UniCesumar

Projeto, implementação e Teste de Software

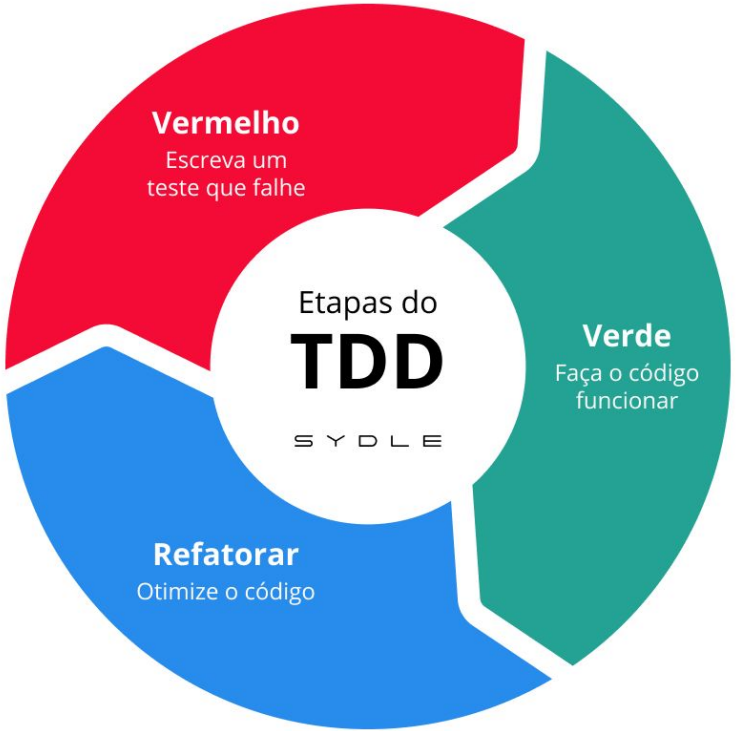
REVISÃO 2° BIMESTRE

Prof. Esp. Dacio F. Machado

Test-Driven Development

TDD significa Test-Driven Development, que é uma abordagem de desenvolvimento de software que coloca um forte foco na escrita de testes antes de escrever o código real.

Ciclo de desenvolvimento Red,Green, Refactor.



Testar algo que não existe

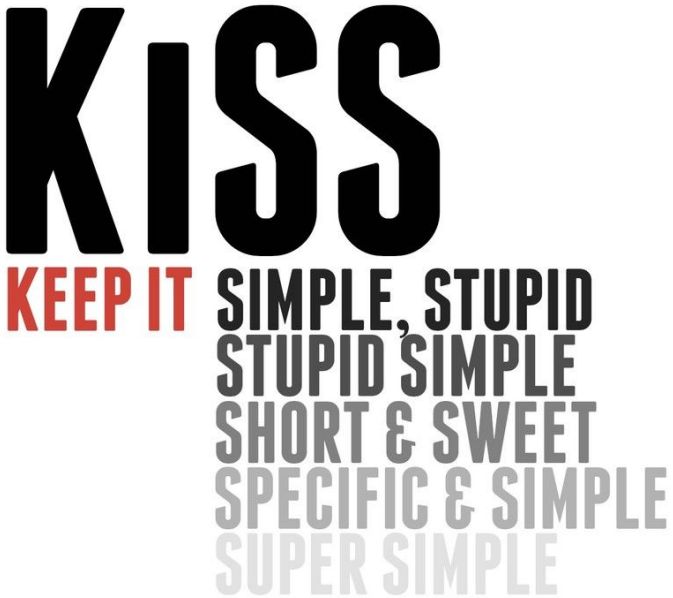
Criar algo que passe no teste.

Melhorar o que foi criado

Keep It Simple, Stupid

é um princípio de design que defende a simplicidade e a evitação de complexidades desnecessárias em sistemas e projetos.

A ideia central é que os sistemas funcionam melhor quando são mantidos simples, e essa filosofia se originou com a Marinha dos EUA em 1960



Integração

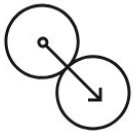
O teste de integração é uma técnica sistemática para construir a arquitetura de software ao mesmo tempo que conduz testes para descobrir erros associados com as interfaces.

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

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.

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 ?

Exemplo de teste de integração

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

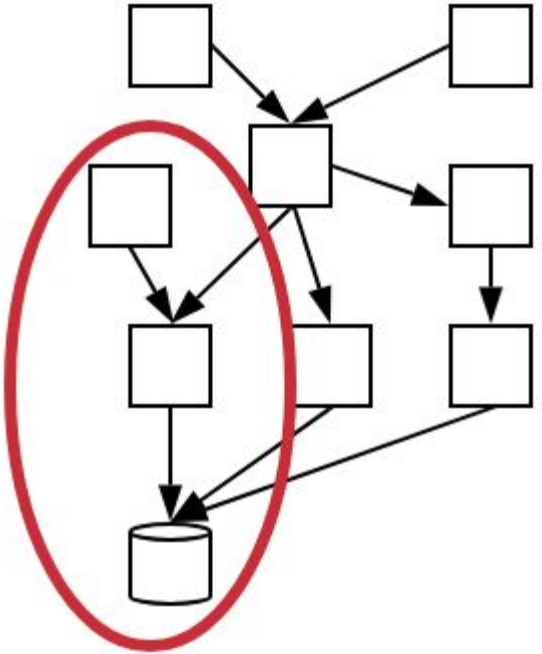
- página de login
- módulo de caixa de entrada
- módulo de exclusão de e-mail

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.

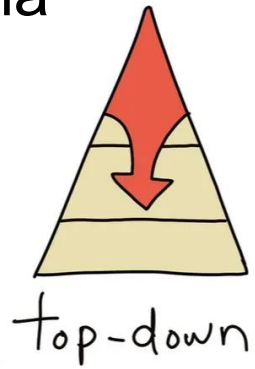
Exemplo de teste de integração

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.*

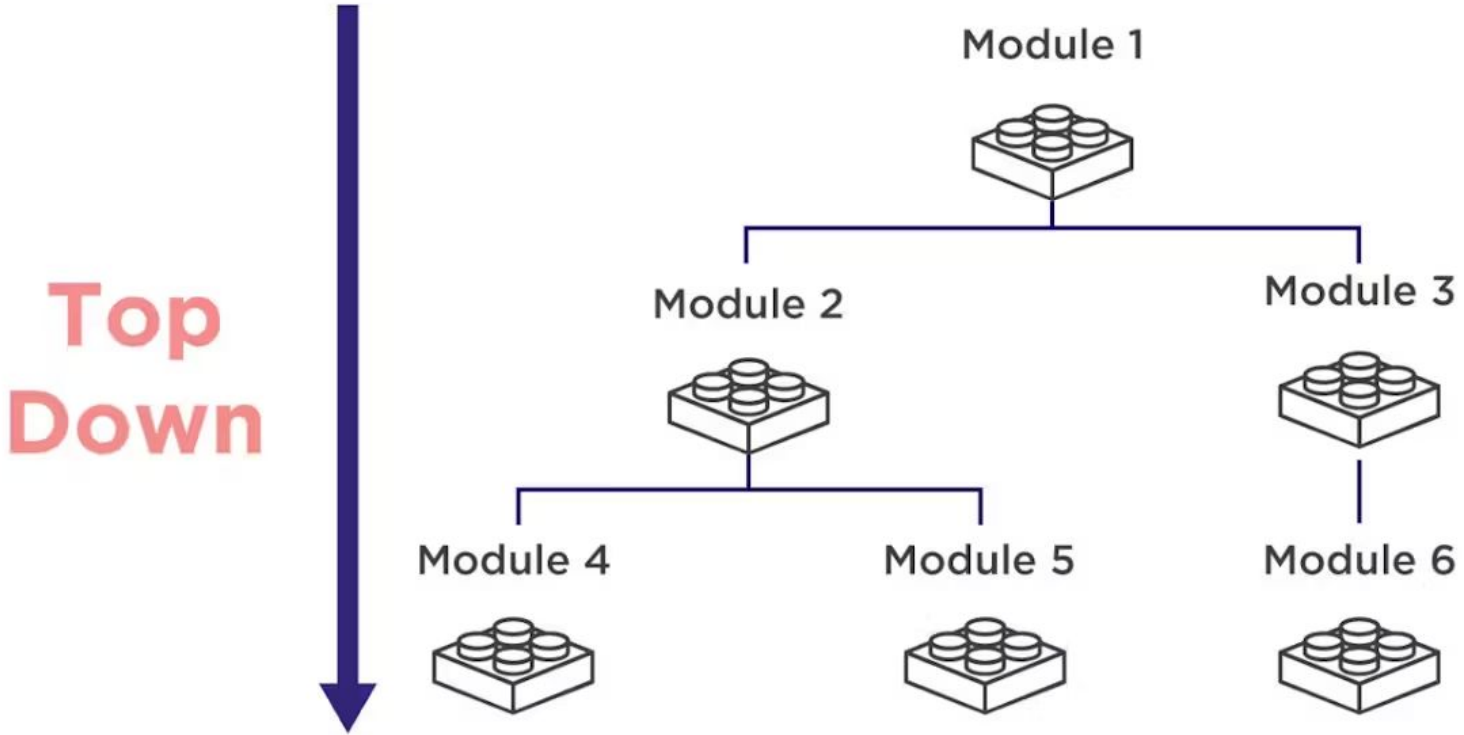


Abordagem de cima para baixo

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.



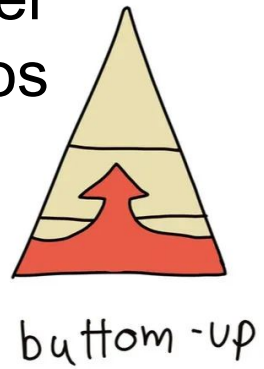
Um **stub** é um módulo real no ambiente de teste, que fornece respostas predeterminadas a chamadas.


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

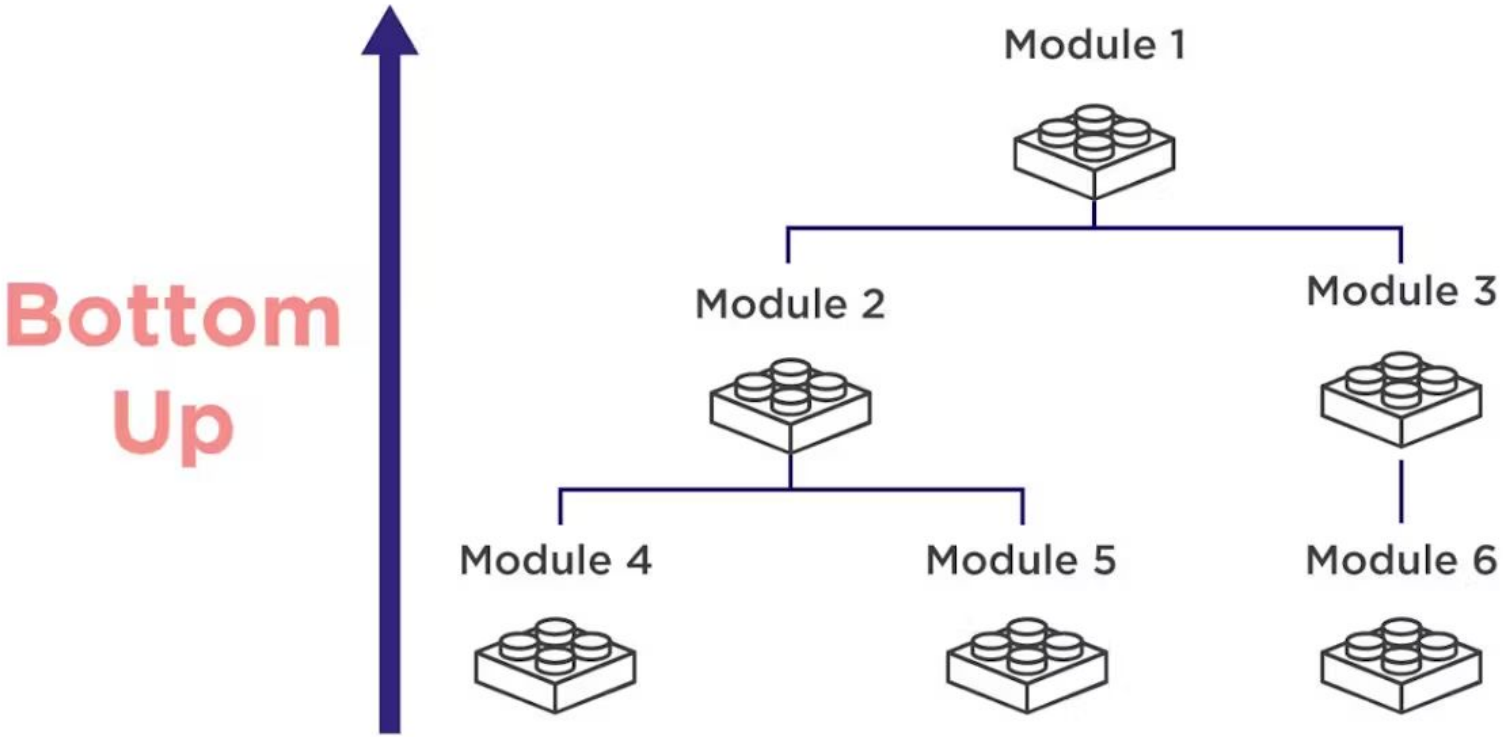
Prós	Contras
Testar primeiro os componentes de alto nível ajuda a detectar problemas precocemente em partes críticas do sistema.	Os stubs são necessários para componentes de nível inferior, o que torna os testes mais complexos.
Isso garante que as principais funcionalidades do sistema funcionem conforme 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 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.

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.

Características Dos Testes Automatizados

1. **Conciso**: deve ser simples e direto, sempre que possível.
2. **Explícito**: deve relatar claramente quaisquer falhas ou desvios.
3. **Replicável**: pode ser executado várias vezes com os mesmos resultados.
4. **Robusto**: resiste a interferências externas e produz resultados consistentes.
5. **Necessário**: detecta divergências entre o esperado e o implementado.
6. **Clareza/Manutenção**: possui código compreensível e fácil de manter.
7. **Eficiente**: tem bom desempenho durante a execução.
8. **Independente**: não depende de outros testes e pode ser executado isoladamente.
9. **Rastreável**: está ligado diretamente aos requisitos e suas origens.

Relembrando o Processo de Desenvolvimento

O ciclo de vida do desenvolvimento de software (SDLC) é um processo estruturado que as equipes de desenvolvimento seguem para criar software com qualidade, de forma econômica e segura. Suas etapas são:

- **Análise**
- **Planejamento**
- **Design (Projeto)**
- **Implementação (Codificação)**
- **Testes**
- **Implantação (Entrega)**
- **Manutenção**

Essas fases estão geralmente conectadas e podem ser realizadas em sequência ou simultaneamente, conforme o modelo adotado.

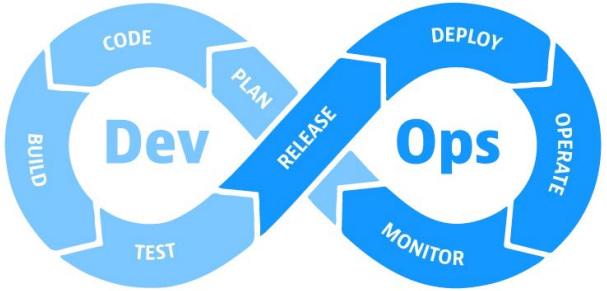
Integração Contínua (CI)

A integração contínua (CI) é um processo de desenvolvimento de software em que os desenvolvedores integram novo código à base de código durante todo o ciclo de desenvolvimento.

A **CI** foi uma resposta aos desafios dos processos associados à integração e implementação e são fundamentais para as práticas modernas de DevOps.

Ajudam a simplificar o processo de criação, fornecendo feedback rápido sobre o desempenho da integração.

Faz com que as equipes detectem e corrijam erros antes que eles afetem o software



Vantagens e Desafios

Validação contínua: cada mudança no código é verificada para evitar falhas e regressões.

Feedback rápido: os testes retornam resultados mais rapidamente que os testes manuais.

Correções mais eficientes: bugs são resolvidos com mais facilidade logo após serem introduzidos.

Maior confiabilidade: elimina erros humanos ao repetir tarefas de teste.

Execução em paralelo: permite escalar os testes para ganhar tempo, conforme a infraestrutura disponível.

Vantagens e Desafios

Embora a automação de testes elimine muitas tarefas monótonas e repetitivas, ela não torna desnecessária a equipe de QA. Em vez de gastarem tempo em tarefas repetitivas, podem se concentrar em definir casos de teste, escrever testes automatizados e aplicar sua criatividade e engenhosidade em testes exploratórios.

Falsos positivos/negativos: testes instáveis ou mal definidos geram falhas injustificadas ou deixam passar erros reais.

Manutenção constante: mudanças no software exigem atualizações frequentes nos testes, aumentando o custo de manutenção.

Tempo de execução elevado: muitos testes ou testes lentos tornam o pipeline demorado e dificultam a agilidade.

Ambiente instável: dependências externas tornam os testes inconsistentes; é necessário isolar e estabilizar o ambiente.

Desenvolvimento orientado por teste

- Em um pipeline de IC com teste automático, é sempre interessante melhorar a cobertura do teste.
- Cada nova função que segue pelo pipeline de IC deve ser acompanhado de um conjunto de testes para assegurar que o novo código se comporte como o esperado.
- O Test Driven Development (Desenvolvimento orientado por teste, TDD) é uma prática de escrever o código de teste e os casos de teste antes de fazer qualquer código de função real.
- Como parte do processo de testes automatizados em CI, o desenvolvimento orientado por testes cria o código de forma iterativa e testa um caso de uso por vez; e notifica as equipes sobre o desempenho do código em todas as áreas funcionais da aplicação.

A TODOS,

BOA SORTE.

OBRIGADO

dacio.francisco@unicesumar.edu.br