DESENVOLVIMENTO DE SISTEMAS

JOSÉ VITOR
LORENA APARECIDA
THAYNÁ MARLEN
THIAGO WILLER
VICTOR GOUVEIA
WANUSSIA SANTOS
WENDEL JUNIOR
WILLIAM DUQUE

TESTES DE SOFTWARE

CONTAGEM
2024

INTRODUÇÃO

O presente trabalho tem como principal objetivo mostrar os diferentes tipos de testes de Software mais utilizados, sendo eles: testes de integração, unitários, aceitação e de sistema. Bem como abordagens e metodologias de testes TDD(Test-Driven Development) e BDD(Behavior-Driven Development) destacando seus principais benefícios e também suas desvantagens.

O QUE SÃO TESTES DE PROGRAMAÇÃO?

Os testes de programação são diferentes métodos utilizados para analisar a eficácia de um programa desenvolvido para diferentes finalidades. Existem diferentes formas de validar um programa, temos o com métodos adequados para cada tipo de caso específico.

Os testes de unidade analisam isoladamente cada unidade do código, por sua vez, o teste de integração verifica a interação entre módulos, garantindo que os componentes funcionam juntamente de forma correta. todos com um objetivo em comum, identificar e corrigir falhas de um protótipo ou de um projeto já implementado, a fim atender às expectativas do cliente.

TIPOS DE TESTES

1. TESTES UNITÁRIOS

Os testes unitários são responsáveis por verificar partes específicas do código, para garantir que funcionem isoladamente de acordo com sua função. Devem ser feitos diversos testes dentro da função para que diversos caminhos que foram tomados, ainda continuem funcionando como esperado.

Podem ser feitos de duas maneiras:

- Testes unitários automatizados: são feitos através de programas e códigos, são mais ágeis, por outro lado, podem não identificar todos os erros.
- Testes unitários manuais: necessitam de desenvolvedores qualificados que entendam a complexidade de identificação de diversos cenários possíveis para minimizar os erros. Isso eleva o custo pela mão de obra qualificada.
-

Um bom teste unitário deve ser rápido, funcionar isoladamente de forma concisa, retornar sempre os mesmos resultados ao repetir o teste para ser considerável seguro e confiável,

Prós:

- Mais fácil identificar erros dentro do código
- Fácil estruturação e manutenção
- Satisfação do cliente com redução de erros.
- Otimização de tempo, os testes são programados uma só vez, evitando que os programadores desenvolvam testes a cada uso.

Contra:

- Por mais que o desenvolvedor preveja diversos cenários para os testes, pode ocorrer situações inesperadas gerando erros.
- Como são feitos em blocos, podem ficar muito extensos.

2. TESTES INTEGRAÇÃO

Teste de integração é uma etapa do processo de desenvolvimento de software em que módulos ou componentes são combinados e testados em grupo. Esse tipo de teste visa verificar a eficiência e a segurança da comunicação entre sistemas. Essa etapa é essencial para garantir que o software funcione sem erros de integração.

O teste de integração é geralmente realizado em fases, começando com os módulos mais simples e progredindo para os módulos mais complexos. Além disso, ele pode ser realizado de forma manual ou automatizada e deve ser feito de forma colaborativa entre desenvolvedores, testadores e engenheiros de software.

Quais falhas o teste de integração consegue identificar?
O teste de integração é uma etapa essencial para garantir a qualidade do software. Ao identificar e corrigir falhas de interface, o teste de integração ajuda a garantir que o sistema funcione corretamente e de forma confiável.

O teste de integração automatizado oferece várias vantagens, conheça as principais:

- Facilidade de execução: Os testes de integração automatizados podem ser executados com um simples comando, o que facilita a execução dos testes.

- Redução de custos: O teste de integração automatizado pode ajudar a reduzir os custos de teste, pois reduz a necessidade de testes manuais e os erros humanos.

- Melhoria da qualidade: O teste de integração automatizado pode ajudar a melhorar a qualidade do software, pois ajuda a identificar falhas de integração.

3. TESTES SISTEMA

Definição:
Fase do ciclo de vida do desenvolvimento de software onde o sistema completo e integrado é testado como um todo.

Objetivo: Verificar se o sistema atende aos requisitos especificados e funciona conforme esperado.

Funcionalidade: verificar se todas as funcionalidades operam conforme especificado.

Desempenho: avaliar a eficiência sob diferentes condições de carga.

Segurança : garantir que o sistema é seguro contra ameaças.

Usabilidade: Testar a facilidade de uso para usuários finais.

Compatibilidade: assegurar funcionamento em diferentes ambientes e dispositivos.

Confiabilidade: verificar a estabilidade ao longo do tempo.
Interoperabilidade: garantir interação com outros sistemas e componentes externos.

Ferramentas Comuns:

- Selenium: Testes de interface web.
- JUnit/TestNG: Testes de unidade e integração em Java.
- JMeter: Testes de desempenho e carga.
- Postman: Testes de APIs.
- Appium: Testes de aplicativos móveis.
- Cucumber: Testes de aceitação automatizados.
- Jenkins: Integração contínua e automação de testes.

3. TESTES DE ACEITAÇÃO

O texto discute a importância dos testes de aceitação para minimizar o custo do desenvolvimento de software. Sugere que mapear todas as expectativas do cliente em testes desde o início permite desenvolver o software de acordo com as funcionalidades desejadas. Sommerville (2007) define o teste de aceitação como crucial para verificar se o sistema atende aos requisitos do cliente com dados reais. O modelo V enfatiza a integração do teste desde as fases iniciais do ciclo de vida do software.

No contexto do Acceptance Test-Driven Development (ATDD), os testes de aceitação são usados desde o início do projeto para guiar o desenvolvimento, melhorar a análise e garantir

entrega alinhada com as necessidades do cliente. Melnik (2009) destaca que mudanças nos requisitos têm um grande impacto nos testes de aceitação. Ferreira (2005) menciona que esses testes ajudam a refinar a análise inicial ao prever diversas interações do usuário.

Os desenvolvedores escrevem testes para cada funcionalidade antes da codificação para melhorar o entendimento das necessidades do cliente, manter a simplicidade do sistema e validar o software de forma abrangente (Teles, 2004). Koskela (2008) define os testes de aceitação como especificações do comportamento desejado do sistema, focados no valor entregue ao cliente e expressos na linguagem do domínio do problema.

Em resumo, os testes de aceitação são essenciais para validar se as histórias de usuário foram implementadas corretamente, facilitando a colaboração entre cliente e desenvolvedores ao longo do desenvolvimento de software.

4. TESTES DE SEGURANÇA

Importância do teste de segurança:
- Proteção de dados sensíveis
- Prevenção de acessos não autorizados
- Conformidade com regulamentos (LGPD, GDPR
- Evitar danos à reputação

Tipos de testes de segurança:
- Testes de Penetração: Simula ataques reais.
- Teste de Vulnerabilidade: Identifica falhas conhecidas.
- Fuzzing: Envia dados aleatórios.
- Revisão de Código: Inspeciona o código.

Ferramentas:
- Nmap: Scanner de rede.
- Metasploit: Testes de penetração.
- OWASP ZAP: Proxy de segurança.
- Nessus: Scanner de vulnerabilidades.

Boas práticas de segurança: Atualização regular de software, controles de acesso, criptografia de dados,treinamento de usuários, monitoramento contínuo.

Processo:
- Planejamento e Preparação

- Reconhecimento
- Análise de Vulnerabilidades
- Exploração
- Relatório

Abordagens e Metodologias de Teste

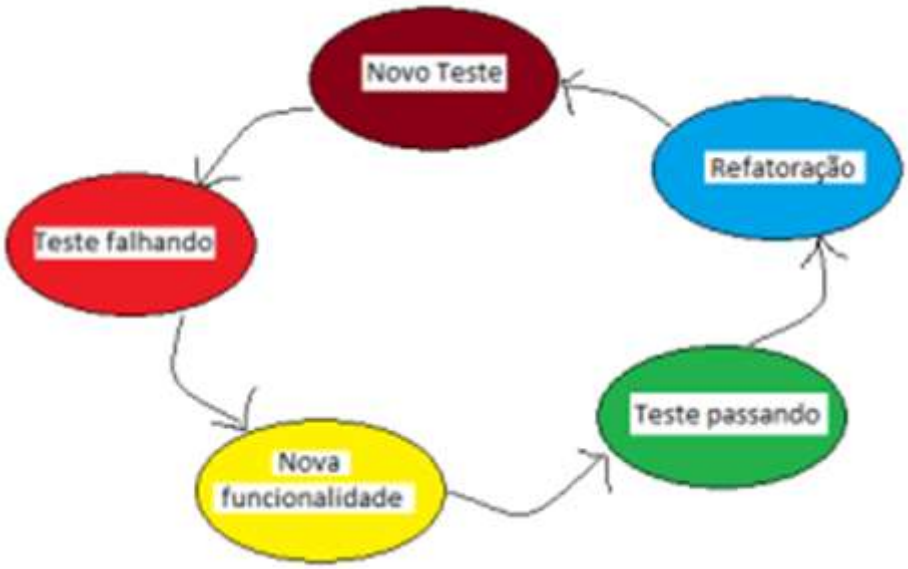
Test-Driven Development(TDD):

Baseia-se em pequenos ciclos de repetições, onde para cada funcionalidade do sistema um teste é criado antes. Este novo teste criado,inicialmente falha, já que ainda não temos a implementação da funcionalidade em questão e, em seguida, implementamos a funcionalidade para fazer o teste passar

Ciclo de desenvolvimento:

-Os ciclos são Red,Green, Refactor. Logo eles :Escrevemos um Teste que inicialmente não passa (Red)Adicionamos uma nova funcionalidade do sistemaFazemos o Teste passar(Green).Refatoramos o código da nova funcionalidade (Refactoring)

Escrevemos o próximo teste:



Vantagens para a utilização do TDD:

1-Feedback rápido sobre a nova funcionalidade e sobre as outras funcionalidades existentes no sistema

2-Código mais limpo, já que escrevemos códigos simples para o teste passar

3-Segurança no Refactoring pois podemos ver o que estamos ou não afetando

4-Segurança na correção de bugs

5-Maior produtividade já que o desenvolvedor encontra menos bugs e não desperdiça tempo com depuradores

6-Código da aplicação mais flexível, já que para escrever testes temos que separar em pequenos "pedaços" o nosso código, para que sejam testáveis, ou seja, nosso código estará menos acoplado.

Descrição da existência dos três ciclos:

Red: escreva um pequeno teste automatizado que, ao ser executado, irá falhar;

Green: implemente um código que seja suficiente para ser aprovado no teste recém-escrito;

Refactor: refatore o código, a fim dele ser melhorado, deixando-o mais funcional e mais limpo.

Desafios do TDD:

O TDD pode levar mais tempo do que a abordagem tradicional de desenvolvimento, pois os testes devem ser escritos antes do código. Isso pode aumentar o tempo de desenvolvimento, especialmente no início do projeto.

Behavior-Driven Development(BDD)

É uma técnica de desenvolvimento ágil que tem por objetivo, integrar regras de negócios com linguagem de programação, focando o comportamento do software. Além disso, pode-se dizer também, que BDD é a evolução do TDD. Isto porque, os testes ainda orientam o desenvolvimento, ou seja, primeiro se escreve o teste e depois o código.

Comparações entre TDD e BDD:

O Test Driven Development (TDD) e o Behavior Driven Development (BDD) são duas metodologias de desenvolvimento de software que têm como objetivo garantir a qualidade do código produzido. A principal diferença entre elas é que o TDD é focado em uma linguagem mais técnica, com o objetivo de testar alguma funcionalidade, enquanto o BDD é focado em uma linguagem mais próxima da linguagem natural, ou seja, testes de comportamento.

Ferramentas populares do BDD:

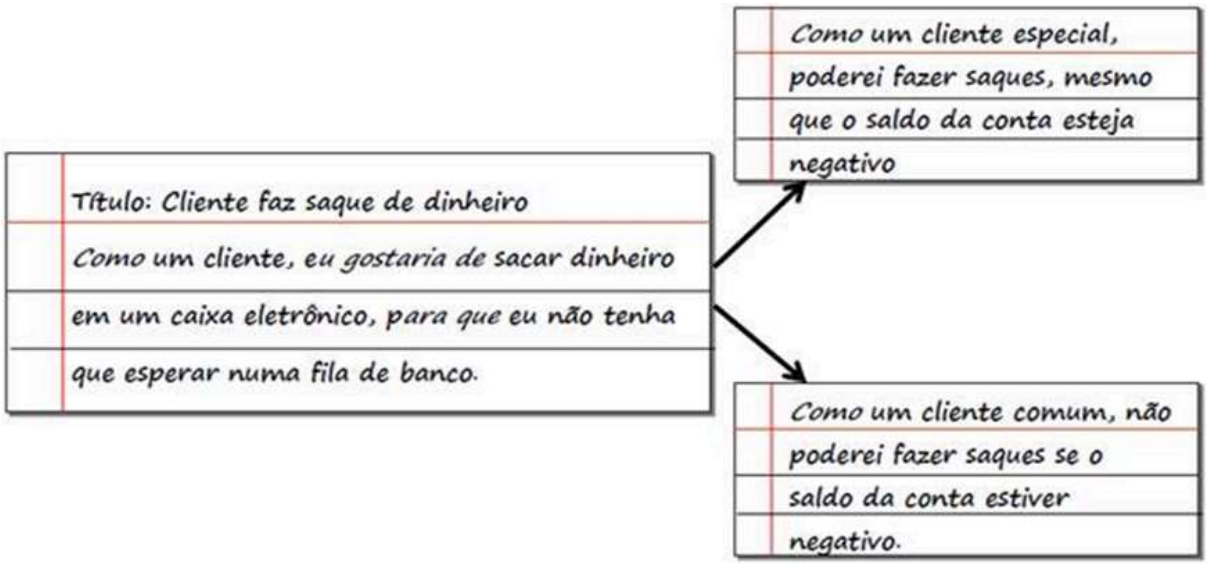
Cucumber uma plataforma popular de código aberto para colaboração BDD. Ele ajuda as equipes a criar funcionalidades em texto sem formatação. Ele fornece coesão usando DSL que é fácil de compartilhar com toda a equipe. Ele implementa a sintaxe Gherkin.

O Lettuce é construído em cima do Cucumber e funciona muito bem para aplicativos baseados em Python.

O Specflow é outra plataforma BDD de código aberto que usa a sintaxe Gherkin. Ele foi criado para a plataforma .NET.

Exemplos práticos de BDD:

-Exemplo 1:



-Exemplo 2:

US002 - Cadastro de Solicitação de Proposta de Crédito
COMO um novo cliente EU quero preencher um formulário PARA solicitar uma proposta de crédito
Cenário 01: Idade inferior a 18 anos DADO que estou preenchendo o campo de data de nascimento do formulário QUANDO digito a data de nascimento inferior a 18 anos ENTÃO o sistema exibe na tela a informação “Não é permitido propostas a menores de 18 anos”. **Cenário 02: CPF Inválido** DADO que estou preenchendo o campo CPF do formulário de proposta QUANDO digito o CPF do cliente errado ENTÃO o sistema exibe na tela a informação “CPF inválido”.

Práticas e Ferramentas

Automação de testes

A Automação de Testes é um dos processos que compõem a metodologia Quality Assurance (Quality Assurance, ou simplesmente QA, significa Garantia da Qualidade. Trata-se de um conjunto de atividades que têm como principal objetivo assegurar que um produto ou serviço a ser oferecido aos clientes da empresa tenha o padrão de qualidade pretendido, considerando as definições iniciais da empresa e as expectativas dos clientes) e que usa um software para testar um aplicativo em desenvolvimento, aplicando ferramentas e estratégias que comparam os resultados com o objetivo do produto, garantindo a qualidade do que será entregue.

Os testes de regressão, técnica que consiste na aplicação de versões mais recente do software, para garantir que não surgiram novos defeitos em componentes já analisados, são repetitivos, pois verificam se cada alteração de um programa não gerou problemas.A automação torna esses testes mais rápidos e menos cansativos do que os testes manuais.

Benefícios

A Automação de Testes traz uma série de vantagens para os projetos de desenvolvimento de software, tais como:

- Aumento da qualidade: possibilita detectar e corrigir erros mais rapidamente, evitando que eles se propaguem para as fases posteriores do ciclo de vida do software ou para os usuários finais. Além disso, a Automação de Testes garante uma maior cobertura e consistência dos testes, reduzindo as chances de falhas humanas ou de esquecimento de cenários de teste importantes;
- Redução de custos: diminui o tempo e os recursos necessários para realizar os testes, gerando uma economia significativa para o projeto;
- Otimização do tempo: acelera o processo de teste, permitindo que os testes sejam executados de forma paralela, contínua e integrada ao desenvolvimento. Isso contribui para a entrega mais rápida e frequente de software, seguindo os princípios de metodologias ágeis e de DevOps;
- Melhoria da satisfação: melhora a experiência e a satisfação dos clientes, dos usuários e dos próprios testadores. Os clientes e os usuários recebem um software de maior qualidade, com menos bugs e mais funcionalidades. Os testadores podem se dedicar a atividades mais estratégicas e criativas, como planejar, projetar e analisar os testes, em vez de executá-los manualmente.

Desafios:

Apesar dos benefícios, a Automação de Testes também apresenta alguns desafios que devem ser considerados e superados, tais como:

Escolha das Ferramentas: existem diversas ferramentas disponíveis no mercado para automatizar os testes de software, cada uma com suas características, vantagens e desvantagens.É preciso avaliar criteriosamente as opções e escolher as ferramentas que melhor se adequa ao contexto, aos objetivos e às necessidades do projeto.

Definição do Escopo: Nem todos os testes podem ou devem ser automatizados. É preciso definir quais testes são candidatos à automação, considerando fatores como frequência, complexidade, estabilidade e criticidade dos testes.Em geral, recomenda-se automatizar os testes executados com muita frequência, que são simples e padronizados, possuem requisitos estáveis e são críticos para o funcionamento do software.

Manutenção dos testes: Os testes automatizados também precisam ser mantidos e atualizados conforme o software evolui. É necessário garantir que os testes estejam sempre alinhados com os requisitos e as funcionalidades do software, e que sejam capazes de identificar os erros esperados.Além disso, deve-se monitorar e analisar os resultados dos testes, para verificar se eles estão sendo executados corretamente e se estão gerando valor para o projeto.

Capacitação da Equipe: A Automação de Testes requer um conjunto de habilidades e conhecimentos específicos, tanto técnicos quanto gerenciais.A equipe de teste deve ser capacitada para que ela possa planejar, projetar, implementar, executar e gerenciar os testes automatizados de forma eficiente e eficaz. Também é preciso promover uma cultura de colaboração e comunicação entre os envolvidos no projeto, para que a automação de testes seja integrada ao processo de desenvolvimento de software.

Como começar a automatizar os testes

Para iniciar um projeto de Automação de Testes, deve-se seguir alguns passos básicos, que podem variar conforme a metodologia e a ferramenta utilizadas. De forma geral, os passos são:

- **Definir a estratégia de automação**

A estratégia de automação é o documento que define os objetivos, o escopo, as ferramentas, as técnicas, os critérios, os indicadores e as responsabilidades da automação de testes.Ela deve ser elaborada com base nas características e nos requisitos do projeto, e deve ser revisada e atualizada periodicamente.

- **Selecionar os casos de teste**

Os casos de teste são as especificações dos testes que serão automatizados, contendo os dados de entrada, as ações, os resultados esperados e os critérios de aceitação. Eles devem ser selecionados segundo a estratégia de automação, priorizando os testes que trazem mais benefícios para o projeto.

- **Desenvolver os scripts de teste**

Os scripts de teste são os códigos que implementam os casos de teste, utilizando a ferramenta de automação escolhida. Devem ser desenvolvidos seguindo as boas práticas de programação, como modularização, reutilização, documentação, etc.

- **Testar os scripts de teste**

Os scripts de teste devem ser testados antes de serem executados no ambiente de teste, verificando se eles estão funcionando corretamente, se eles cobrem todos os cenários previstos e se eles geram os resultados esperados.

Os testes dos scripts de teste podem ser feitos de forma manual ou automatizada, dependendo da ferramenta e da complexidade dos scripts.

- **Executar os testes automatizados**

Os testes automatizados devem ser executados no ambiente de teste, seguindo um cronograma definido na estratégia de automação.Os mesmos podem ser executados de forma isolada ou integrada, dependendo do tipo e do nível dos testes e devem ser monitorados e controlados, verificando se estão sendo executados conforme o planejado e se não estão causando impactos negativos no ambiente de teste.

- **Analisar os resultados dos testes automatizados**

Os resultados dos testes automatizados devem ser analisados após a execução dos testes, verificando se eles atendem aos critérios de aceitação definidos na estratégia de automação.Estes resultados devem ser registrados e reportados, gerando relatórios, gráficos, indicadores e evidências que possam auxiliar na tomada de decisão e na melhoria contínua da automação de testes.

Esses são os passos básicos para começar a automatizar os testes de software. Eles podem variar conforme o contexto, o objetivo e a complexidade de cada projeto.Por isso, é importante adaptar a automação de testes às necessidades e às características de cada projeto, buscando sempre a otimização e a qualidade dos processos de desenvolvimento de software.

Desafios e Melhores Práticas

Alguns desafios comuns encontrados nos Testes de Software:

Manutenção de Testes: os testes podem se tornar obsoletos rapidamente devido a alterações no código, exigindo atualizações constantes.

Testes Flutuantes: os testes podem ser afetados por fatores externos, como alterações de ambiente ou dependências, tornando-os instáveis.

Cobertura de Teste: garantir que todos os aspectos do sistema sejam testados pode ser desafiador, especialmente em sistemas complexos.

Automação de Testes: automatizar os testes pode ser caro e demorado, e os testes automatizados ainda podem exigir manutenção manual.

Falta de Especialistas: encontrar e reter testadores de software qualificados pode ser difícil.

Estratégias para Superar esses Desafios

Manutenção de Testes: Use técnicas de design de teste modular para facilitar a atualização de testes. Automatize os testes sempre que possível para reduzir o esforço de manutenção.

Testes Flutuantes: Teste em vários ambientes e configurações para identificar e mitigar dependências externas. Use testes de ponta a ponta para validar o comportamento do sistema em condições reais.

Cobertura de Teste: Use técnicas de cobertura de código para identificar áreas não testadas. Realize testes exploratórios para complementar os testes automatizados.

Automação de Testes: Priorize os testes que fornecem maior valor e retorno sobre o investimento. Use ferramentas de automação de teste para reduzir o tempo e o esforço de teste.

Falta de Especialistas: Forneça treinamento e desenvolvimento para os testadores existentes. Colabore com equipes de desenvolvimento para promover práticas de teste eficazes. Considere terceirizar os serviços de teste quando necessário.

Conclusão

Com base nas pesquisas feitas sobre testes de sistemas, conseguimos identificar a importância dessa etapa no processo de desenvolvimento de softwares, para entrega de um produto final com qualidade, segurança e redução de bugs, gerando assim redução de custos e satisfação do cliente.

Tomamos conhecimento sobre os diversos tipos de testes existentes e destacamos os mais utilizados, sendo eles testes unitários, integrados, de sistemas, aceitação e de segurança, mostrando suas principais vantagens e desvantagens, bem como diversas abordagens e metodologias utilizadas.

Recursos Adicionais

Artigos:

Título:Desenvolvimento de software dirigido por teste de aceitação
Autor: Fabiana Braga
http://hdl.handle.net/1843/BUOS-94LMVC

Título:Tipos de testes
Autor: Sten Pittet

https://www.atlassian.com/br/continuous-delivery/software-testing/types-of-software-testing#:~:text=Eles%20consistem%20em%20testar%20m%C3%A9todos,um%20servidor%20de%20integra%C3%A7%C3%A3o%20cont%C3%ADnua.

Título: Introdução a Teste de Software
Autor: Arilo Cláudio Dias Neto

Disponível em: <https://www.researchgate.net/profile/Arilo-Neto/publication/266356473_Introducao_a_Teste_de_Software/links/5554ee6408ae6fd2d821ba3a/Introducao-a-Teste-de-Software.pdf>. Acesso em: 23 jun. 2024a.
Disponível em: <https://www.researchgate.net/profile/Arilo-Neto/publication/266356473_Introducao_a_Teste_de_Software/links/5554ee6408ae6fd2d821ba3a/Introducao-a-Teste-de-Software.pdf>. Acesso em: 23 jun. 2024a.
Disponível em: https://www.researchgate.net/profile/Arilo-Neto/publication/266356473_Introducao_a_Teste_de_Software/links/5554ee6408ae6fd2d821ba3a/Introducao-a-Teste-de-Software.pdf

Livros:

Introdução ao teste de software, autores do livro Márcio Eduardo, José Carlos Maldonado ,Mário Jino.
Simplificando testes de software, autora: Anne Caroline Rocha

Autores:

Michael Bolton
Andy Grove
Tales Augusto De Vieira Santos

REFERÊNCIA

([s.d.])
Disponível em: <https://repositorio.ufmg.br/bitstream/1843/BUOS-94LMVC/1/fabianabragadeassis.pdf>. Acesso em: 28 jun. 2024b.

VIEIRA, H. Testes de integração: o que é, quais os tipos e como automatizar? Disponível em: <https://www.objective.com.br/insights/teste-de-integracao/>. Acesso em: 28 jun. 2024.

PACHECO, H. Test-Driven Development: Saiba os benefícios para testes automatizados. Disponível em: <https://www.objective.com.br/insights/test-driven-development/>. Acesso em: 28 jun. 2024.

LEE, G. O que é o Behavior Driven Development (BDD)? Disponível em: <https://www.loadview-testing.com/pt-br/blog/o-que-e-o-behavior-driven-development-bdd/>. Acesso em: 28 jun. 2024.

GAMA, A. Test Driven Development: TDD Simples e Prático. Disponível em: <https://www.devmedia.com.br/test-driven-development-tdd-simples-e-pratico/18533>. Acesso em: 28 jun. 2024.

https://l1nq.com/thHTW

Automação de Testes: O que é, por que fazer e como começar - Blog Accurate