Engenharia de Software

Tecnologia de Software

Aula 5: Implementação e teste

Fábio Levy Siqueira
levy.siqueira@usp.br

Implementação

- Responsabilidades
  - Gerar o código fonte
    - Algoritmo adequado / detalhes tecnológicos
    - Pode envolver diversas linguagens
  - Executar alguns níveis de teste
  - Integração
  - Gerar executáveis, bibliotecas e componentes

- A implementação é baseada no projeto

Implementação

- Preocupações (SWEBOK, 2014)
  - Minimizar complexidade
    - Correção, Eficiência e Elegância (legível)
  - Antecipar mudanças
    - Desperdício X Retrabalho
    - (Responsabilidade maior da arquitetura)
  - Permitir a verificação
    - Facilitar as atividades de Verificação e Validação
    - Facilitar a depuração

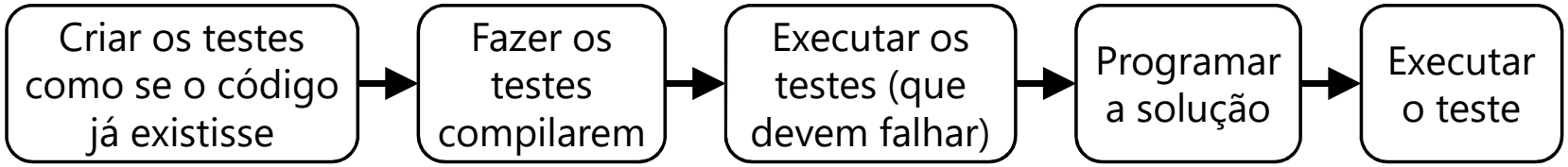
Implementação

- Preocupações
  - Reuso
    - Usar bibliotecas, frameworks, código e componentes
    - Criar elementos reusáveis → Planejamento
  - Padronização
    - Padrões de código
      - Uso de recursos
      - Comunicação entre elementos de software
      - Acesso a plataformas
    - Documentação
    - Ferramentas

Implementação e testes

- Programador faz alguns níveis de teste

- Test-Driven Development (TDD)
  - Testes de unidade são feitos antes do código
  - Abordagem usada pelo *Extreme Programming*



Integração

- Juntar o trabalho dos programadores
  - Gera uma versão "completa" do software
  - Baseado na versão atual de cada item
  - Verifica o trabalho dos desenvolvedores
    - As partes do software funcionam em conjunto?
- Juntar o software a outros elementos
  - Software e hardware

- Importância da gerência de configuração

Integração

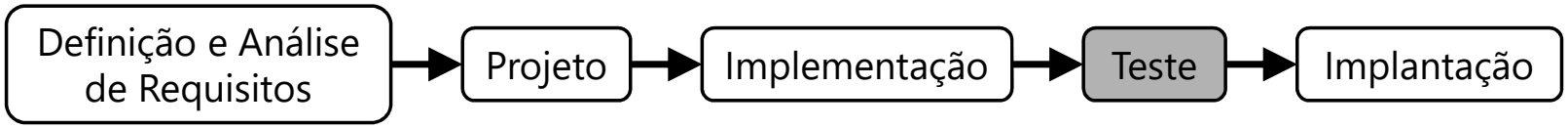
- O ideal é integrar com alta frequência
  - ...mas a integração tem um custo envolvido...
  - Algumas questões a considerar
    - Complexidade do software (e de seus módulos)
    - Complexidade da integração
      - Equipes distribuídas
    - Duração das iterações

- Pode ser necessário um plano de integração
  - Dependência entre componentes

Teste

Teste

- Atividade de controle da qualidade do produto



- Parte do processo de Verificação e Validação
  - Forma mais comum de V&V
  - (Aula 10)

Teste

- O que é teste?

  "Teste é o processo de executar um programa com o objetivo de encontrar erros" (MYERS, 2012, p.6)

Teste

- Então por que não testar **tudo**?
  - *Exemplo*: Cálculo do preço de um produto em uma loja

```
double getValorComJuros(double valor, double taxa, int prazo)
{...}
```

Estratégias de teste

Estratégias de teste

- Ajudam a escolher conjuntos de dados que têm maior probabilidade de existência de erros
  - Diminuem o número de casos de teste

- Duas estratégias principais
  - **Estrutural**: caixa branca
  - **Funcional**: caixa preta

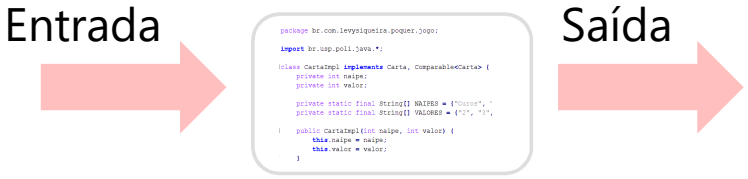
Teste caixa preta

- Alvo do teste como uma caixa preta
  - Não se usa nenhum detalhe interno

- Foco na <u>especificação</u>
  - A saída obtida para uma determinada entrada é a esperada?

Teste caixa branca

- O teste é feito considerando o conhecimento da estrutura interna do código



- Foco nos caminhos
  - Condições, laços e comandos

Estratégias de teste

- Alguns tipos de defeitos só são encontrados aplicando uma das estratégias
  - **Caixa branca**: não se percebe a falta de caminhos
    - Exemplo: faltou considerar alguma condição
  - **Caixa preta**: não se percebe alguns tipos de falhas da lógica interna
    - Exemplo: considerou-se um caso a mais

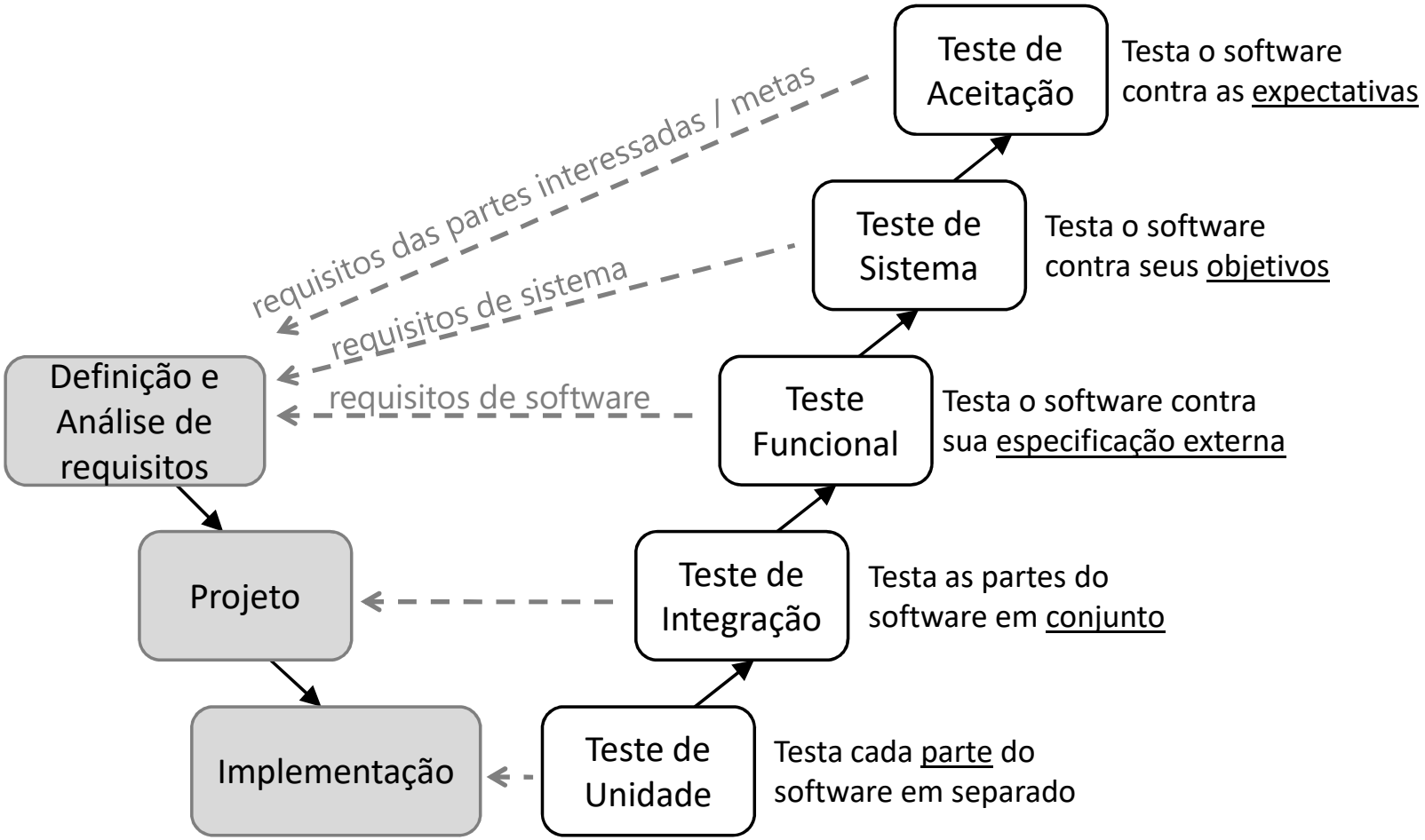
Níveis de Teste

Teste

- A forma de realizar os testes depende do processo
- Normalmente o teste é realizado em vários níveis
- **Teste em V**
  - Cada teste considera aspectos de uma atividade de desenvolvimento

Observação: não há um consenso sobre os nomes dos níveis de teste. Usarei a nomenclatura de Myers (2012)

Teste em V


Teste de Aceitação
Testa o software contra as expectativas
requisitos das partes interessadas / metas
Teste de Sistema
Testa o software contra seus objetivos
requisitos de sistema
Definição e Análise de requisitos
requisitos de software
Teste Funcional
Testa o software contra sua especificação externa
Projeto
Teste de Integração
Testa as partes do software em conjunto
Implementação
Teste de Unidade
Testa cada parte do software em separado

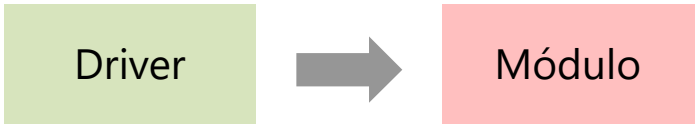
Teste de unidade

- Testa cada módulo do software
  - Menor elemento do software
  - Comparar sua função à interface definida
- Tradicionalmente feito pelo desenvolvedor

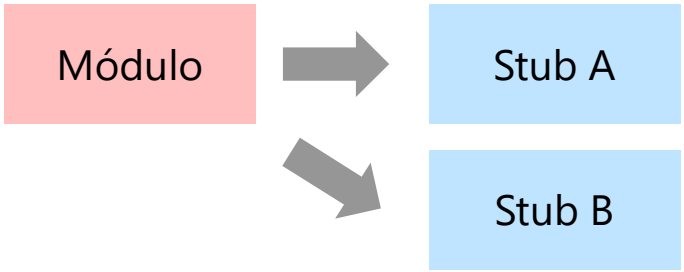
- Algumas questões
  - Qual é a cobertura ideal? É 100%?
  - Deve ser funcional ou estrutural?
  - Quando criar o teste de unidade?
    - Antes de programar a classe (TDD)
    - Depois de programar a classe

Teste de unidade

- Podem ser necessários *drivers* e *stubs*
  - *Driver*: elemento que aceita dados de entrada e os passa para o elemento a ser testado

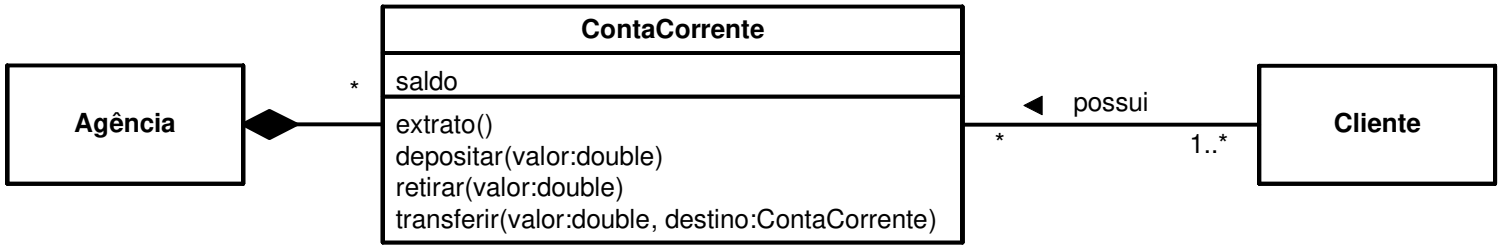


  - *Stub*: substitui um elemento que é chamado
    - Implementação simples, apenas para testar
    - O termo mais genérico é *dublê*



Teste de unidade OO

- *Exemplo*: testar a ContaCorrente
  - Qual é a unidade do teste?



    - É necessário um *driver*? Quais os *stubs* necessários?
    - Existem diversos detalhes...

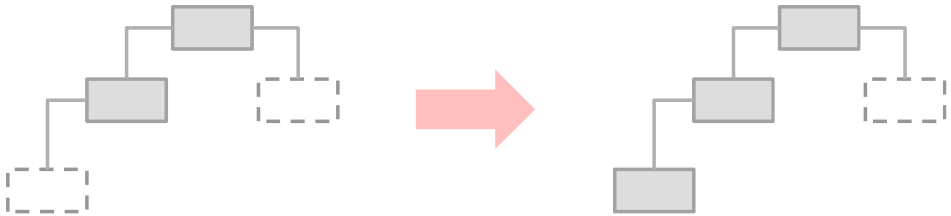
Teste de integração

- Testa as partes trabalhando em conjunto
  - Trata de problemas com as interfaces dos elementos
  - Considera outros componentes / serviços
  - Também chamado de ***teste de serviço***

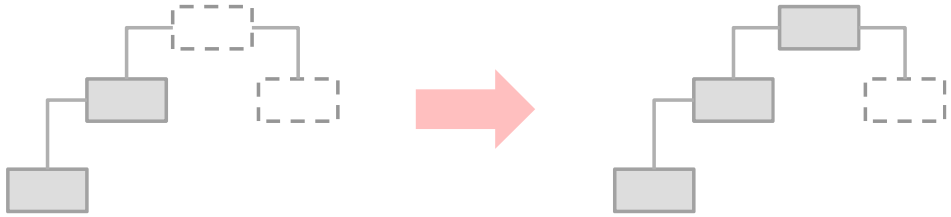
- Duas estratégias principais
  - *Big bang*
    - As partes são integradas de uma vez
    - Problema: identificar a origem de um defeito
  - *Incremental*
    - As partes são integradas aos poucos
    - Top-Down ou Bottom-Up

Teste de integração

- Top-down
  - As partes com dependência são testadas primeiro



- Bottom-up
  - As partes sem dependência são testadas primeiro



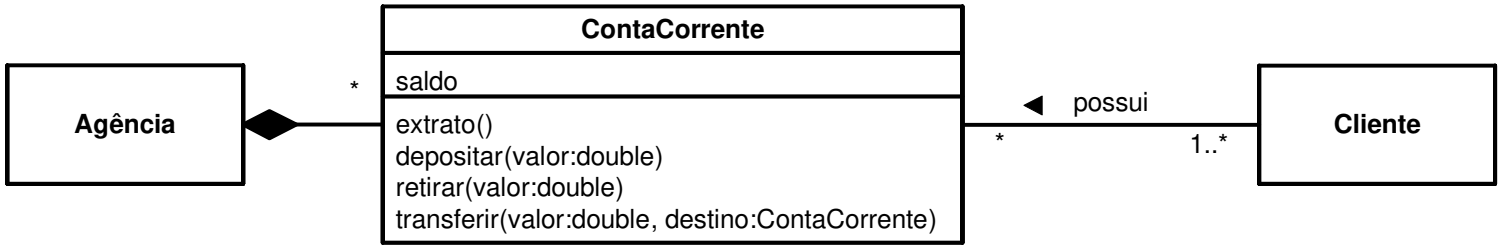
- Cada estratégia tem vantagens e desvantagens

Teste de integração OO

- Existem estratégias específicas para OO
  - Algumas estratégias
    - Teste baseado em *Thread*
      - Integra as classes que tratam de uma "thread de funcionalidade" / evento específico do sistema
      - (Teste de *serviço*)
    - Teste baseado em Uso
      - Testa primeiro as classes independentes e adiciona as classes dependentes
      - Similar à integração Bottom-Up

Teste de integração OO

- *Exemplo*



- *Teste baseado em thread*: transferir
- *Teste baseado em uso*: começa por qual classe?

Teste de integração

- Necessário testar integração com o BD / serviços externos
  - No teste de unidade usamos **dublês**
    - Dummy, Stubs e Mock

Teste funcional

- Procura por diferenças entre o software e seu comportamento externo
  - Testa o software como um todo
  - Normalmente realizado como um teste caixa-preta
  - Também chamado de teste de UI

- Trata da especificação do software
  - Requisitos funcionais

Teste funcional

- *Exemplo*: caso de uso

UC1. Processa venda

...

Fluxo básico

1. Para cada produto da venda:
    a. O Caixa informa o código do produto.
    b. O Software apresenta a descrição e o preço do produto corrente.
    c. O Caixa confirma o produto.
2. O Caixa finaliza a venda.
3. O Software apresenta o total e solicita a forma de pagamento.
4. O Caixa informa que o pagamento será em dinheiro e informa o valor pago.
5. O Software apresenta o troco, imprime o recibo e finaliza a venda.

...

Teste de sistema

- Testa o software frente aos seus "objetivos"
  - Atendimento aos requisitos do sistema
    - Software como parte do sistema

- Sistemas intensivos de software
  - *Exemplo*: sistema web
    - Diversos navegadores e SOs
  - Tradicionalmente aborda os requisitos não funcionais

Teste de sistema

- Alguns tipos de teste
  - Teste de stress
  - Teste de usabilidade
  - Teste de segurança
  - Teste de desempenho
  - Teste de documentação
- Requisitos funcionais
  - Metas/objetivos do cliente documentados

Teste de aceitação

- Testa o software frente às expectativas
  - Realizado pelos clientes/usuários
  - Apoiados por desenvolvedores
- Usado para aceitação do software pelo cliente
  - Teste formal: contrato

- O resultado obtido muitas vezes tem que ser negociado
  - Aceitação (total ou com alterações) ou rejeição

Teste de aceitação

- Alguns tipos de teste de aceitação
  - Teste alfa
    - Uso do software dentro do ambiente de desenvolvimento
    - Em geral para software personalizado
  - Teste beta
    - Usuários em seu ambiente normal
      - Permite analisar diversos ambientes
    - Forma de *marketing*
    - Em geral para software de prateleira

Bibliografia complementar

- MYERS, G. J. **The Art of Software Testing**. John Wiley & Sons, 3ª edição, 2012.
- BOURKE, P.; FAIRLEY, R. E. **SWEBOK – Guide to the Software Engineering Body of Knowledge**. v. 3.0. IEEE, 2014.