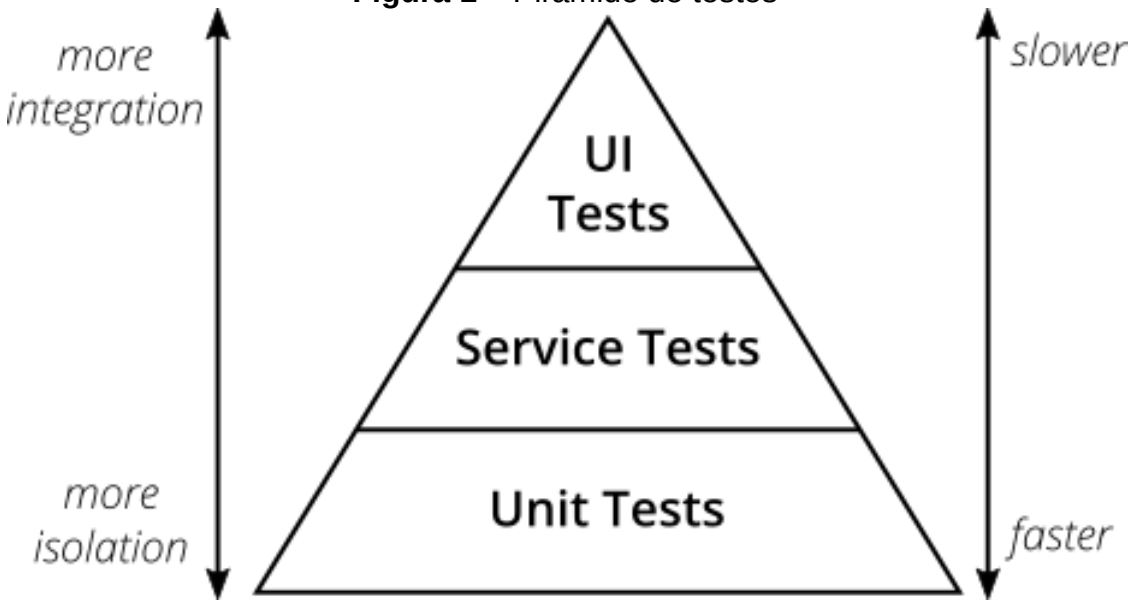
3.2.1 Pirâmide de testes

Uma boa estratégia de testes é fundamental para garantir a qualidade do software desenvolvido. A pirâmide de testes proposta por Mike Cohn sugere que

devemos priorizar a criação de testes unitários na base, seguidos por testes de integração e, por fim, por testes de interface do usuário na ponta. Essa abordagem nos ajuda a maximizar a cobertura de testes, reduzir o tempo e o custo de desenvolvimento e garantir que o software atenda aos requisitos do negócio" (COHN, 2009).

Através da **Figura 1** é possível visualiza-la.

Figura 1 – Pirâmide de testes



Fonte: The Test Pyramid. (Vocke 2018)

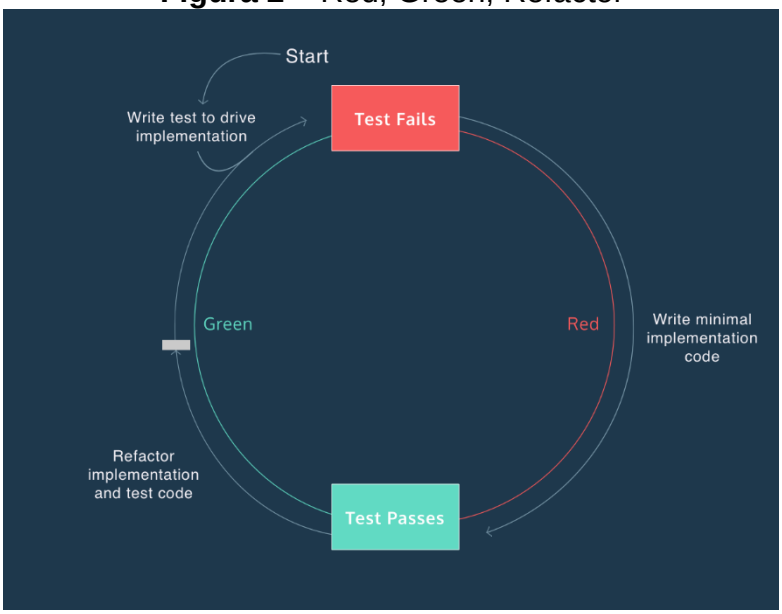
Conforme ilustrado na figura da pirâmide de testes, a base representa a camada onde os testes são mais rápidos e econômicos. À medida que avançamos para camadas superiores, os testes se tornam mais caros e demorados. Isso se deve ao fato de que, na camada de testes unitários, eles são isolados e utilizam uma representação artificial de objetos ou funções (mocks). Por outro lado, em outras camadas, os elementos passam a interagir de maneira mais significativa entre si.

3.2.2 TDD

De acordo com Beck (2003), TDD é uma técnica de desenvolvimento de software que envolve escrever testes automatizados antes de escrever o código-fonte. A ideia é que os testes definam os requisitos e as funcionalidades esperadas do código e, ao executá-los, verifique-se se o código escrito atende a essas expectativas.

Assim, TDD visa garantir que o software desenvolvido tenha uma cobertura de testes ampla e eficaz, o que tende a levar a um código mais confiável, escalável e de qualidade. Além disso, o ciclo TDD, ilustrado na **Figura 2**, incentiva uma abordagem incremental de desenvolvimento, com pequenas iterações de escrita de código e testes.

Figura 2 – Red, Green, Refactor



Fonte: Red, Green, Refactor. (Codeacademy 2023)

Em passos, o ciclo TDD pode ser descrito da seguinte forma:

1. Escrever um teste.
2. Executar o teste e verificar se ele falhou.
3. Escrever um código mínimo para que o teste passe.
4. Executar o teste novamente e verificar se ele passou.
5. Reescrever o código para melhorar sua qualidade.
6. Executar o teste novamente para verificar se ainda está passando
7. Repetir o ciclo para outra funcionalidade.

3.2.3 Testes de regressão

De acordo com "AGILE TESTING" (Crispin; Gregory, 2009, p. 125), testes de regressão são realizados com a finalidade de garantir que as alterações feitas em um software não afetem o funcionamento das funcionalidades existentes.

Tais testes são executados após a realização de mudanças no software, como correções de erros, adições de novas funcionalidades ou alterações na arquitetura, com o objetivo de certificar-se de que o software continue a funcionar corretamente e sem regressões após as mudanças implementadas.

3.3 FERRAMENTAS PARA AUTOMAÇÃO DE TESTES

Existem diversas opções de ferramentas disponíveis para auxiliar na escrita de testes automatizados, cada uma com sua própria especialidade e nível de atuação. Para testes unitários em Java, duas ferramentas bastante utilizadas são o Junit e o TestNG, enquanto em Python, o Unittest e o Pytest são opções populares. Para Javascript, podemos citar o Mocha, o Jasmine e o Jest como algumas das ferramentas mais utilizadas para testes unitários.

Já para testes de integração, o RestAssured é uma das ferramentas mais populares em Java, enquanto em Python, o Requests é bastante utilizado. Em Javascript, o Supertest é uma opção conhecida para realizar chamadas HTTP personalizadas. Vale ressaltar que, em geral, essas ferramentas são utilizadas em conjunto com as ferramentas de testes unitários para realizar asserções e garantir a qualidade do software de forma abrangente.

Além das ferramentas mencionadas para testes unitários e de integração, existem também várias opções disponíveis para testes de interface do usuário (e2e). Esses testes verificam a funcionalidade completa do sistema, simulando as ações do usuário em um ambiente próximo ao real.

Para testes e2e em Javascript, o Cypress é uma das ferramentas mais populares, permitindo a escrita de testes de forma simples e eficiente. Em Python, o Robot Framework é uma opção bem conhecida, que também permite a escrita de testes em outras linguagens. Já para testes e2e em Java, podemos citar o Selenium WebDriver, uma das ferramentas mais utilizadas para automação de testes em navegadores web.

É importante destacar que as ferramentas para testes e2e geralmente exigem mais tempo e recursos computacionais para sua execução, e devem ser utilizadas em conjunto com os testes unitários e de integração para garantir a qualidade e estabilidade do sistema, seguindo a estratégia proposta pela pirâmide de testes.

3.4 DESAFIOS ASSOCIADOS AOS TESTES AUTOMATIZADOS

Embora os testes automatizados sejam uma técnica valiosa e amplamente utilizada para o teste de software, é importante destacar que a sua implementação também envolve desafios que precisam ser superados para obter bons resultados. É essencial que a equipe de desenvolvimento esteja ciente desses desafios e trabalhe de forma estratégica para minimizá-los e garantir a qualidade e eficácia dos testes automatizados.

3.4.1 Seleção adequada das ferramentas de automação

Existem muitas opções disponíveis no mercado, cada uma com suas próprias vantagens e desvantagens. Além disso, as necessidades de automação de testes podem variar de acordo com o tipo de projeto e tecnologia utilizada.

Por exemplo, uma ferramenta que é ótima para a automação de testes de UI em um projeto web pode não ser tão eficaz para a automação de testes de API em um projeto mobile. Além disso, é preciso levar em consideração fatores como a facilidade de uso, a integração com outras ferramentas, o custo e o suporte da comunidade.

Por isso, é importante realizar uma análise cuidadosa das opções disponíveis e selecionar uma ferramenta que atenda às necessidades específicas do projeto, bem como às habilidades da equipe de desenvolvimento e testes. A escolha da ferramenta certa pode economizar tempo e recursos, além de contribuir significativamente para a qualidade do software final.

3.4.2 Manutenção dos testes automatizados ao longo do tempo

Um dos maiores desafios da automação de testes é a manutenção dos testes ao longo do tempo. À medida que o software evolui, novas funcionalidades são adicionadas, bugs são corrigidos e a arquitetura pode ser alterada, o que pode afetar os testes automatizados. É importante que os testes sejam atualizados para refletir essas mudanças e garantir que o software continue funcionando corretamente.

A manutenção dos testes pode ser particularmente desafiadora quando há uma grande quantidade de testes automatizados, tornando difícil acompanhar e atualizar

todos eles. Além disso, a falta de documentação adequada dos testes pode dificultar a manutenção, especialmente se o desenvolvedor que criou o teste não estiver mais disponível para realizar as atualizações necessárias.

Outro desafio é garantir que os testes continuem sendo relevantes e eficazes ao longo do tempo. À medida que o software evolui, os testes devem ser atualizados para garantir que cubram adequadamente as funcionalidades mais recentes e as áreas mais críticas do software.

Para enfrentar esses desafios, é importante ter uma estratégia clara para a manutenção dos testes automatizados. Isso inclui a documentação adequada dos testes, a identificação de testes obsoletos ou duplicados e a realização regular de revisões e atualizações dos testes existentes. Também é importante ter uma equipe dedicada para gerenciar e manter os testes automatizados, com habilidades e conhecimentos adequados em automação de testes e no software em questão.

3.4.3 Flutuação de dados

Um desafio comum na automação de testes é lidar com a flutuação de dados, ou seja, a variação dos dados de entrada que podem afetar o resultado dos testes. Quando os dados mudam, é necessário adaptar os testes para que continuem a produzir resultados precisos e confiáveis.

Uma das maneiras de lidar com a flutuação de dados é usar dados de teste estáveis e consistentes, que não mudam frequentemente e que são representativos dos dados do mundo real que o software pode encontrar. Isso pode exigir a criação de conjuntos de dados de teste e a manutenção de uma base de dados para armazená-los.

Outra abordagem é a geração automatizada de dados de teste, onde são criados conjuntos de dados sintéticos ou aleatórios que são representativos dos dados reais que o software pode receber. No entanto, a geração de dados pode ser complexa e pode levar a dados que não são realistas, o que pode afetar a precisão dos testes.

Em geral, a flutuação de dados pode ser mitigada com uma abordagem cuidadosa e atenta para a criação e manutenção de dados de teste estáveis e representativos, além de uma validação constante dos dados de entrada e saída para garantir a precisão e confiabilidade dos testes automatizados.

3.5 BENEFICIOS DOS TESTES AUTOMATIZADOS

A implementação da automação de testes pode trazer diversos benefícios para um projeto de desenvolvimento de software e de acordo com "AGILE TESTING" (Crispin; Gregory, 2009), alguns desses benefícios são:

1. Redução do tempo de teste: A automação de testes permite que sejam realizados testes repetitivos e tediosos de forma rápida e eficiente, liberando os testadores para focarem em testes mais críticos e exploratórios.
2. Aumento da cobertura de testes: Com a automação de testes, é possível testar uma ampla gama de casos de teste e cenários que seriam impraticáveis de testar manualmente, aumentando a cobertura de testes.
3. Melhoria da qualidade do software: A automação de testes permite a identificação precoce de problemas, tornando mais fácil para os desenvolvedores corrigi-los, melhorando a qualidade do software em geral.
4. Redução de erros humanos: A automação de testes reduz a probabilidade de erros humanos em testes repetitivos e rotineiros.
5. Melhoria da comunicação: A automação de testes ajuda a garantir que todos os envolvidos no projeto tenham um entendimento claro dos requisitos do software e das expectativas de teste, melhorando a comunicação entre as equipes.
6. Redução de custos: A automação de testes pode reduzir os custos de teste a longo prazo, pois os testes podem ser reutilizados em várias iterações do software e em diferentes projetos.

4 RESULTADOS E DISCUSSÃO

A crescente complexidade dos sistemas de software e a necessidade de entregas rápidas e de qualidade têm levado cada vez mais equipes de desenvolvimento a adotar a prática de testes automatizados. Nesse sentido, é importante destacar que a utilização de testes automatizados não só contribui para a garantia da qualidade do software produzido, como também é uma prática essencial para a redução de custos e retrabalhos, bem como para a manutenção da competitividade no mercado.

De fato, a literatura especializada tem destacado a importância dos testes automatizados para garantir a qualidade do software. Dentre os benefícios apontados pelos autores, destaca-se a maior precisão na detecção de falhas, a possibilidade de executar testes em diferentes plataformas e ambientes, e a maior cobertura do sistema.

Beck (2003) afirma que um dos principais benefícios dos testes automatizados é a maior precisão na detecção de falhas. Isso ocorre porque os testes são realizados de forma sistemática e padronizada, o que aumenta a confiabilidade dos resultados obtidos. Além disso, os testes automatizados permitem a execução de testes em diferentes plataformas e ambientes, o que pode contribuir para a identificação de problemas que não seriam encontrados em outras situações, como destaca Beizer (1990).

Cohn (2009) ressalta que a utilização de testes automatizados permite uma maior cobertura do sistema, o que aumenta a confiabilidade do produto final. Isso porque os testes podem ser realizados em diferentes cenários e situações, garantindo que o software seja testado de forma mais completa e abrangente. Isso é particularmente importante em projetos de software complexos, nos quais a cobertura do sistema pode ser um desafio.

No entanto, a implementação de testes automatizados pode apresentar alguns desafios. Cohn (2009) afirma que é fundamental que a equipe de desenvolvimento esteja capacitada e planeje adequadamente a implementação dos testes automatizados. Isso envolve a definição de estratégias para seleção da ferramenta adequada, a criação de um plano de testes bem estruturado e a realização de manutenção constante nos testes automatizados, de forma a garantir que eles continuem eficientes e confiáveis.

Crispin e Gregory (2009) destacam que a implementação de testes automatizados pode apresentar outros desafios, como a necessidade de integrar os testes automatizados com outras práticas de garantia da qualidade, como revisões de código e testes manuais. Isso exige uma abordagem integrada e bem planejada, que leve em conta as necessidades específicas do projeto em questão.

Pressman (2016) destaca que a redução de erros é um dos principais objetivos da garantia da qualidade de software, e que os testes automatizados podem contribuir significativamente para alcançá-lo. Além disso, a ISTQB (2018) enfatiza a importância

da realização de testes em diferentes níveis, desde testes unitários até testes de integração e aceitação.

A utilização de testes automatizados está intimamente relacionada com a metodologia ágil de desenvolvimento de software, como destaca Vocke (2018). A metodologia ágil valoriza a automação de testes, como parte do processo de desenvolvimento, permitindo que sejam realizados com mais frequência e contribuindo para a entrega de software de maior qualidade. Isso ocorre porque os testes automatizados permitem que a equipe de desenvolvimento tenha um feedback rápido e constante sobre o estado do sistema, o que pode contribuir para a detecção precoce de problemas e o aumento da eficiência na resolução desses problemas. A detecção precoce de problemas no software é essencial para evitar prejuízos financeiros e de imagem para a empresa, além de garantir a satisfação dos usuários.

Além disso, os testes automatizados permitem uma maior escalabilidade do processo de teste, uma vez que é possível executar um grande número de testes em um curto espaço de tempo, sem comprometer a qualidade do resultado. Isso é particularmente importante em projetos com prazos apertados, nos quais a equipe de desenvolvimento precisa garantir que o software esteja pronto para ser lançado dentro do prazo estipulado.

Vale destacar também que os testes automatizados podem ser integrados aos processos de integração contínua e entrega contínua (CI/CD), tornando o processo de desenvolvimento ainda mais eficiente. Com a integração contínua, os desenvolvedores podem automatizar a construção e teste do software em um ambiente isolado, garantindo que as mudanças feitas em um módulo do software não afetem outros módulos. Já a entrega contínua permite que o software seja entregue aos usuários finais de forma rápida e segura, por meio de um processo automatizado que envolve a realização de testes automatizados em diferentes fases do processo de desenvolvimento.

Apesar dos benefícios dos testes automatizados, é preciso ter em mente que eles não substituem completamente os testes manuais, como aponta Pressman (2016). Os testes manuais ainda são necessários em algumas situações, como na validação da experiência do usuário e na detecção de problemas não identificados pelos testes automatizados.

Por fim, é importante ressaltar que a utilização de testes automatizados exige uma abordagem sistemática e planejada para sua implementação, como apontado

por Pressman (2016). É fundamental que a equipe de desenvolvimento esteja capacitada e planeje adequadamente a implementação dos testes automatizados, definindo as estratégias de teste adequadas, criando um plano de testes bem estruturado e realizando manutenção constante nos testes automatizados, para garantir sua eficiência e confiabilidade.

Em suma, os testes automatizados são indispensáveis na garantia da qualidade do software, uma vez que eles possibilitam a detecção precoce de problemas e permitem que a equipe de desenvolvimento os corrija antes da entrega do produto final. Além disso, esses testes também contribuem para a redução de erros e para o aumento da eficiência no processo de desenvolvimento, já que é possível realizar testes em diferentes plataformas e ambientes.

No entanto, é importante ressaltar que os testes automatizados não devem ser vistos como uma solução única, pois eles não conseguem detectar todos os tipos de problemas, como aqueles que envolvem a usabilidade do software. Por isso, é necessário complementar os testes automatizados com outros tipos de teste, como os testes manuais, que são realizados por um ser humano e conseguem avaliar aspectos subjetivos do software.

Para que os testes automatizados sejam eficazes, é fundamental que a equipe de desenvolvimento adote uma abordagem sistemática e planejada, desde a seleção da ferramenta mais adequada até a definição de um plano de testes bem estruturado. Além disso, é importante realizar a manutenção constante dos testes automatizados, para que eles continuem relevantes e eficientes ao longo do tempo. Dessa forma, a implementação dos testes automatizados se torna uma estratégia eficaz para garantir a qualidade do software e aumentar a satisfação do usuário final.

5 CONCLUSÃO

Em conclusão, a implementação de testes automatizados é uma técnica valiosa para garantir a qualidade do software. Eles oferecem uma série de benefícios, como maior rapidez e precisão, redução de erros e repetitividade, além de permitir a execução de testes em diferentes plataformas e ambientes. No entanto, a implementação dos testes automatizados requer uma equipe capacitada e tempo adequado para escrita dos testes, além de enfrentar desafios como a seleção da ferramenta adequada, manutenção dos testes ao longo do tempo, criação de testes

confiáveis e consistentes e lidar com a flutuação de dados. Portanto, uma abordagem sistemática e bem planejada para a implementação dos testes é fundamental para garantir sua eficiência e confiabilidade.

Por fim, espera-se que este estudo tenha contribuído para o entendimento dos benefícios e desafios dos testes automatizados e para a importância de uma abordagem cuidadosa e bem planejada na implementação dessas técnicas.

REFERÊNCIAS

BECK, K. (2003). **Test-Driven Development: By Example**. Addison-Wesley Professional.

BEIZER, B. (1990). **Software Testing Techniques (2nd ed.)**. Van Nostrand Reinhold.

COHN, Mike. **Succeeding with Agile: Software Development using Scrum**. Addison-Wesley Professional, 2009.

CRISPIN, Lisa; GREGORY, Janet. **Agile testing: a practical guide for testers and agile teams**. Addison-Wesley Professional, 2009.

ISTQB. (2018). **Guia de Referência para Teste de Software**. Disponível em: https://www.istqb.org/downloads/send/51-ctfl2018/288-istqb-ctfl-2018-syllabus-pdf.html. Acesso em: 18 set. 2022.

MYERS, Glenford J. **The art of software testing**. John Wiley & Sons, 2004.

PRESSMAN, R. S. **Engenharia de software: uma abordagem profissional**. 7. ed. Porto Alegre: AMGH, 2016.

SANTOS, R. L.; CANDELORO, J. M. **Metodologia científica: a construção do conhecimento**. Rio de Janeiro: DP&A, 2006.

CODEACADEMY. **Red, Green, Refactor**. Disponível em: https://www.codecademy.com/articles/tdd-red-green-refactor. Acesso em: 22 mar. 2023.

VOCKE, Ham. **The Practical Test Pyramid**. 2018. Disponível em: https://martinfowler.com/articles/practical-test-pyramid.html. Acesso em: 20 mar. 2023.