Escopo de Testes

De testes em pequena escala aos testes em larga escala

Escopo de Testes

De testes em pequena escala aos testes em larga escala

Apresentação

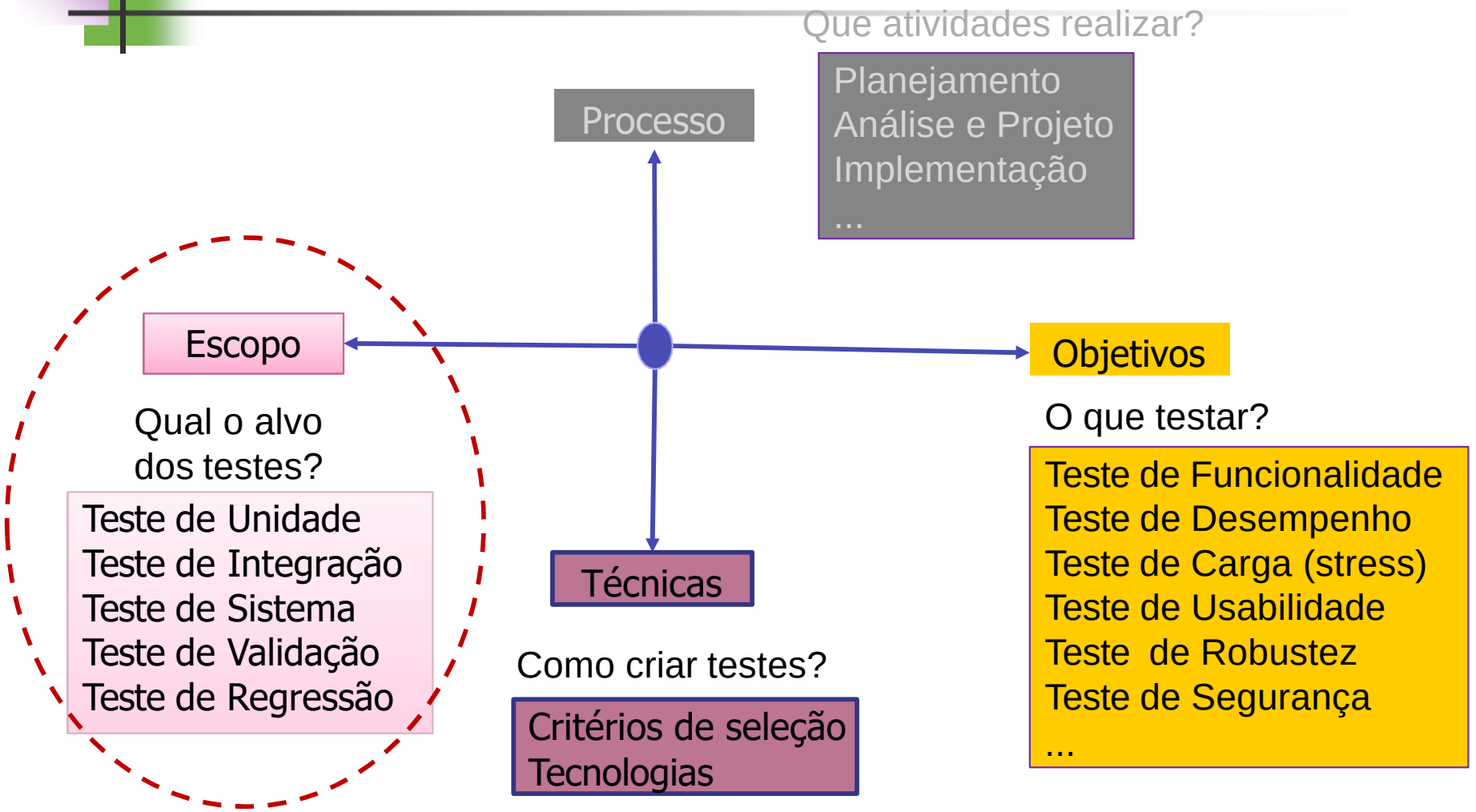
Objetivos

- Entender a divisão dos testes em escopos, indo dos testes em pequena escala (funções, components) aos testes em larga escala (sistemas, validação)
- Conhecer estratégias para testes quando há modificações

Tópicos

- Testes de Unidades
- Testes de Integração
- Testes de Sistemas
- Testes de Validação
- Testes de Regressão

Dimensões dos testes

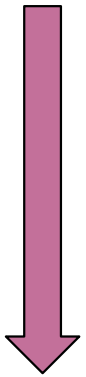


Escopo de testes

- Descreve a granularidade do Item em Teste (IeT)
- Determina ordem de execução dos testes:

Testes de Regressão

1. Testes de Unidades
2. Testes de Integração
3. Testes de Sistemas
4. Testes de Validação

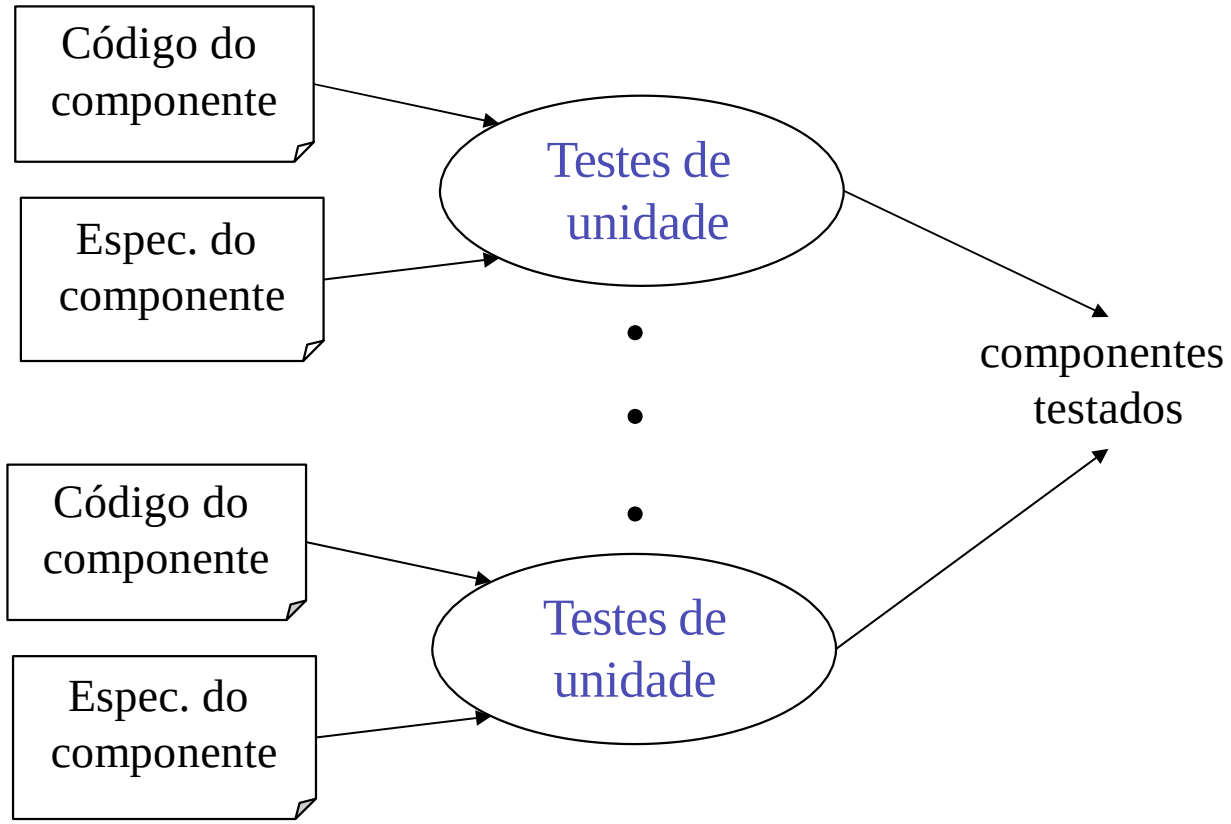


pequena escala

larga escala

[Pressman2006, cap13]

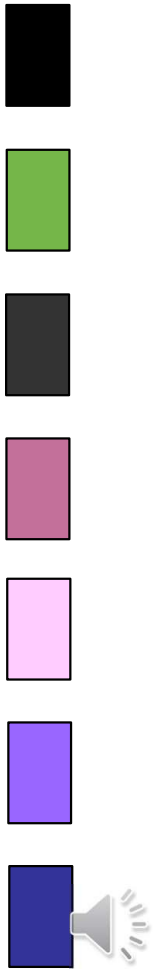
Testes de Unidades



Objetivo

- Revelar a presença de defeitos em uma **unidade** do sistema

- Item em Teste (IeT) = unidade:
  - Entidade executável independente.
  - Seu código fonte pode ou não ser conhecido: depende se os testes estão sendo feitos pelo fornecedor ou pelo usuário
  - Pode representar:
    - Uma função.
    - Uma classe ou um tipo abstrato de dados.
    - Um grupo pequeno de classes.
    - Um serviço ou micro-serviço
    - Um componente
    - Um *famework*
    - Um sistema, cujo acesso se dá através de sua interface: gráfica ou API (**Application Programming Interface)**



Características dos Testes de Unidades

- São geralmente criados e aplicados pelo programador com apoio de ferramentas
- São de execução frequente (a cada mudança no código) □ devem rodar rápido

⇩

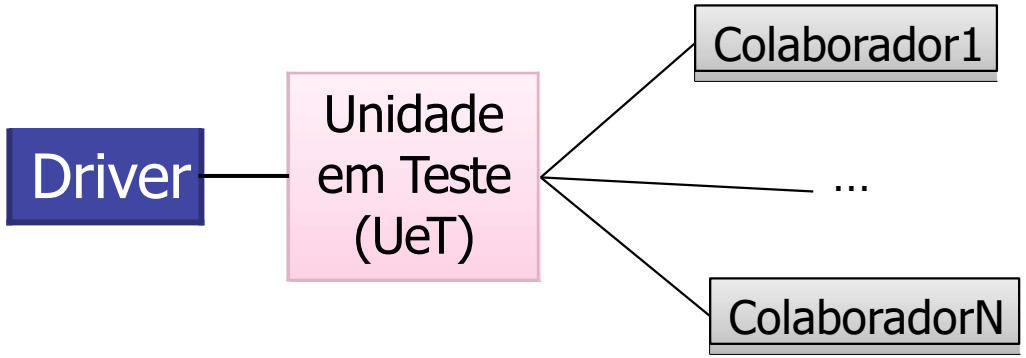
forte candidato a automatização

Mais considerações, veja em: https://martinfowler.com/bliki/UnitTest.html

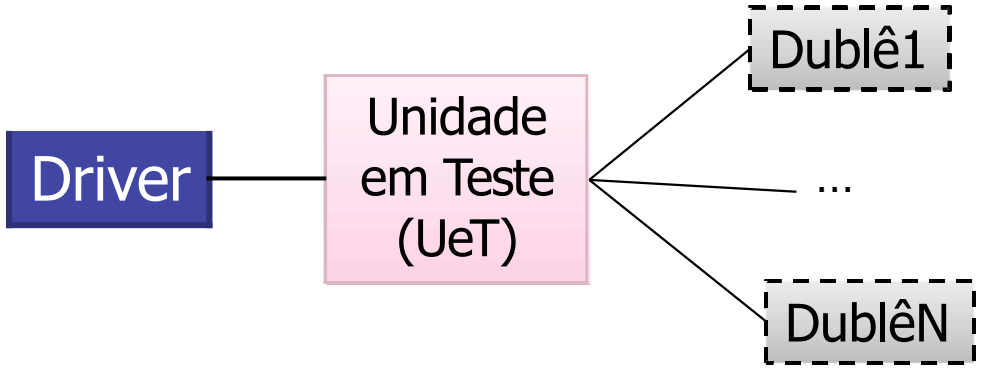
Quais defeitos procurar?

- Os Testes de Unidade visam revelar a presença de defeitos em:
  - interfaces: parâmetros de entrada e saída
  - estruturas de dados: integridade dos dados armazenados
  - condições de limite: a unidade opera adequadamente nos limites estabelecidos?
  - tratamento de erros: a descrição do erro é inteligível? A descrição corresponde ao erro encontrado? O tratamento de exceção é adequado?

Formas de testar

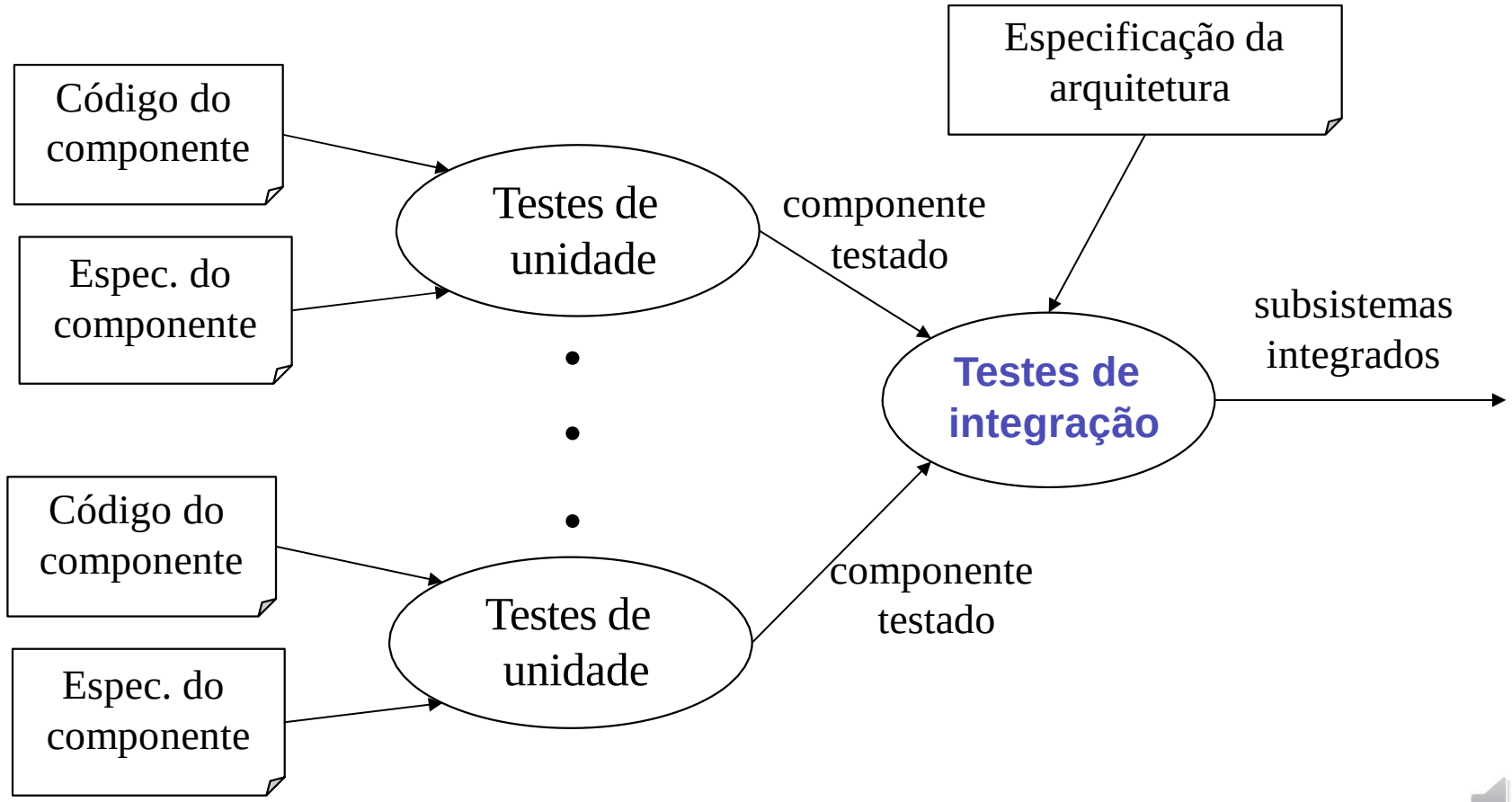


- **Assume que os servidores funcionam bem**
- **Risco de efeitos colaterais**
- **Risco de não-determinismo**
- **Risco de baixo desempenho**

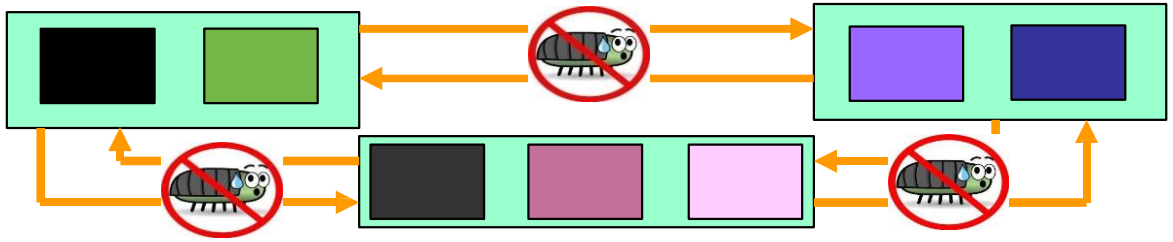


- **Preparação mais demorada**
- **Não tem risco de efeitos colaterais**
- **Melhor controlabilidade da UeT □□risco de não-determinismo**
- **□Velocidade de execução**

Testes de Integração



Objetivos



http://qualidade-de-software.blogspot.com/p/glossario-padrao-de-termos-utilizados.html

Teste de Integração visa expor **defeitos** nas **interfaces** e nas **interações** entre **componentes** ou **sistemas** integrados.

Escala

Testes de Integração de Componentes – Glossário IEEE

"Teste em que componentes de **software**, componentes de **hardware** ou ambos são combinados e testados para avaliar a interação entre eles"

Testes de Integração de Sistemas – Glossário ISTQB

Teste em que **sistemas** são integrados a outros sistemas de organizações externas

Testes de Interoperabilidade

Modelo de defeitos de integração entre componentes

- [Pezzè & Young 2008, cap21]

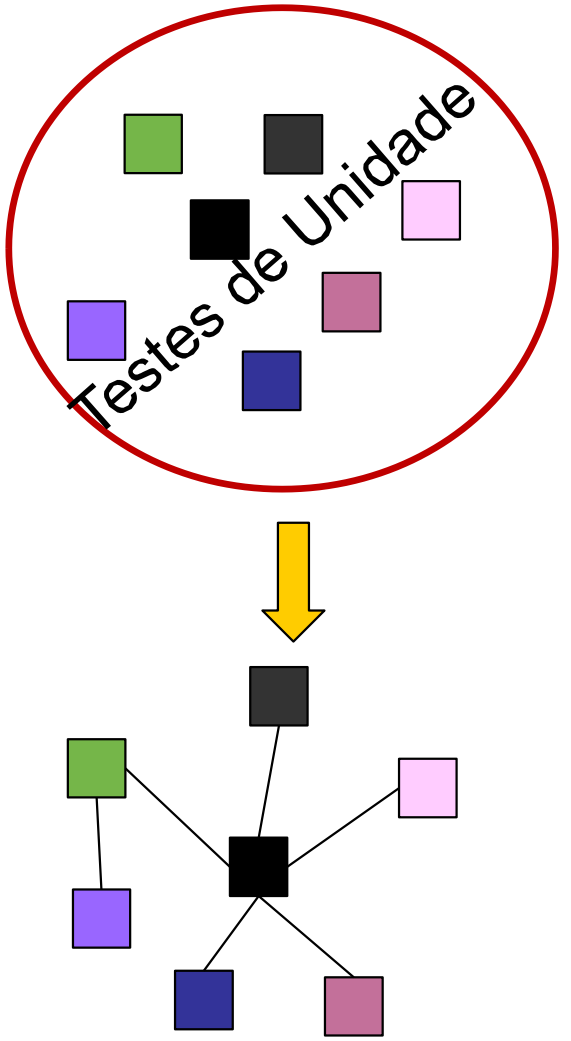
- Defeitos de integração geralmente se devem a especificações incompletas:
  - Interpretação inconsistente de parâmetros e valores:
    - Ex.: Possível causa da perda do satélite Mars Climate: mistura de sistema métrico (metro) com sistema inglês (jardas)
  - Violação dos valores do domínio ou dos limites de capacidade ou tamanho:
    - Buffer overflow: módulo A viola limite de buffer do módulo B, que não verifica violação
  - Efeitos colaterais nos parâmetros ou recursos:
    - Ex.: Módulo A tem arquivo "invisível" (interno) "tmp" ; módulo B solicita criação de outro "tmp" no mesmo diretório

- Incompatibilidades dinâmicas
  - Chamadas polimórficas podem ser levadas dinamicamente a ativar métodos incompatíveis
- Perda ou má compreensão de funcionalidades
  - Módulo A conta nº acessos ao site Web; módulo B, cliente, interpreta que contagem é feita por endereço IP único
- Violação de propriedades não-funcionais
  - Módulo A espera que módulo B atenda uma requisição em 30s, mas B leva 50s para atendê-la

Estratégias de Teste de Integração

Não Incremental

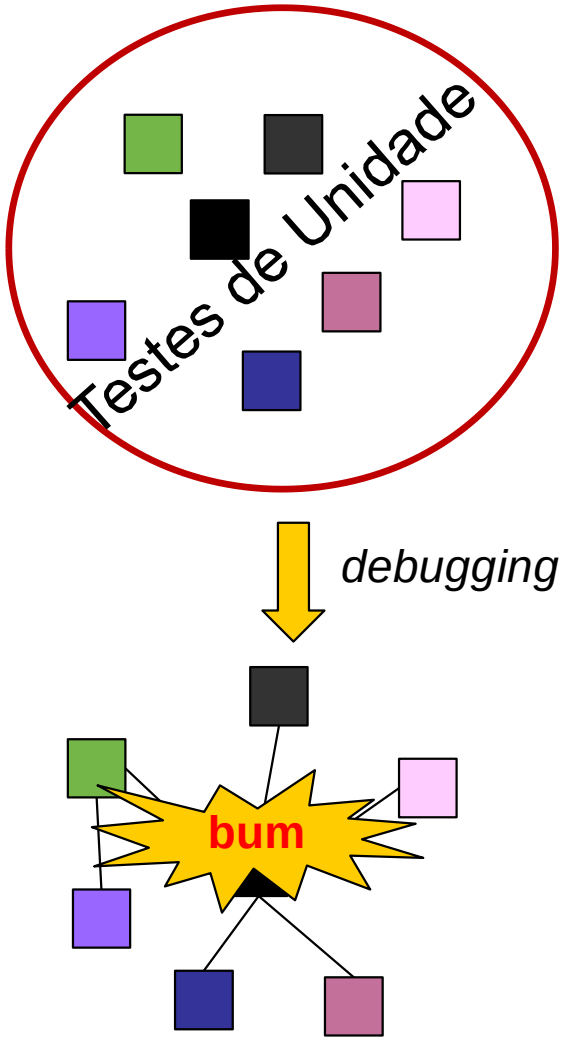
- Também conhecida como *bigbang*
- Todas as Unidades são integradas de uma só vez
- Exigem menor esforço de preparação:
  - Não necessitam de dublês
  - Testador não precisa conhecer nem a arquitetura do sistema nem interfaces entre as Unidades



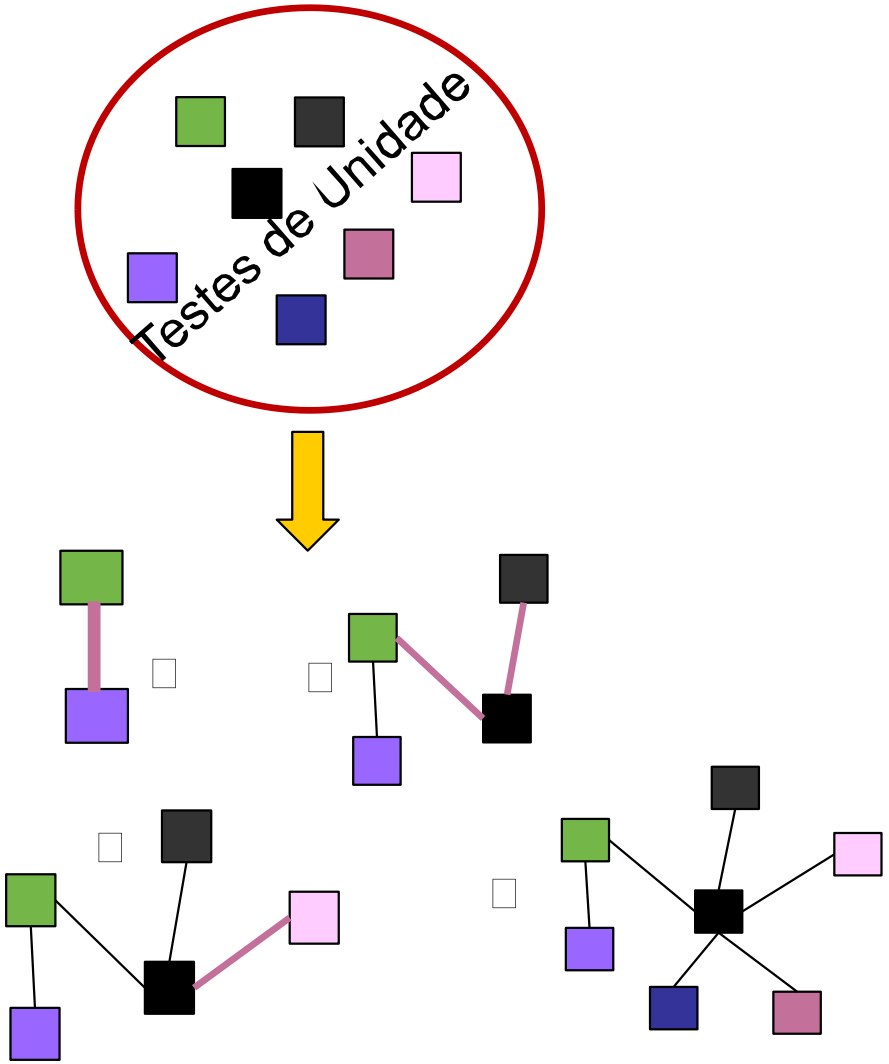
Estratégias de Teste de Integração

Não Incremental

- Também conhecida como *bigbang*
- Todas as Unidades são integradas de uma só vez
- Exigem maior esforço para diagnóstico e correção de defeitos:
  - chances de ocorrência de falhas severas ou catastróficas em Operação



Estratégias de Teste de Integração



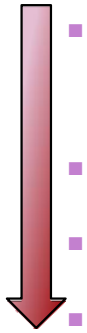
Incremental

- Unidades são integradas aos poucos
  - De preferência, aos pares
- Exigem maior esforço de preparação:
  - Determinar a ordem de integração
  - Criação e manutenção de componentes de apoio

Determinação da ordem de integração

inspirado em [Jin e Offutt 1996]

- Baseia-se nas dependências (acoplamento) entre as Unidades. Dadas duas Unidades A e B:
  - A **usa** B (ou vice-versa), sem passagem de parâmetros e sem compartilhar dados globais ou componentes externos
  - há **passagem de parâmetros** entre A e B
  - A e B **compartilham dados** não-locais ou globais
  - A e B **compartilham** os mesmos **dispositivos externos**



- Diferentes estratégias de integração de acordo com a forma como as dependências são representadas

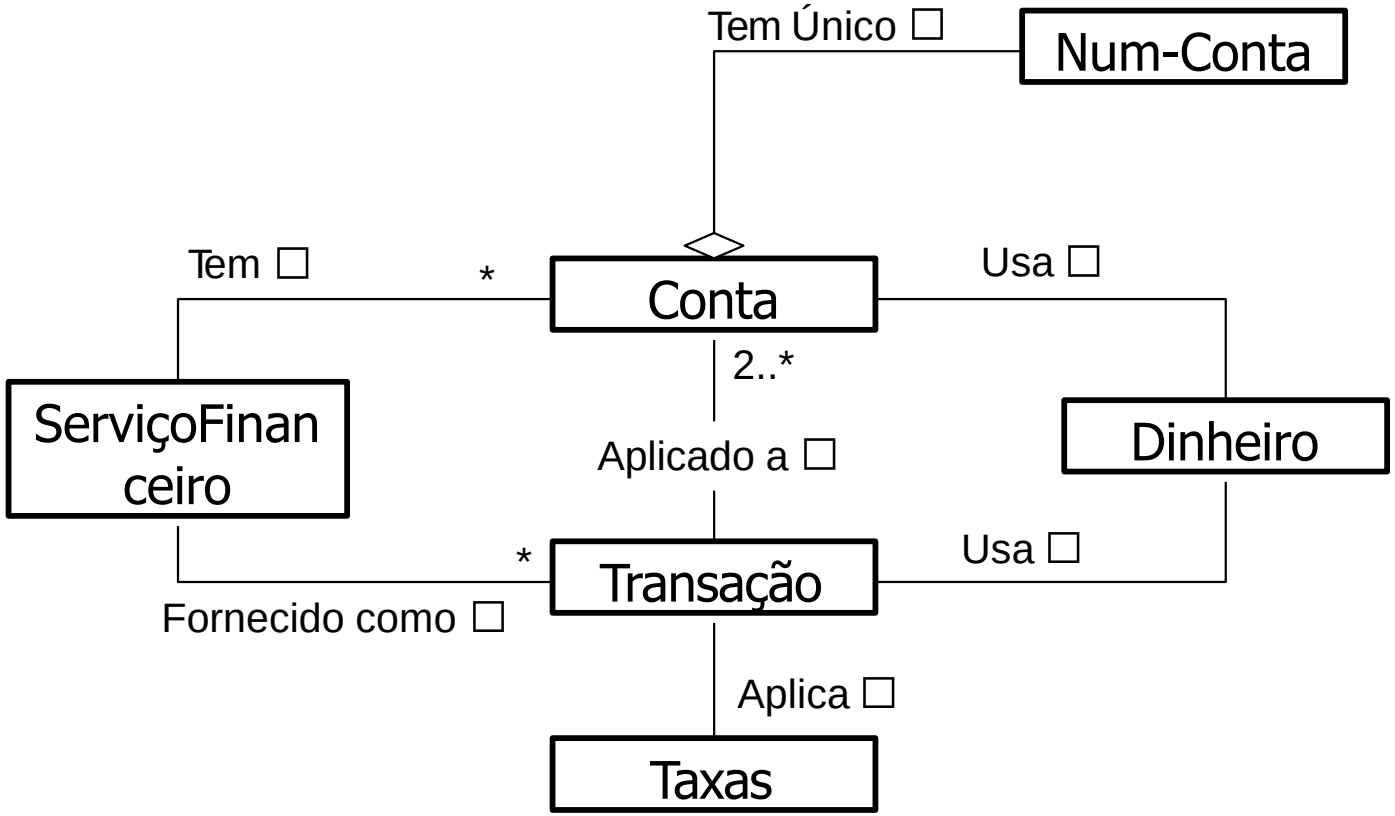
Estratégias incrementais

- **Baseadas na estrutura**
  - *Ordem de integração baseada nas dependências estruturais*
    - visão caixa cinza (arquitetura)
- **Orientadas por funcionalidades**:
  - *Ordem de integração baseada nas dependências para a realização de funcionalidades*
- **Mista**
  - Combinação de diversas abordagens

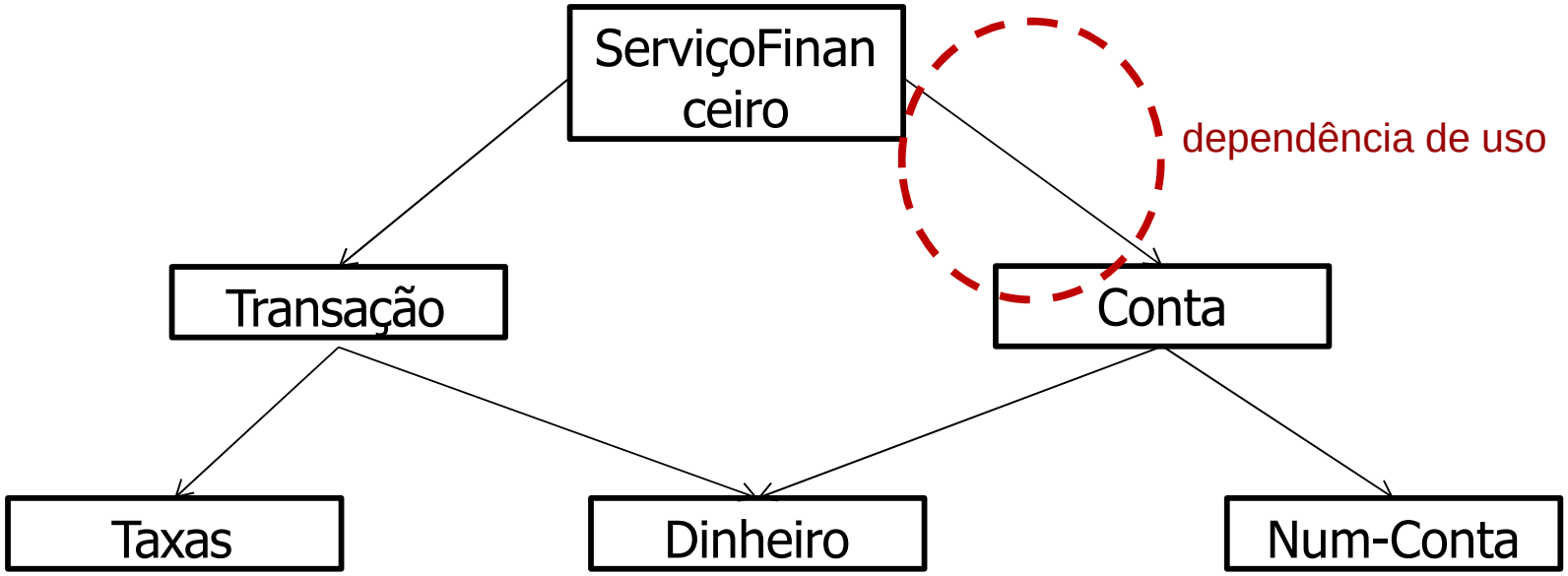
Estratégias baseadas na estrutura

- Modelo de Teste:
  - Árvore de dependências
  - Grafo de dependências (ou de chamadas)
- Ordem de integração:
  - descendente
  - ascendente
  - por camadas
  - mista

Exemplo – Diagrama de Classes do Serviço Financeiro

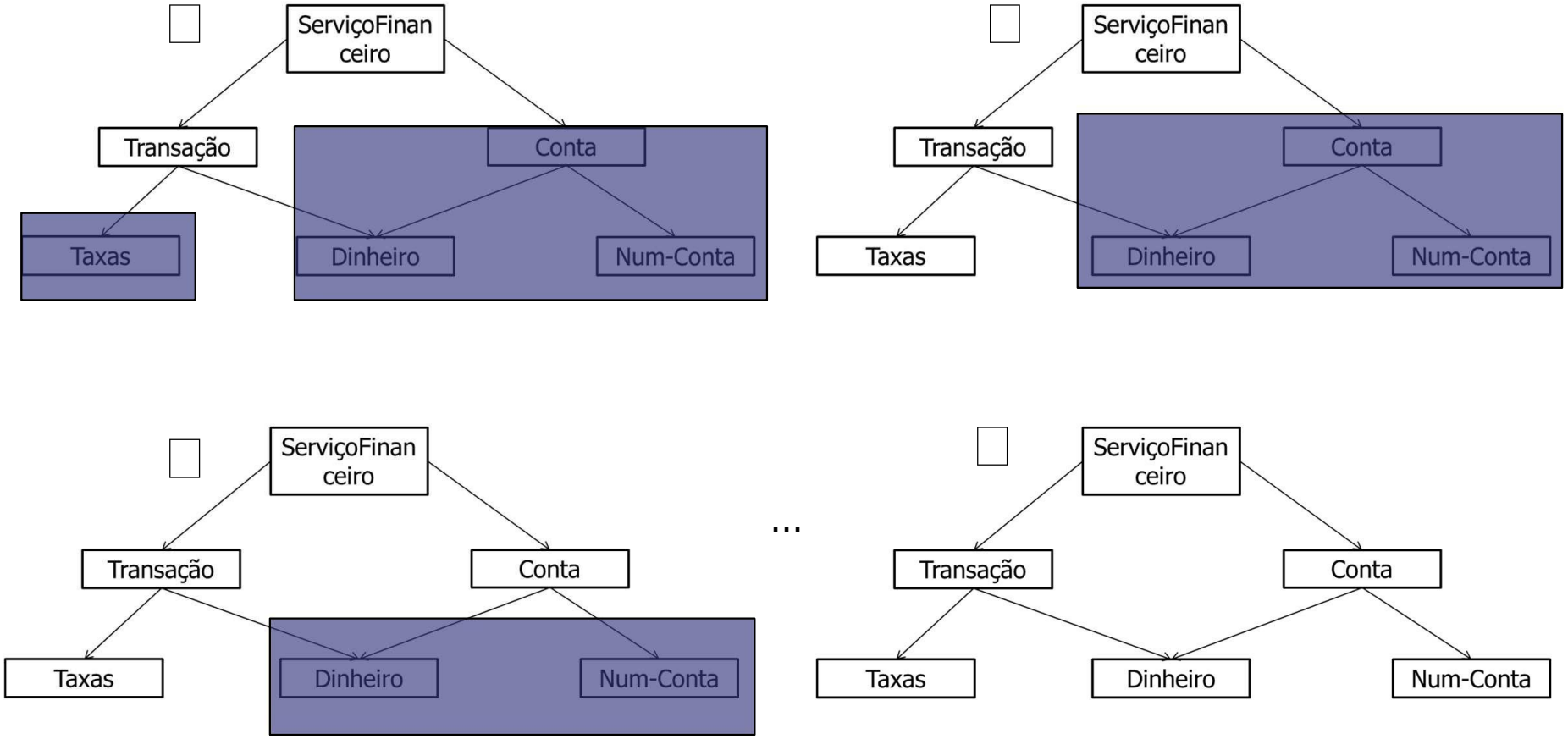


Árvore de Dependência



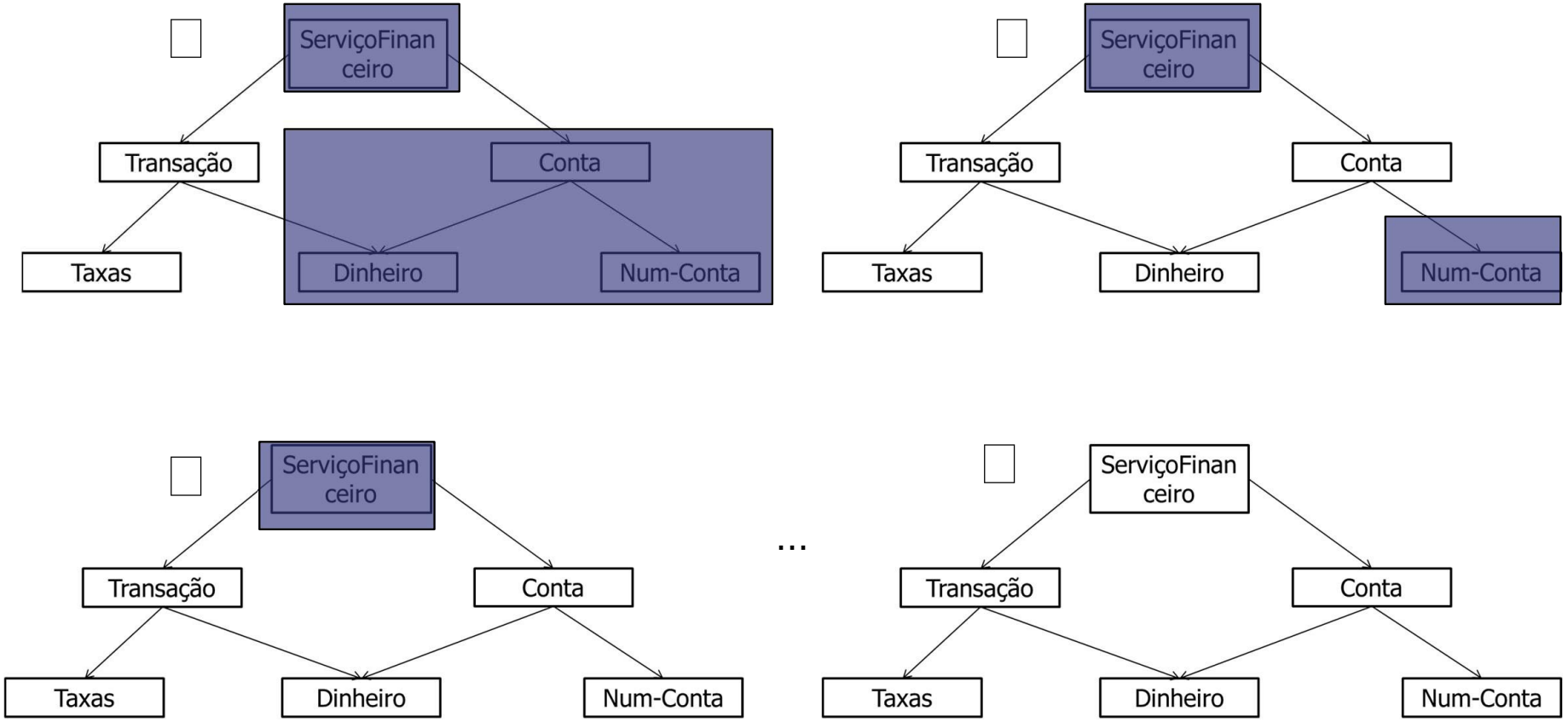
Estratégia descendente (*top-down*)

Uso de dublês



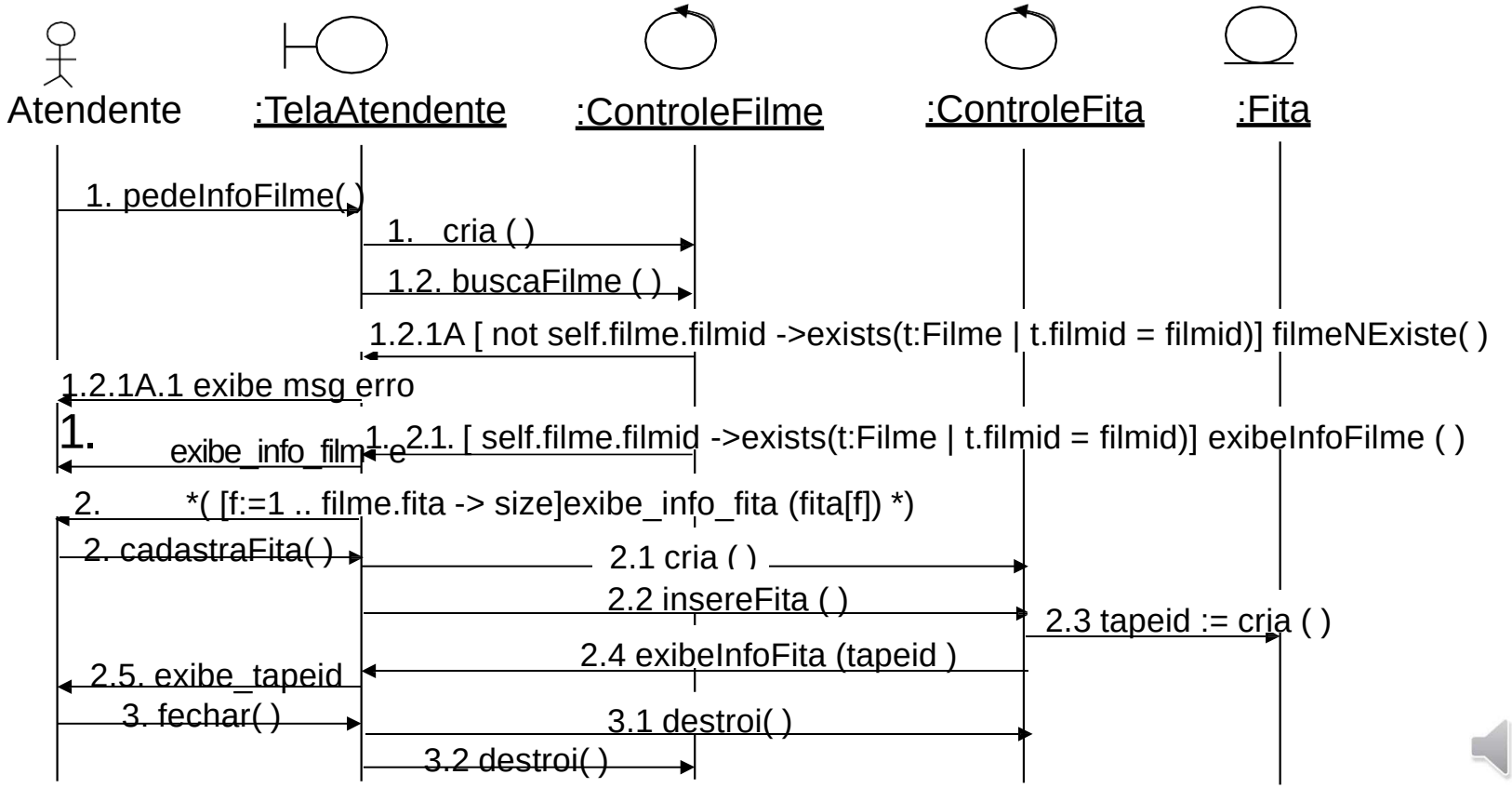
Estratégia ascendente (*bottom-up*)

Uso de drivers



Estratégia funcional

Integração por colaboração: baseada nos diferentes cenários de uso (presentes nos casos de uso)



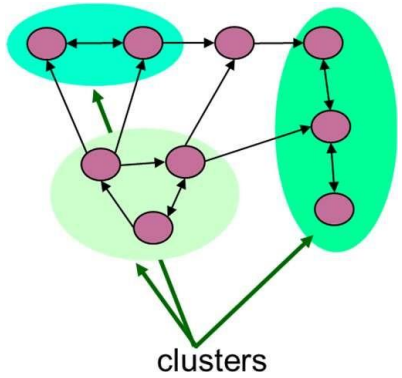
Integração mista

- Combina várias das técnicas anteriores
  - Usar as técnicas mais adequadas de acordo com a parte do sistema que se quer integrar
- Útil quando sistema a ser integrado depende de uma infraestrutura que também está em desenvolvimento
  - Não vale a pena criar dublês para substituir a infra-estrutura de execução
  - Começar os testes integrando as unidades que compõem a infra-estrutura
  - Depois da infra-estrutura ter sido devidamente testada, integrar as demais unidades do sistema a ela, da maneira mais conveniente

Problema na determinação da ordem de integração

- Dependência ⇒ necessidade de dublês
- Objetivo: determinar uma ordem de integração que reduza o número de dublês

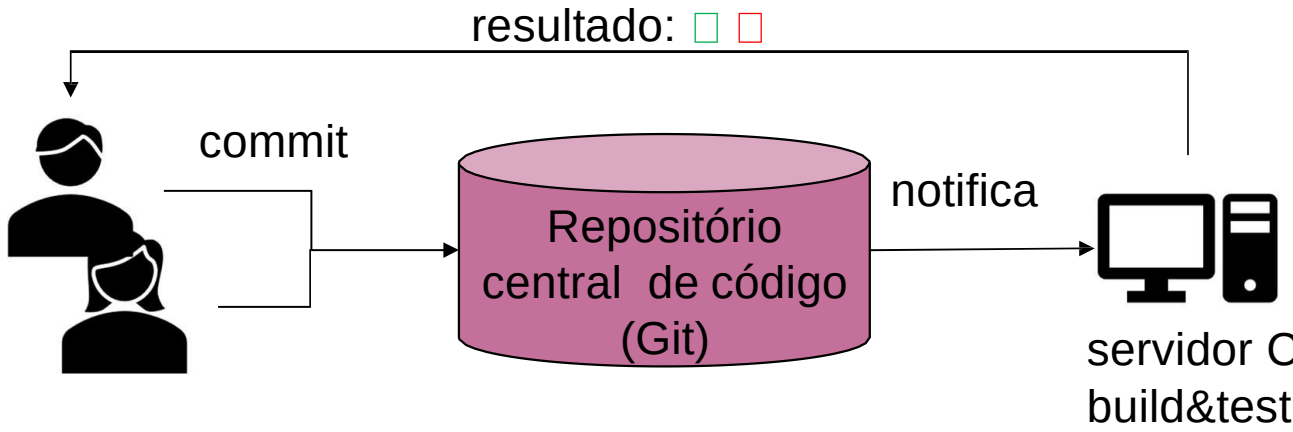
- Problema indecidível, pois pode haver dependências cíclicas acoplamento forte
  - refatorar a arquitetura ou
  - "quebrar" ciclos durante os testes uso de dublês



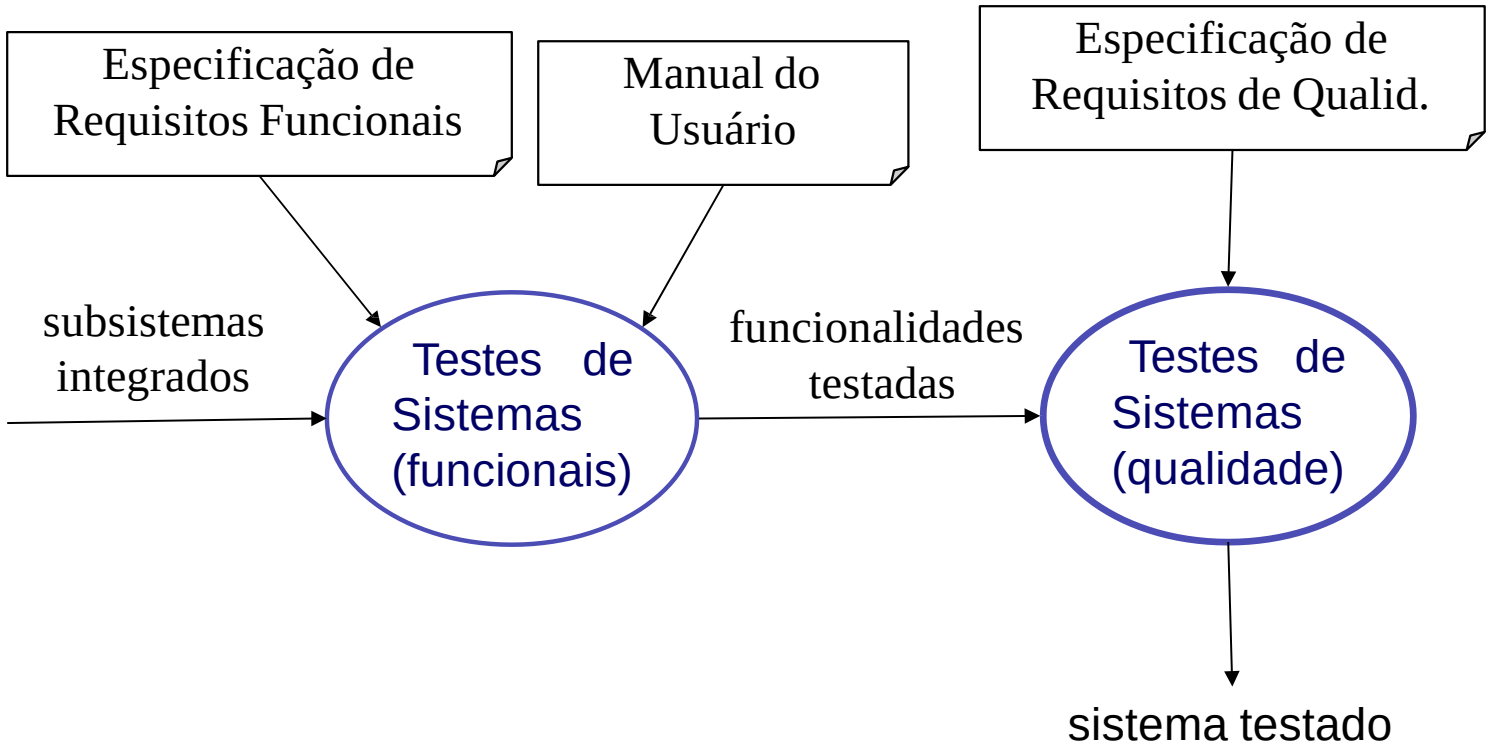
Testes de Integração ≠ Integração Contínua

- Testes de Integração:
  - Sistema é testado por partes, para revelar defeitos na interação entre seus componentes

- Integração Contínua (CI):
  - Visa garantir que um novo incremento não "quebra" uma *build*

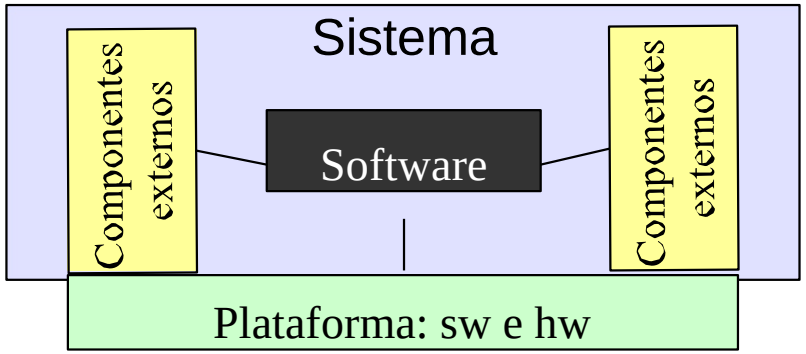


Testes de Sistemas



Objetivo

- Determinar se o sistema completo satisfaz aos requisitos
  - Requisitos funcionais
  - Requisitos não-funcionais

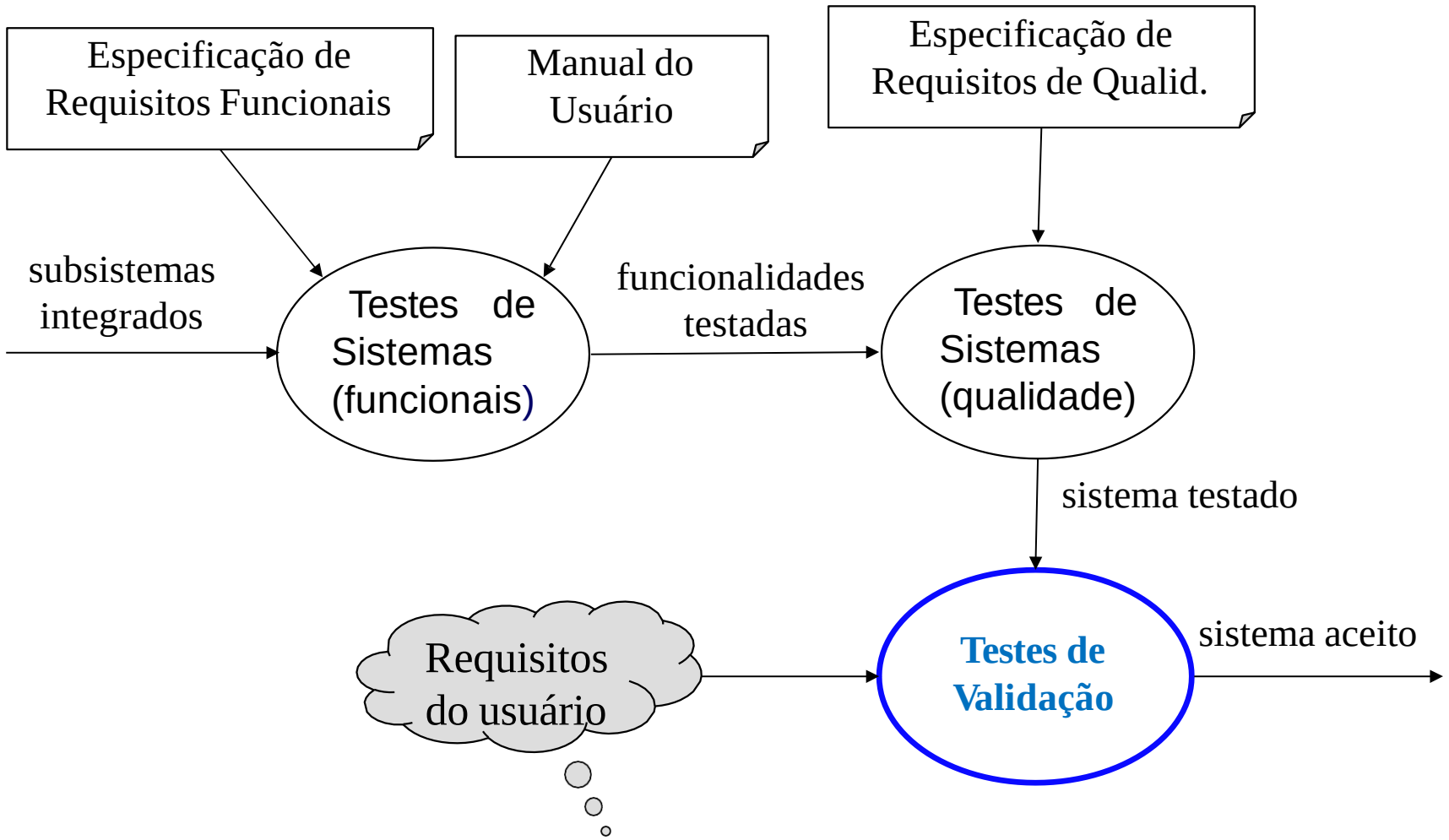


- Por que é necessário?
  - Algumas funções ou características de qualidade só podem ser testados quando o sistema está completo (em-casa + terceiros)
- Pode ser realizado:
  - Em ambiente o mais próximo possível do ambiente de operação
    - □controlabilidade e □observabilidade
  - Em ambiente de execução simulado
    - ex.: sistemas embarcados

Testar se requisitos foram atendidos

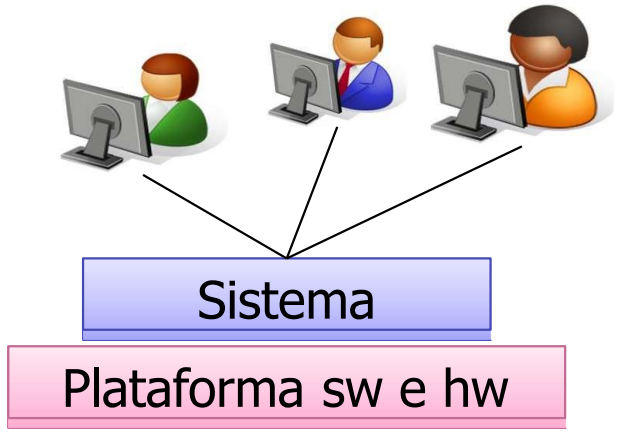
Tipos de testes	Objetivo
Funcionalidade	Determinar se o sistema satisfaz aos requisitos funcionais
Desempenho	Determinar se o tempo de resposta satisfaz ao esperado
Carga	Testar o comportamento do sistema em presença de vários usuários simultâneos
Volume	Verificar se o sistema é capaz de tratar um grande volume de dados
Configuração	Verificar se o sistema funciona adequadamente em diferentes plataformas de hardware e de software
Robustez	Testar o sistema diante de situações inválidas ou inesperadas
Segurança	Testar se o sistema apresenta comportamento de risco em presença de ataques
...	

Testes de Validação



Para que servem

- Validação do sistema:
  - O sistema está pronto para uso?
  - O sistema atende ao que o cliente quer?
  - Quais são os riscos para o negócio (se houver)?
  - O desenvolvimento atendeu às restrições de projeto?
  - ...

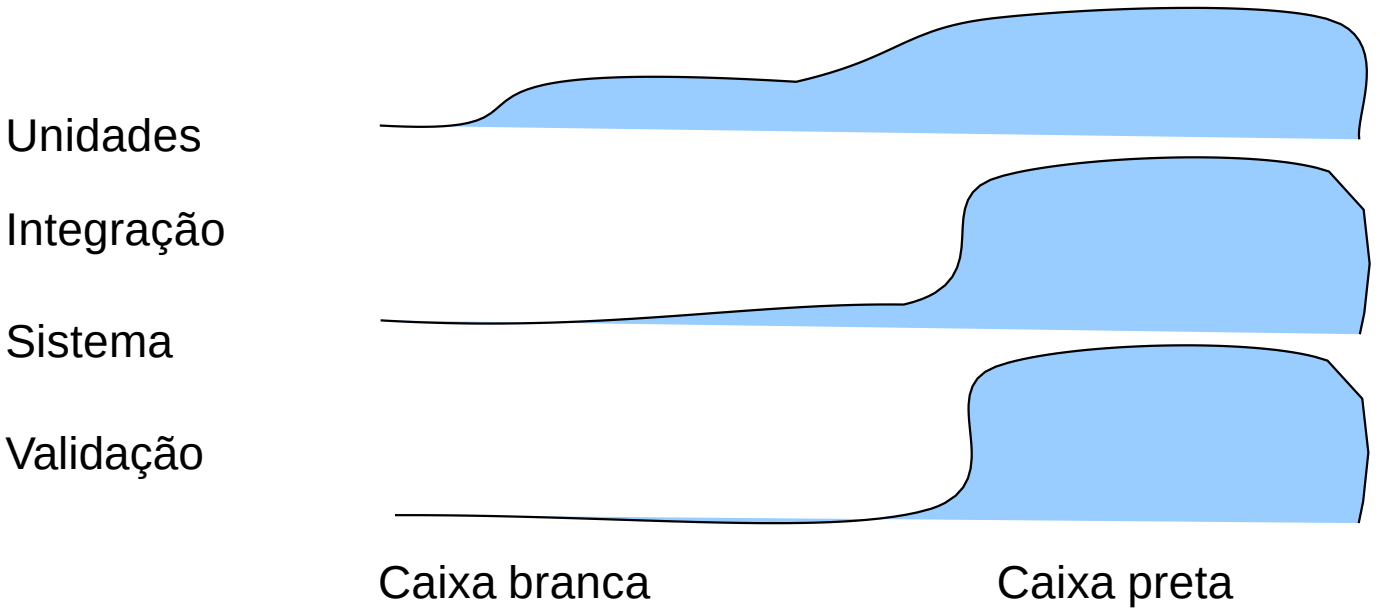


Formas de realização

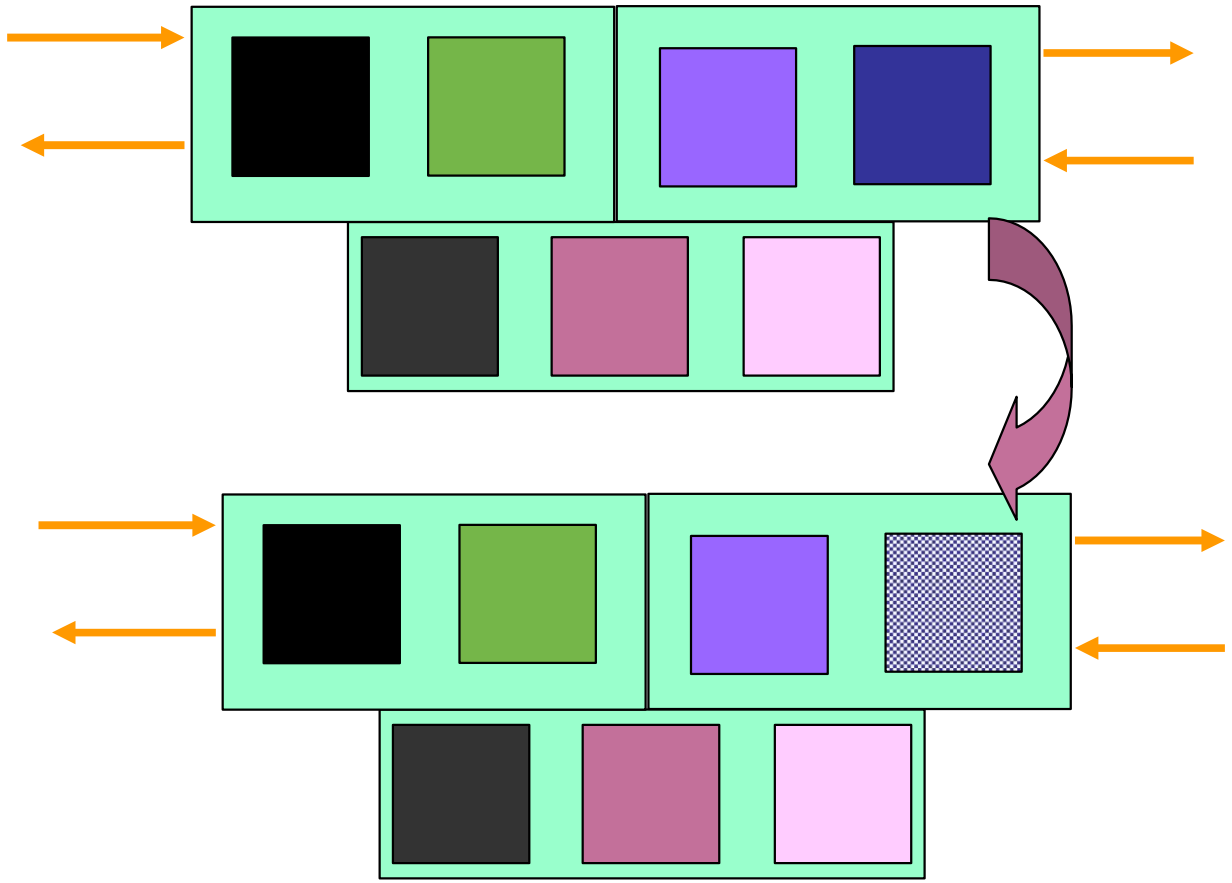
Fonte: [Spillner et al. 2007, cap. 3]

Testes de conformidade	Determinam se o sistema atende aos critérios de aceitação estabelecidos por contrato	
Testes por Grupos de Usuários	Determinam se o sistema satisfaz a diferentes grupos de usuários	
Crowdtesting (crowd sourced software testing)	Diversos grupos de pessoas testam software em ambientes reais usando seus próprios dispositivos. Usados tanto em Testes de Sistemas quanto de Validação, em especial, Testes de Segurança e de Usabilidade.	
Testes Regulatórios	Determinam se o sistema atende a requisitos administrativos ex.: realiza backups periódicos? Satisfaz requisitos de segurança operacional ou de informação?)	
Testes de Aceitação	Determinam se o sistema executa em diferentes ambientes de operação. Testes realizados por usuários selecionados.	**Testes Alfa**: testes realizados em ambiente de desenvolvimento
		Testes Beta: testes realizados em ambiente de produção

Escopo e Técnicas de Teste



Testes de Regressão



Para que serve

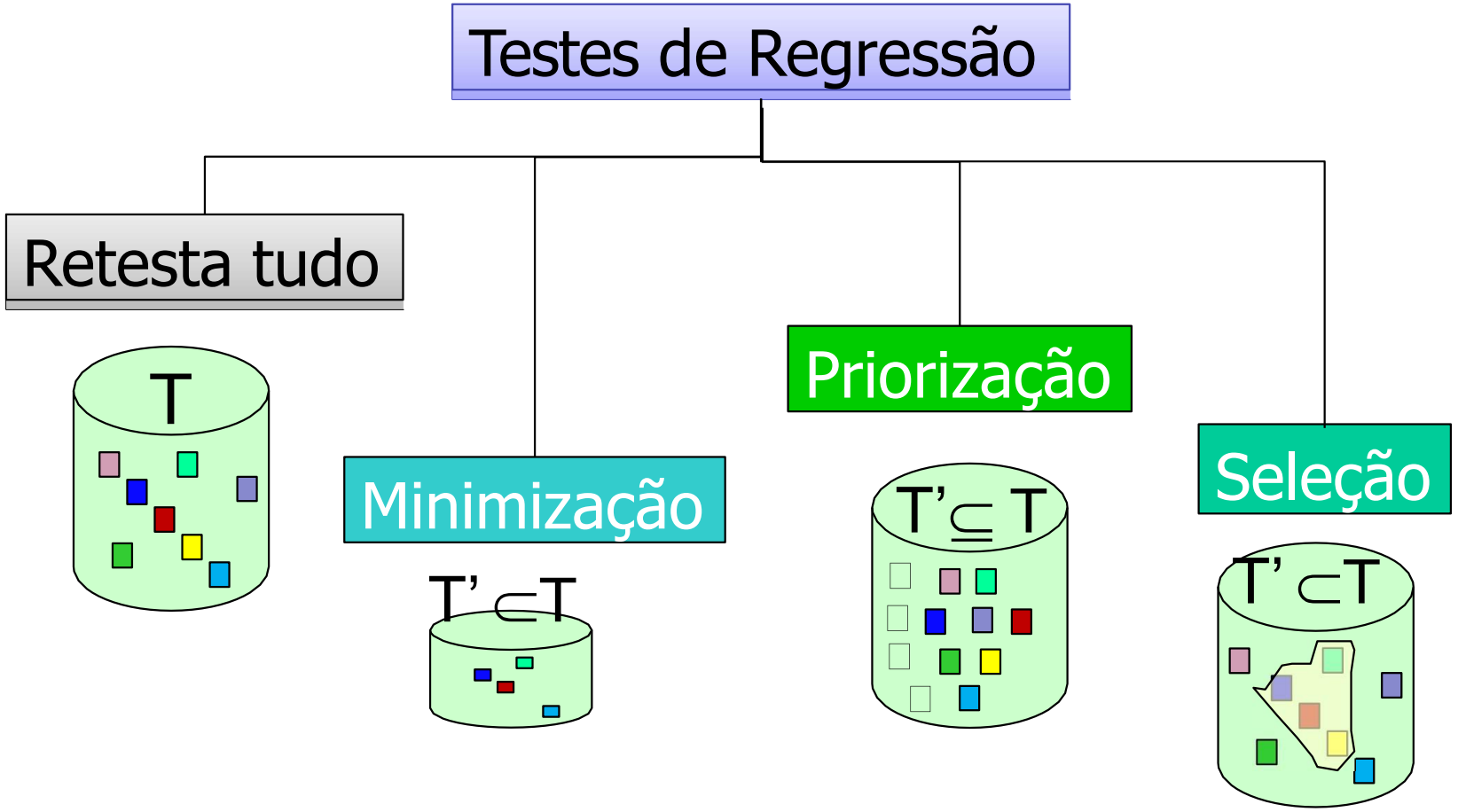
- Testes realizados a cada vez que há alterações
- Objetivo:
  - Mostrar que modificações realizadas não afetaram as partes que não foram modificadas
  - Validar modificações feitas

 Mostrar que o sistema não regrediu

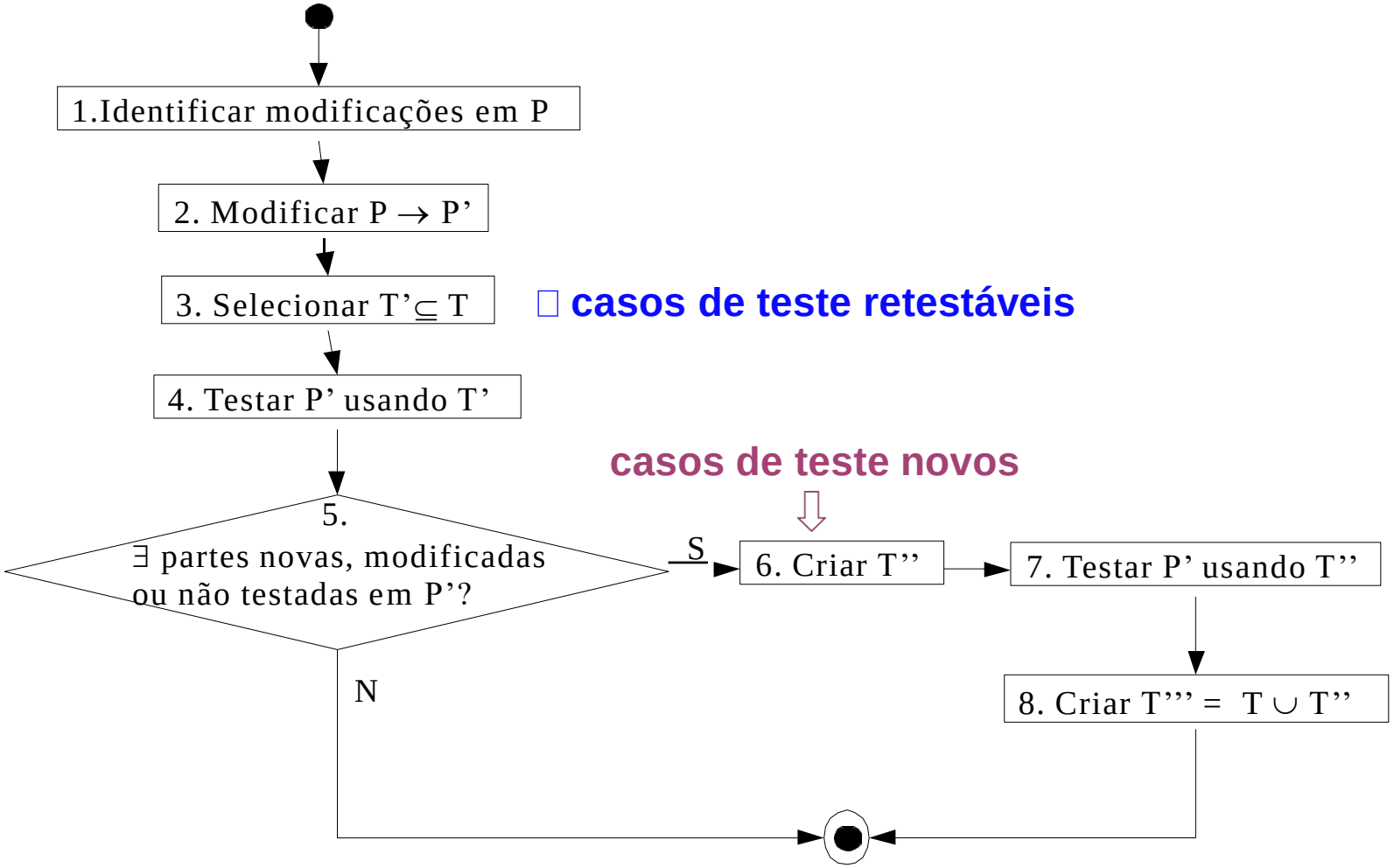
Quando aplicar

- Para testar aplicações críticas que devem ser retestadas freqüentemente
- Para testar sw que é alterado constantemente durante o desenvolvimento (ex.: Processo Incremental ou Evolutivo)
- Para testar componentes reutilizáveis para determinar se são adequados para o novo sistema
- Durante os testes de Integração
- Durante os testes, após correções
- Em fase de manutenção (corretiva, adaptativa, perfectiva ou preventiva)
- Para identificar diferenças no comportamento do sistema quando há mudanças de plataforma (uso de seqüências-padrão ou “benchmarks”)

Estratégias



Seleção: o processo



 casos de teste retestáveis

Classificação dos casos de teste

Casos de teste existen-tes (T)	**Reutilizáveis**	• Executam partes do código que não foram modificadas • Desnecessário reaplicá-los □ Necessário mantê-los
	Retestáveis	• Executam partes do código que foram modificadas □ Necessário reaplicá-los
	Descartáveis	•Testam aspectos da especificação que não mais existem (obsoletos) • Testam partes do código que não mais existem (“quebrados”) • Tornaram-se similares a outros casos de teste (redundantes) •São sensíveis a entradas ou estados que não podem ser controlados pelos testes (incontroláveis) □ NÃO é necessário mantê-los
Casos de teste novos	Estruturais	• Criados para testar partes novas do código
	Especificação	• Criados para testar os novos requisitos

Técnicas seletivas

- A escolha dos casos de teste pode se basear:
  - Na especificação
    - (Re)testam as funcionalidades afetadas pelas alterações:
      - Funcionalidades alteradas ou que contêm módulos alterados
  - Na arquitetura
    - Retestam componentes que realizam funcionalidade alterada ou que contenham módulos alterados
  - No código
    - Retestam módulos que interagem com módulos alterados

Técnicas baseadas na especificação

- A seleção baseia-se na análise do modelo de especificação.
- Úteis nos testes de regressão progressivos
- Técnicas baseadas nos casos de uso [Binder99, c.15] :
  - Requerem menor esforço para seleção
  - Casos de uso de maior risco:
    - retestar casos de uso essenciais para o sistema
  - Casos de uso mais frequentes
    - retestar casos de uso referentes a funcionalidades mais utilizadas
- Outras técnicas (mais custosas):
  - baseadas em cenários
  - baseadas em modelos

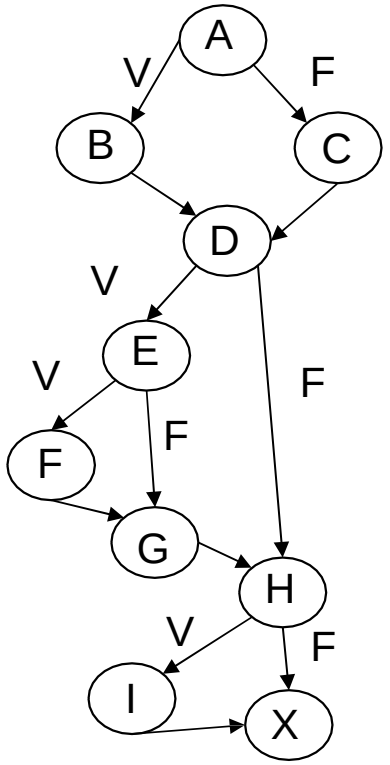
Seleção baseada no código

- Uma técnica:
  - Seleção baseada em segmento modificado
- Algumas técnicas se baseiam na construção do Grafo de Fluxo de Controle (GFC) do programa
  - Passeio síncrono no grafo original e no grafo modificado para identificar as modificações

Exemplo

```
P
if A then B
else C;
if D then
   if E then F;
   G;
if H then I;
X;
```

G

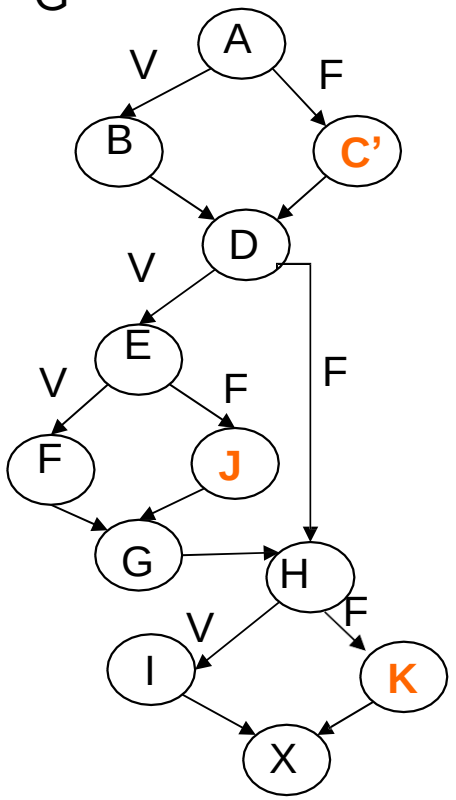


Matriz de cobertura

testes	caminho

```
P'
if A then B
else C';
if D then
   if E then F
   else J;
   G;
 if H then I
   else K;
X;
```

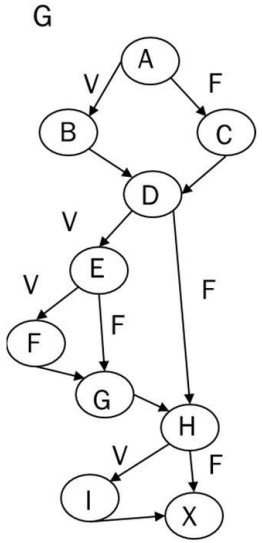
G'

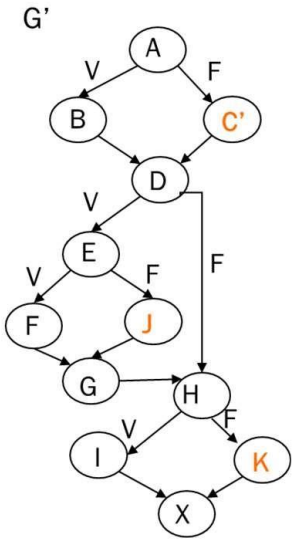


quais casos de teste são retestáveis?

Técnicas – seleção de T’

Percorre G e G’.
Seleciona todos os casos de teste que passam por arestas de G’ com rótulo igual mas nós destinos diferentes





Conjunto T’

Modificação	Seq. segura mínima
C → C’	t4 (**AC**DHX)
+ J	t2 (ABD**EG**HIX)
+K	t1 (ABDEFGHIX)
	t2 (ABDEGHIX)
	t3 (ABDHIX)
	t4 (ACD**HX**)

Em suma

- Escopo de testes determinam a ordem de execução dos testes
  - partes do sistema □ sistema completo
- Aplicável a qualquer processo de desenvolvimento
- Testes de Regressão: aplicável sempre que há mudanças
  - durante os testes
  - após entrega
- Testes de Unidade e Testes de Regressão são os mais frequentes □ automatizá-los é essencial

Referências

- M.Pezzè, M. Young. *TesteeAnálisedeSoftware*. Bookman 2008.
- J. Z. Gao, H.-S. J. Tsao, Y. Wu. *TestingandQuality AssuranceforComponent-BasedSoftware*. Artech House, 2003.
- R.Binder. *TestingOOSystems*. Addison Wesley, 1999, cap13. Jin, Zhenyi, and A. Jefferson Offutt. "Coupling based criteria for integration testing." *Software Testing, Verification and Reliability* 8.3 (1998): 133-154.