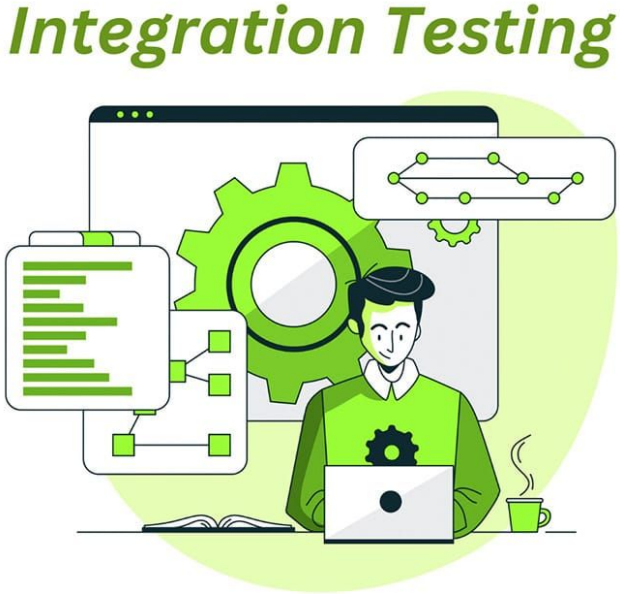
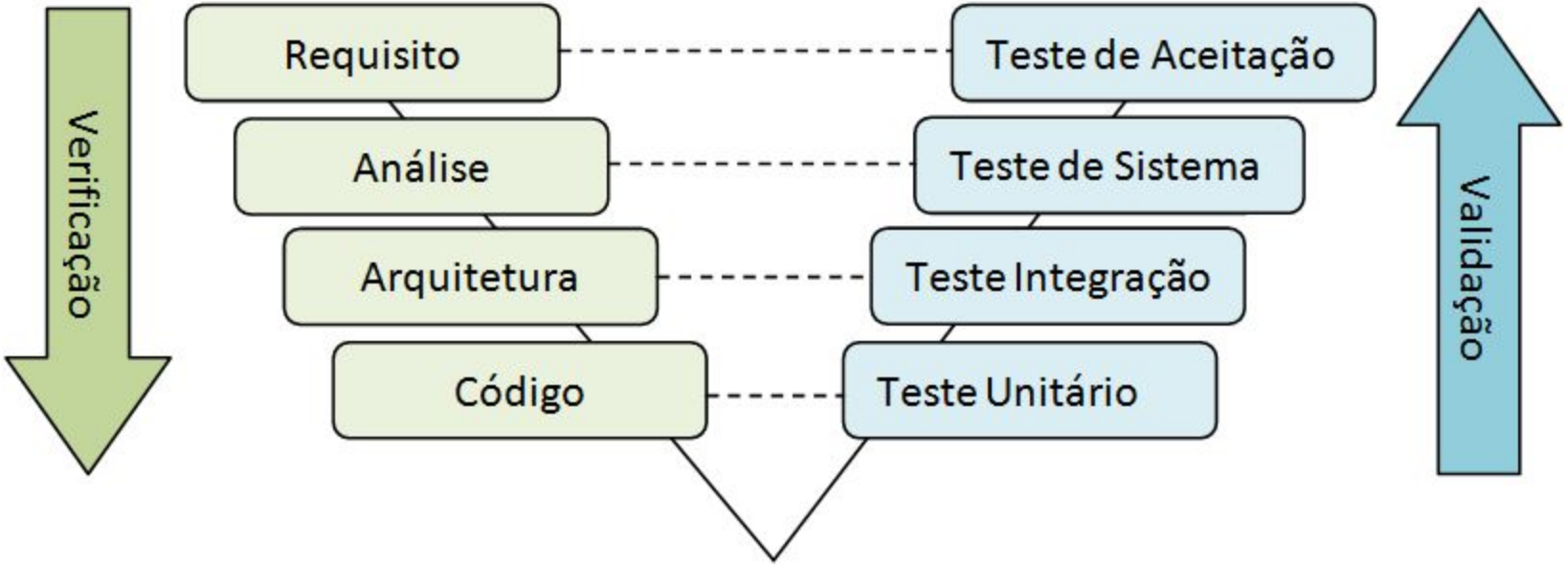


UniCesumar

Projeto, implementação e Teste de Software

Teste de Integração

Prof. Esp. Dacio F. Machado


Verificação
Requisito
Análise
Arquitetura
Código
Teste de Aceitação
Teste de Sistema
Teste Integração
Teste Unitário
Validação

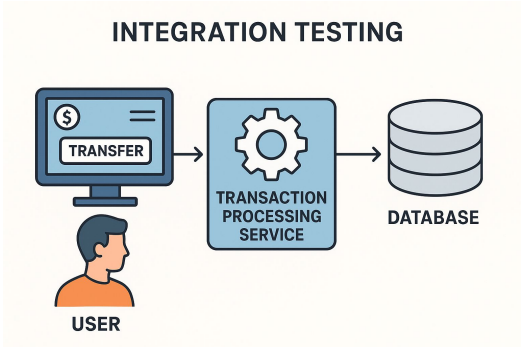
Integração

O teste de integração é uma técnica sistemática para construir a arquitetura de software ao mesmo tempo que conduz testes para descobrir erros associados com as interfaces.

O objetivo é construir uma estrutura de programa determinada pelo projeto a partir de componentes testados em unidade.

Exemplo de teste de integração

Em uma aplicação bancária, o teste de integração envolve verificar a interação entre a interface do usuário, o serviço de processamento de transações e o banco de dados. Quando um usuário inicia uma transferência de fundos, o teste garante que a transação seja processada, os saldos das contas sejam atualizados corretamente e os detalhes da transação sejam registrados de forma consistente em todos os componentes integrados.



Exemplo de teste de integração

Vamos supor que você tenha um aplicativo de e-mail com os seguintes módulos:

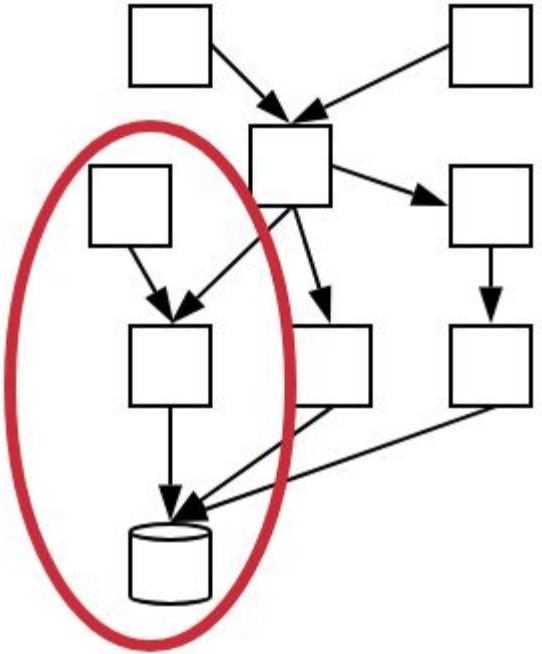
- página de login
- módulo de caixa de entrada
- módulo de exclusão de e-mail

Nesse cenário, você não precisa testar a funcionalidade de páginas individuais. Em vez disso, você testará como cada página se interliga com as outras, como verificar a ligação entre a página da caixa de entrada e a página de exclusão de e-mails. Da mesma forma, a integração entre a página de login e o módulo da caixa de entrada precisa ser verificada.

Esse tipo de teste é realizado por meio de testes de integração.

Exemplo de teste de integração

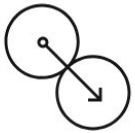
Os **testes de integração** ou **testes de serviços**, que verificam uma funcionalidade ou transação completa de um sistema. *Logo, são testes que usam diversas classes, de pacotes distintos, e podem ainda testar componentes externos,* como bancos de dados. *Testes de integração demandam mais esforço para serem implementados e executam de forma mais lenta.*



Os casos de teste utilizados nos testes de integração ajudam os desenvolvedores a focar em áreas específicas da operação:

Fluxo de dados

Os dados que percorrem um sistema seguem uma trajetória, indo da origem ao destino. Essa informação passa por processamento à medida que atravessa diferentes etapas e componentes do sistema. Esse processo de movimentação é conhecido como fluxo de dados.



Coordenação de interface

Assim como equipes eficazes precisam de liderança, há uma “inteligência superior” que orienta a operação e a interação fluida entre os componentes do software. Chamamos esse processo de gerenciamento de coordenação de interfaces.

Protocolos de comunicação

São os protocolos de comunicação que determinam o modo como os dispositivos trocam informações. Esses protocolos definem as regras para a transferência de dados e determinam como as mensagens devem ser estruturadas. Os protocolos de comunicação também definem como os sistemas devem corrigir falhas quando ocorrem erros.

Integração

Muitas vezes há uma tendência de tentar integração não incremental; isto é, construir o programa usando uma abordagem **big bang**.

Todos os componentes são combinados com antecedência.

O programa inteiro é testado como um todo.

Big Bang Theory

Outra forma importante de realizar testes de integração é por meio da integração tipo big bang. Nesse caso, todas as unidades, componentes e módulos do sistema são integrados e testados de uma só vez, como se formassem uma única unidade.

Big Bang Theory

No entanto, essa forma de teste é limitada. Se o processo mostrar que o sistema não funciona como deveria, o teste big bang não indica quais partes estão falhando na integração.

O teste big bang oferece uma resposta rápida quando o sistema funciona corretamente com todos os seus elementos.

Por que não ?

Um novato no mundo do software pode levantar uma questão aparentemente legítima quando todos os módulos tiverem passado pelo teste de unidade:

“ Se todos funcionam individualmente, porque você duvida que funcionem quando estiverem juntos? ”

Novamente,
Por que não ?

ATIVIDADE

Atividade consiste em simular o desenvolvimento da aplicação através de módulos, cada módulo é de responsabilidade de um time(equipe) aplicando o método estudado anteriormente(TDD), que posteriormente serão integrados como um único sistema.

Assim podemos validar o questionamento levantado no slide anterior se o funcionamento unitário garante o funcionamento integral

Após isso podemos utilizar como estudo de caso para outras técnicas como CI/CD

Descrição da Aplicação

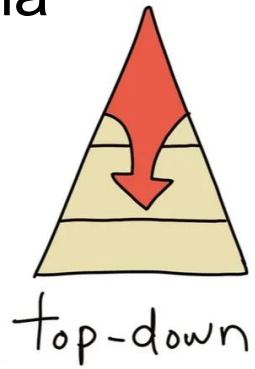
A Universidade Tabajara “TecLearn TABAJARA” uma instituição das Organizações Tabajara, decidiu informatizar o controle de sua biblioteca. O sistema é para o **empréstimo de livros**, **cadastro de usuários**, **controle de acervo** e **relatórios de uso**.

A universidade contratou uma empresa de desenvolvimento júnior (os alunos) para criar o sistema em módulos, cada equipe ficando responsável por uma parte do sistema. A integração e os testes finais definirão se o sistema será aprovado.

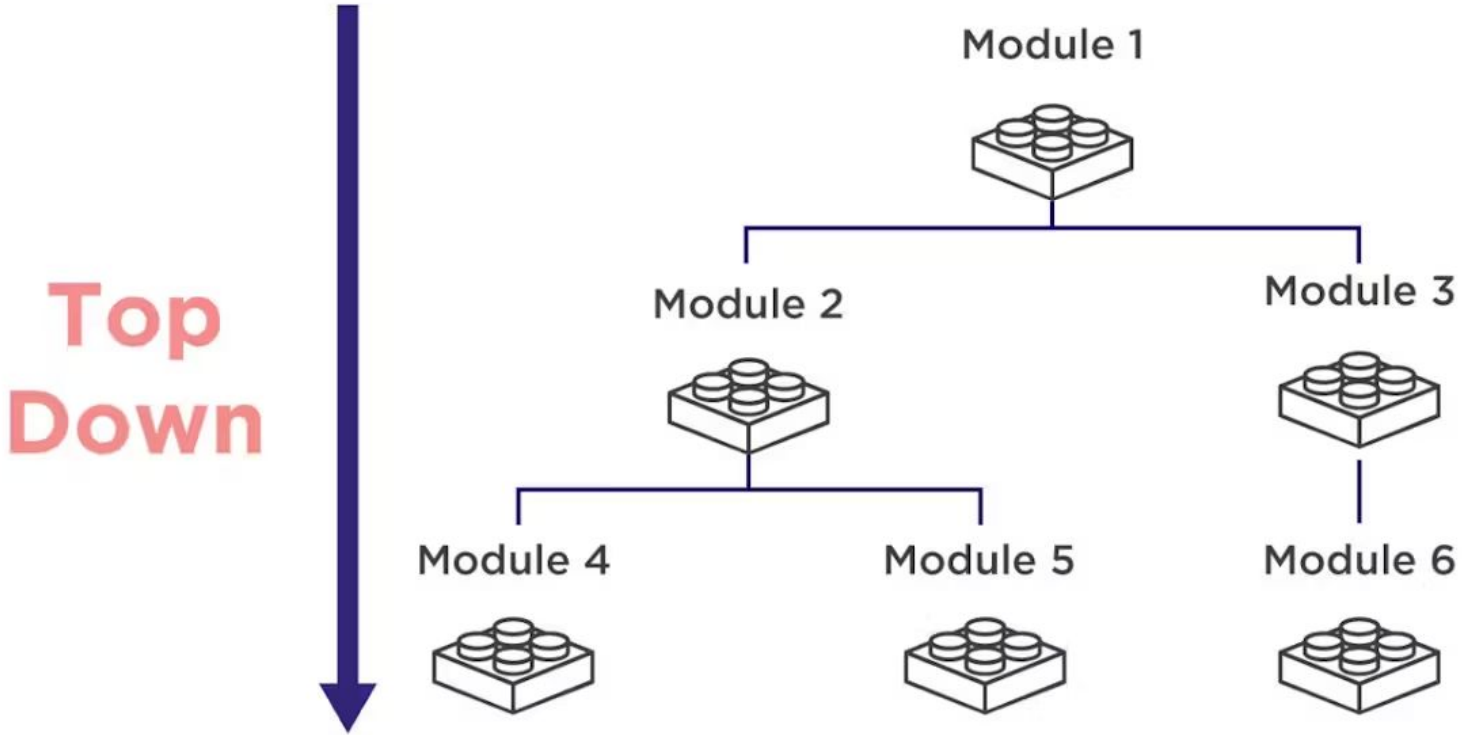
Visão geral do sistema O SGBU faz o controle de usuários, acervo (livros), empréstimos e relatórios. O sistema final será composto pela integração dos módulos desenvolvidos
por cada equipe.

Abordagem de cima para baixo

A integração dos blocos/módulos é avaliada progressivamente de cima para baixo. Blocos individuais são testados escrevendo STUBS de teste. Depois disso, as camadas inferiores são gradualmente integradas até que a camada final seja montada e testada. A integração de cima para baixo é um processo muito orgânico porque se alinha com a forma como as coisas acontecem no mundo real.



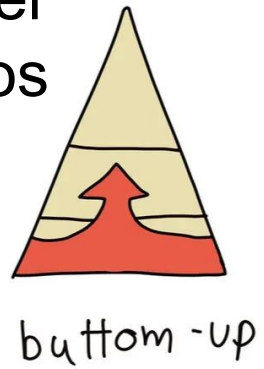
Um **stub** é um módulo real no ambiente de teste, que fornece respostas predeterminadas a chamadas.


Top
Down
Module 1
Module 2
Module 3
Module 4
Module 5
Module 6

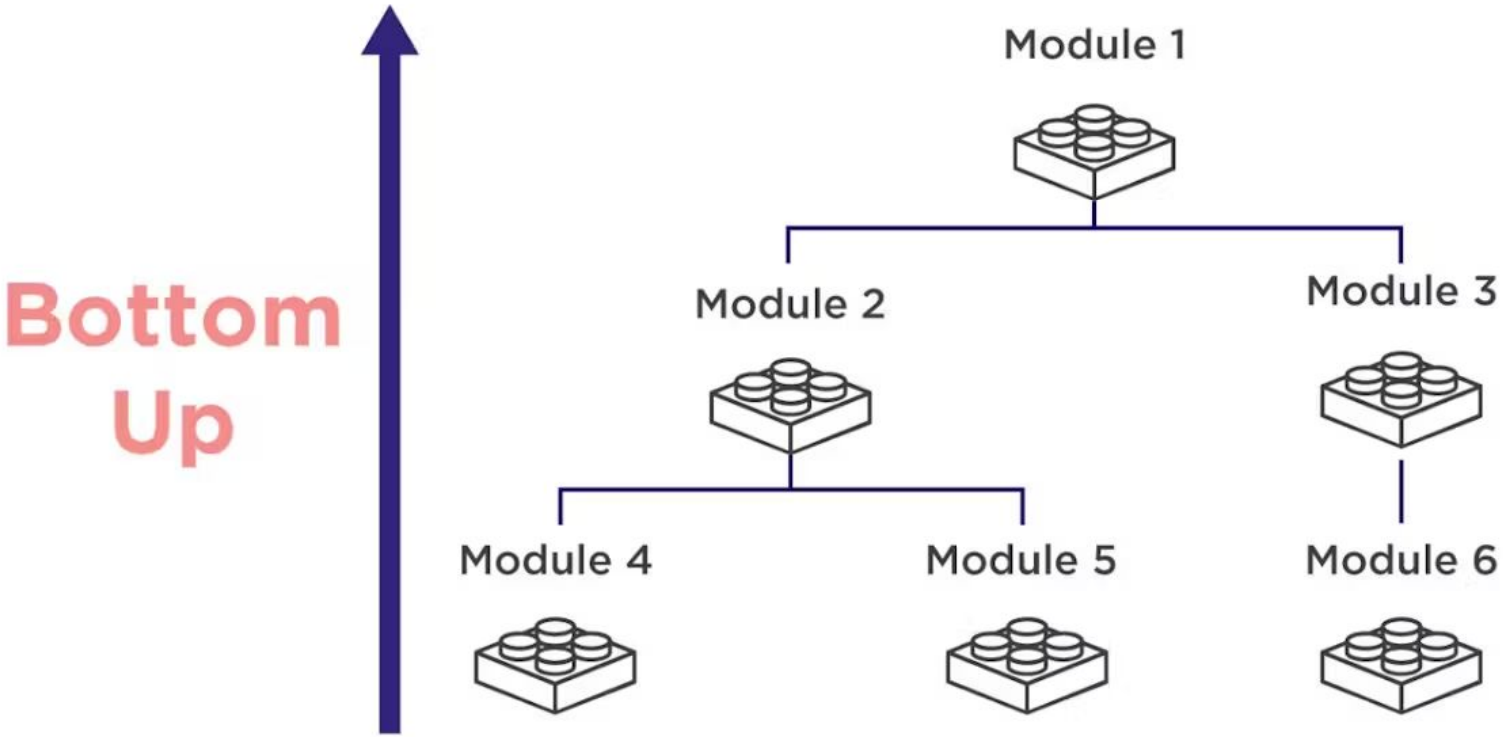
Prós	Contras
Testar primeiro os componentes de alto nível ajuda a detectar problemas precocemente em partes críticas do sistema.	Os stubs são necessários para componentes de nível inferior, o que torna os testes mais complexos.
Isso garante que as principais funcionalidades do sistema funcionem conforme o esperado desde o início do desenvolvimento.	São necessários mais recursos para simular componentes de nível inferior.
A detecção precoce de problemas pode evitar reparos dispendiosos mais tarde.	Os componentes de nível inferior não são testados até mais tarde, o que causa atrasos.

Abordagem de baixo para cima

Os blocos/módulos são testados em ordem crescente até que todos os níveis de blocos/módulos tenham sido combinados e testados como uma unidade. Essa abordagem usa programas estimulantes chamados DRIVERS. Em níveis mais baixos, é mais fácil detectar problemas ou bugs. A desvantagem dessa abordagem é que os problemas de nível superior só podem ser identificados após a conclusão da integração de todos os blocos.



Prós	Contras
Testar primeiro os componentes de nível inferior garante que a base do sistema seja sólida.	Problemas em componentes de nível superior podem não ser detectados até mais tarde, o que dificulta sua correção.
Componentes de baixo nível são mais simples e fáceis de testar, o que os torna mais gerenciáveis.	O processo de teste é lento porque começa com componentes de nível inferior.
Isso reduz o risco de falhas graves no sistema, pois os problemas são encontrados primeiro em componentes individuais.	Interações complexas entre módulos não são testadas precocemente, o que dificulta os testes de integração posteriormente.


Bottom
Up
Module 1
Module 2
Module 3
Module 4
Module 5
Module 6

Métodos de testes de integração

Teste de integração automatizado: parte essencial do processo de desenvolvimento de software, o teste automatizado é mais uma forma de avaliar como os componentes de software funcionam juntos. Esse processo de teste de integração executa casos de teste com base em ferramentas especializadas e scripts automatizados. Assim, é possível identificar e corrigir problemas de integração antes da implementação. E por ser automatizado, todo o sistema se torna mais eficiente e ágil. O teste automatizado é um elemento fundamental da integração contínua, prática de DevOps que depende de um repositório compartilhado com atualizações constantes no código.

Métodos de testes de integração

Teste de caixa-preta: a analogia (ideia) do teste de caixa-preta pode ser aplicada a qualquer situação em que se entende que o funcionamento interno da caixa-preta (seja aquele que envolve código de computador ou algum outro aspecto operacional, como lucros divulgados da empresa) não está sujeito a ser totalmente compreendido . No caso do teste de integração de caixa-preta, isso significa que os testadores não querem se debruçar sobre os códigos específicos usados em diferentes módulos; em vez disso, preferem uma resposta mais simples e rápida sobre se os sistemas, componentes e módulos funcionam em harmonia.

Métodos de testes de integração

Teste de caixa branca: em contraste direto com o teste de caixa preta, o teste de caixa branca pressupõe que os testadores desejam examinar o código relevante durante o processo de teste na esperança de identificar áreas problemáticas e fazer alterações corretivas no código para depurar esses problemas, mesmo que essa abordagem quase certamente seja mais demorada. A frase "caixa branca" reflete esse desejo de clareza no funcionamento interno do sistema, embora a expressão "caixa transparente" possa ser uma descrição mais precisa do que os testadores procuram.

Métodos de testes de integração

Teste de ponta a ponta: Como o nome sugere, o teste de ponta a ponta (às vezes chamado de teste E2E) oferece aos testadores uma maneira de verificar as funções de todo o sistema, do início ao fim. Além disso, os testes E2E podem imitar cenários de teste do mundo real e preparar o terreno para os testes de integração, incorporando planos de teste que determinam quais unidades serão testadas. O teste E2E costuma surgir em uma fase posterior do processo de integração, depois da conclusão dos testes de integração e antes do teste de aceitação pelo usuário

Ferramentas de testes de integração

São diversas ferramentas e frameworks de testes de integração:

Citris: o Citris atende à enorme base de usuários do Java (o que o torna uma das linguagens de programação mais populares do mundo) com um framework JavaTM de código aberto. O Citris consegue gerenciar o uso de APIs (como transações) e gerar mensagens de teste.

Katalon: o software de testes automatizados do Katalon Studio incorpora o framework de código aberto **Selenium**, uma ferramenta baseada em navegador que permite escrever scripts de teste em várias linguagens, como JavaScript, NodeJS e Python.

Ferramentas de testes de integração

Carteiro: o teste de integração de API é bem servido pelo Postman. A ferramenta também se destaca na maneira como permite a colaboração e acomoda a automação. Os usuários também conseguem escrever um teste sem a obrigação de armazená-lo em uma coleção ou salvá-lo. O testador faz uma solicitação no Postman e recebe uma URL correspondente.

SoapUI: o SoapUI oferece uma ferramenta de código aberto para testar aplicações web e realizar testes de integração. Com essa ferramenta, os testadores contam com uma interface gráfica que permite criar casos de teste e trabalhar com os dados de teste de forma prática.

Melhores práticas para testes de integração

Utilize ferramentas de teste automatizadas : Utilize ferramentas como o Postman para testes de API ou o Selenium para testes de aplicações web para automatizar testes de integração repetitivos. A automação ajuda a acelerar o processo de teste e a melhorar a precisão.

Priorize as interfaces críticas : concentre-se em testar primeiro as integrações mais críticas, como a conexão entre o gateway de pagamento e o sistema de gerenciamento de pedidos em um aplicativo de comércio eletrônico, para garantir que as funcionalidades principais sejam robustas.

Boas práticas para testes de integração

Crie casos de teste abrangentes : Desenvolva casos de teste detalhados que cubram vários cenários de integração, incluindo casos extremos e condições de erro. Por exemplo, teste como o aplicativo se comporta quando uma resposta da API é atrasada ou retorna um erro inesperado.

Utilize técnicas de mocks e stubs : Ao testar integrações com serviços externos (como APIs de terceiros), use mocks ou stubs para simular o comportamento desses serviços. Essa abordagem permite realizar testes sem depender dos serviços externos reais, reduzindo as dependências.

Integração Contínua

OBRIGADO

dacio.francisco@unicesumar.edu.br