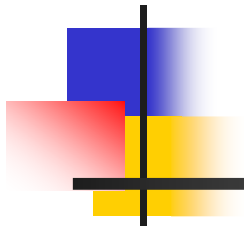


# Teste de Sistemas de Software



Fundamentos de Testes de  
Software

PUC Minas São Gabriel



# Motivação

---

- Ocorrência de falhas humanas no processo de desenvolvimento de software é considerável
- Processo de testes é indispensável na garantia de qualidade de software
- Custos associados às falhas de software justificam um processo de testes cuidadoso e bem planejado



# Conceitos Básicos

---



# Falha, Falta e Erro

---

- Falha

- Incapacidade do software de realizar a função requisitada (aspecto externo)
- Exemplo
  - Terminação anormal, restrição temporal violada



# Falha, Falta e Erro

---

- Falta
  - Causa de uma falha
  - Exemplo
    - Código incorreto ou faltando

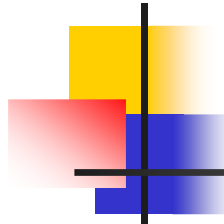


# Falha, Falta e Erro

---

- Erro

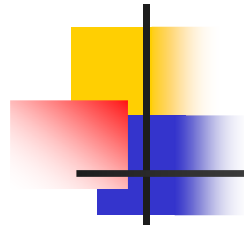
- Estado intermediário (instabilidade)
- Provém de uma falta
- Pode resultar em falha, se propagado até a saída



# Falta, erro e falha

---



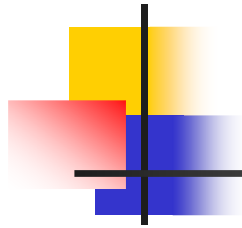


# Noção de confiabilidade

---

- Algumas faltas escaparão inevitavelmente
  - Tanto dos testes
  - Quanto da depuração
- Falta pode ser mais ou menos perturbadora
  - Dependendo do que se trate e em qual frequência irá surgir para o usuário final





# Noção de confiabilidade

---

- Assim, precisamos de uma referência para decidir
  - Quando liberar ou não um sistema para uso
- Confiabilidade de software
  - É uma estimativa probabilística
  - Mede a frequência com que um software irá executar sem falha
    - Em dado ambiente
    - E por determinado período de tempo



# Noção de confiabilidade

---

- Assim, entradas para testes devem se aproximar do ambiente do usuário final



# Dados e Casos de Teste

---

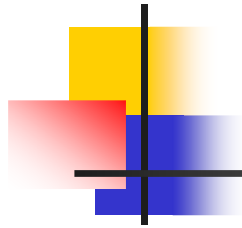
- Dados de Teste
  - Entradas selecionadas para testar o software
- Casos de Teste
  - Dados de teste, bem como saídas esperadas de acordo com a especificação (Veredicto)
  - Cenários específicos de execução



# Finalidade dos testes

---

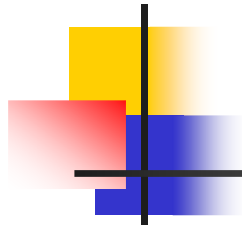
- Averiguar se todos os requisitos do sistema foram corretamente implementados
- Assegurar, na medida do possível, a qualidade e a corretude do software produzido
- Reduzir custos de manutenção corretiva e re-trabalho



# Finalidade dos testes

---

- Assegurar a satisfação do cliente com o produto desenvolvido
- Identificar casos de teste de elevada probabilidade para revelar erros ainda não descobertos (com quantidade mínima de tempo e esforço)
- Verificar a correta integração entre todos os componentes de software



# Eficácia de Testes

---

- A atividade de teste é o processo de executar um programa com a intenção de descobrir um erro
- Um bom caso de teste é aquele que apresenta uma elevada probabilidade de revelar um erro ainda não descoberto
- Um teste bem sucedido é aquele que revela um erro ainda não descoberto



# Padronização de Testes

---

- Sistemático
  - Testes aleatórios não são suficientes
  - Testes devem cobrir todos os fluxos possíveis do software
  - Testes devem representar situações de uso reais
- Documentado
  - Que testes foram feitos, resultados, etc.
- Repetível
  - Se encontrou ou não erro em determinada situação, deve-se poder repeti-lo



# Abordagens de teste

---

- Abordagem funcional ("caixa preta")
  - Os testes são gerados a partir de uma análise dos relacionamentos entre os dados de entrada e saída, com base nos requisitos levantados com os usuários
    - Especificação (pré e pós-condições)
  - Geralmente é aplicado durante as últimas etapas do processo de teste





# Abordagens de teste

---

- Abordagem funcional ("caixa preta")
  - Objetivo
    - Erros associados a não satisfação da especificação
    - Erros na GUI
    - Erros nas estruturas de dados ou acesso ao banco de dados
    - Problemas de integração



# Abordagens de teste

---

- Abordagem estrutural ("caixa branca")
  - Os testes são gerados a partir de uma análise dos caminhos lógicos possíveis de serem executados
  - Conhecimento do funcionamento interno dos componentes do software é usado



# Abordagens de teste

---

- Abordagem estrutural
  - Objetivo
    - Garantir que todos os caminhos independentes dentro de um módulo tenham sido exercitados pelo menos uma vez
    - Realizar todas as decisões lógicas para valores falsos e verdadeiros
    - Executar laços dentro dos valores limites
    - Executar as estruturas de dados internas



# Abordagens de teste

---

- Abordagem estrutural
  - Programador
    - Testa o programa em pedaços
    - Encontra quais as partes do programa já foram testadas
    - Conhece quais partes do programa serão modificadas
    - Verifica os limites internos no código que são invisíveis ao testador externo

**“É parte da atividade de codificação”**



# Estágios de Teste

---

- Teste de Unidade
- Teste de Aspectos OO
- Teste de Integração
- Teste de Sistema
- Teste de Aceitação



# Estágios de teste

---

- Teste de unidade
  - Componentes individuais (ex.: métodos, classes) são testados para assegurar que os mesmos operam de forma correta
- Teste de aspectos OO
  - Teste de Iteradores
  - Teste de Abstrações de Dados
  - Teste de Hierarquia de Tipos



# Estágios de teste

---

- Teste de integração
  - A interface entre as unidades integradas é testada
- Teste de sistema
  - Os elementos de software integrados com o ambiente operacional (hardware, pessoas, etc.) são testados como um todo



# Estágios de teste

---

- Testes de aceitação (“caixa preta”) são realizados pelo usuário
  - Finalidade é demonstrar a conformidade com os requisitos do software
- Envolve treinamento, documentação e empacotamento
- Podem ser de duas categorias:
  - Testes *alfa*
    - Feitos pelo usuário, geralmente nas instalações do desenvolvedor, que observa e registra erros e/ou problemas

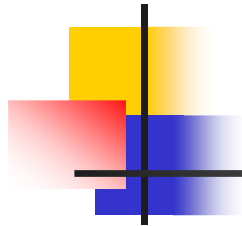




# Estágios de teste

---

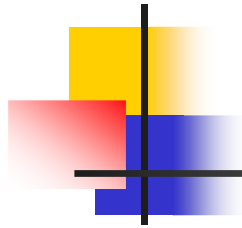
- Testes de aceitação (“caixa preta”) são realizados pelo usuário
  - Testes *beta*
    - Feitos pelo usuário, geralmente em suas próprias instalações, sem a supervisão do desenvolvedor. Os problemas detectados são então relatados para o desenvolvedor



# Tipos de Teste

---

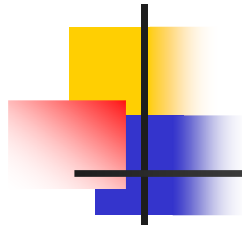
- São definidos em relação aos diversos tipos de requisitos descritos no documento de requisitos
- Alguns exemplos são:
  - Teste funcional
  - Teste de recuperação de falhas
  - Teste de segurança
  - Teste de performance
  - Teste de carga



# Tipos de teste

---

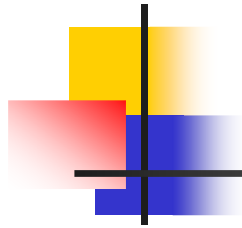
- Teste de funcionalidade (regras de negócio)
  - A funcionalidade geral do sistema em termos de regras de negócio (fluxo de trabalho) é testada
  - Condições válidas e inválidas



# Tipos de teste

---

- Teste de recuperação de falhas
  - O software é forçado a falhar de diversas maneiras para que seja verificado o seu comportamento
  - Bem como a adequação dos procedimentos de recuperação
    - A recuperação pode ser automática ou exigir intervenção humana



# Tipos de teste

---

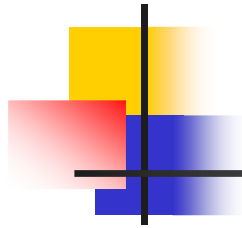
- Teste de segurança e controle de acesso
  - Verifica se todos os mecanismos de proteção de acesso estão funcionando satisfatoriamente
- Teste de integridade de dados
  - Verifica a corretude dos métodos de acesso à base de dados e a garantia das informações armazenadas



# Tipos de teste

---

- Teste de performance
  - Verifica tempo de resposta e processamento (para diferentes configurações, número de usuários, tamanho do BD, etc.)
  - Exemplo
    - Recuperar conta de usuário em x segundos
  - São necessários definir
    - Servidores e clientes, sistemas operacionais e protocolos utilizados



# Tipos de teste

---

- Teste de volume (carga)
  - Foca em transações do BD
  - Verifica se o sistema suporta altos volumes de dados em uma única transação
  - Verifica o número de terminais, modems e bytes de memória que é possível gerenciar



# Tipos de teste

---

- Teste de estresse
  - Verifica a funcionalidade do sistema em situações limite
    - Pouca memória ou área em disco, alta competição por recursos compartilhados (ex: vários acessos/transações no BD ou rede)
    - Exemplo: *pode-se desejar saber se um sistema de transações bancárias suporta uma carga de mais de 100 transações por segundo ou se um sistema operacional pode manipular mais de 200 terminais remotos*





# Tipos de teste

---

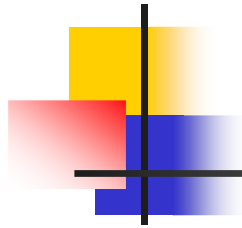
- Teste de configuração ou portabilidade
  - Verifica o funcionamento adequado do sistema em diferentes configurações de hardware/software
  - O que testar
    - Compatibilidade do software/hardware
    - Configuração do servidor
    - Tipos de conexões com a Internet
    - Compatibilidade com o browser



# Tipos de teste

---

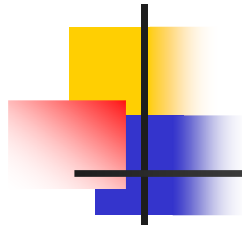
- Teste de instalação e desinstalação
  - Verifica a correta instalação e desinstalação do sistema para diferentes plataformas de hardware/software e opções de instalação
  - O que testar
    - Compatibilidade do hardware e software
    - Funcionalidade do instalador/desinstalador sob múltiplas opções/condições de instalação
    - GUI do programa instalador/desinstalador



# Tipos de teste

---

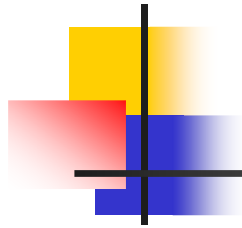
- Teste da GUI (usuário)
  - Aparência e comportamento da GUI
  - Navegação
  - Consistência
  - Aderência a padrões
  - Tempo de aprendizagem
  - Funcionalidade



# Tipos de teste

---

- Teste de documentação
  - Verifica se a documentação corresponde à informação correta e apropriada:
    - online
    - escrita
    - help sensível ao contexto
- Teste de ciclo de negócios
  - Garante que o sistema funciona adequadamente durante um ciclo de atividades relativas ao negócio



# Teste de regressão

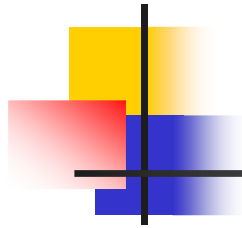
---

- Re-execução de testes feitos após uma manutenção corretiva ou evolutiva
- Em processos de desenvolvimento iterativos, muitos dos artefatos produzidos nas primeiras iterações são usados em iterações posteriores



# Teste de unidade

---

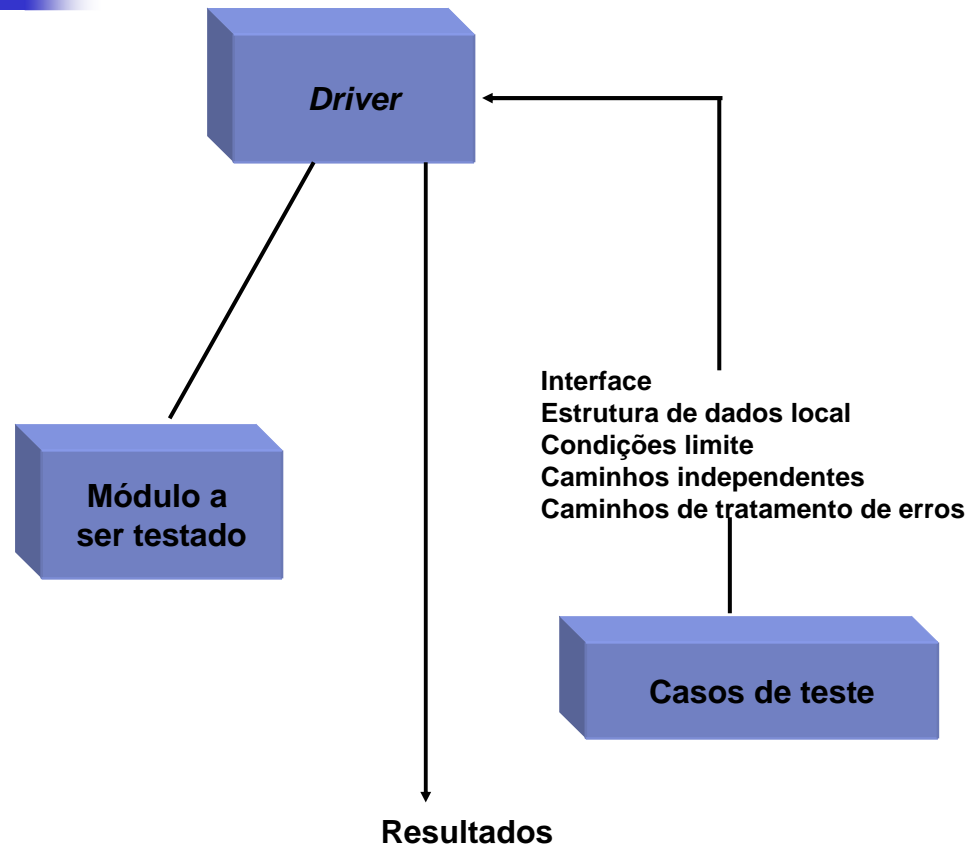


# Teste de unidade

---

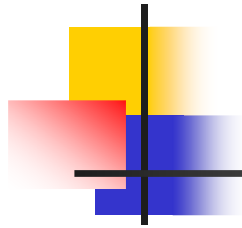
- Investigar a qualidade de componentes individuais (ex: métodos, classes)
- Objetivo:
  - Testar comportamento (especificação) e estrutura interna (lógica e fluxo de dados)

# Estratégia para Teste de unidade



- *Driver de teste* - programa que executa o módulo a ser testado usando os dados do caso de teste e verifica o veredicto
  - Este será o propósito de uso do JUnit

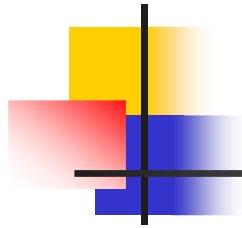




# Teste de Caixa Preta

---

- Casos de teste são gerados usando somente a especificação
- Vantagens:
  - Procedimento de teste não é influenciado pela implementação
  - Resultados dos testes podem ser avaliados por pessoas sem conhecimento da linguagem de programação
  - Robusto em relação a mudanças na implementação (Abordagem XP...)



# Teste de Caixa Preta

---

- Deve-se analisar a relação entre a pré e a pós-condição
- Tentar cobrir todas as combinações lógicas existentes entre essas partes
- Dada a relação  $\text{pré} \Rightarrow \text{pós}$ , tem-se
  - $\text{pré}=\text{true} \Rightarrow \text{pós}=\text{true}$
  - $\text{pré}=\text{false} \Rightarrow \text{exceção}$

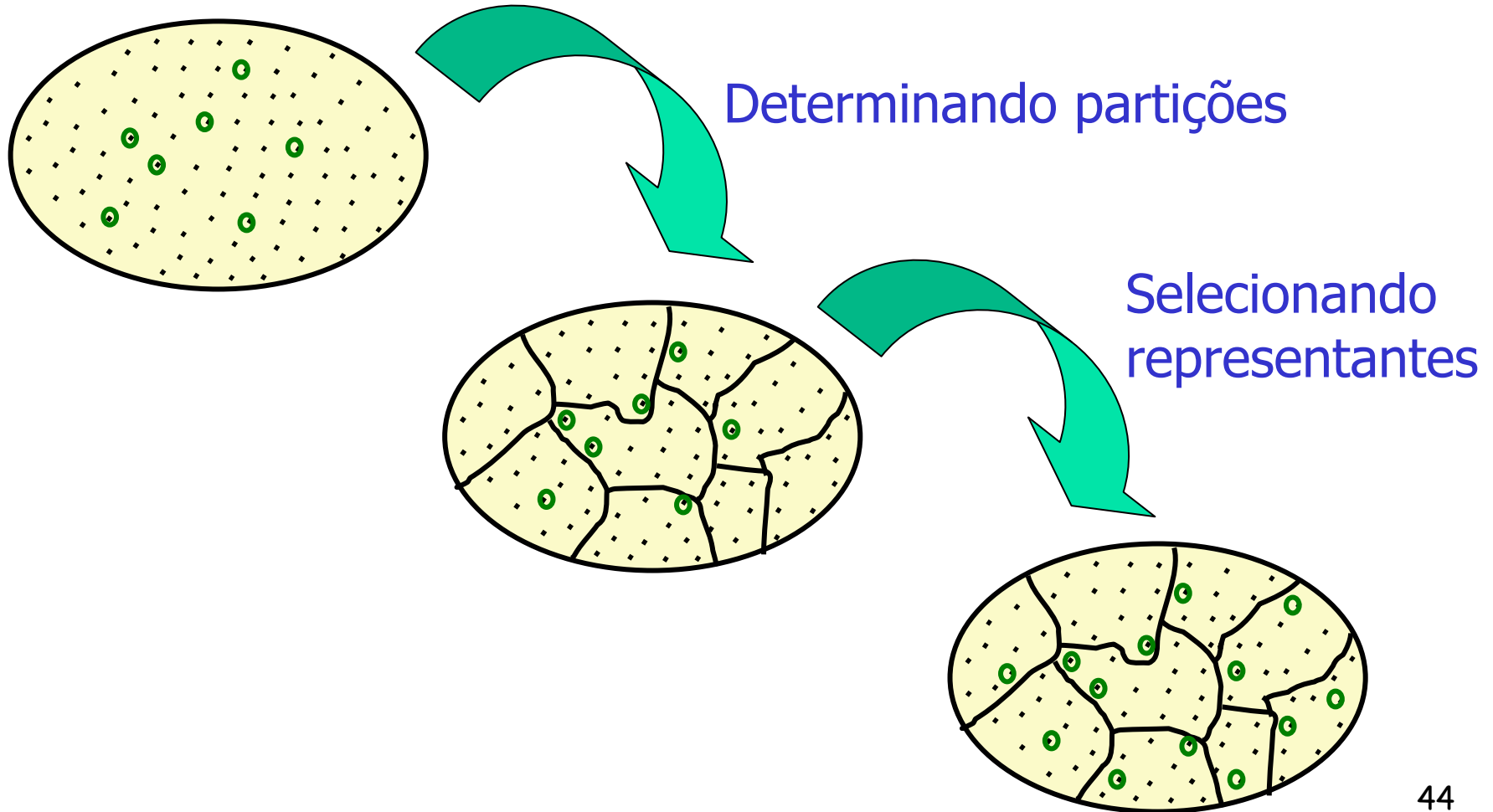


# Seleção de Dados de Teste

---

- Há várias técnicas para seleção de dados de teste
  - Particionamento
  - Fronteiras
  - Pares ortogonais
  - Etc.

# Particionamento





# Fronteiras

---

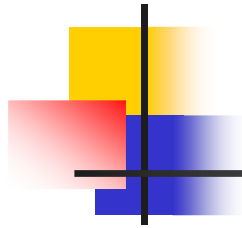
- Estatísticas indicam que há uma maior suscetibilidade a erros nas fronteiras de partições (limites dos tipos)
- Tanto em dados válidos quanto inválidos
  - Assim, para  $x > 0$ , não bastaria usar qualquer  $x > 0$  (particionamento)
  - Mas sim  $x = 1$  (válido no limite) e  $x = 0$  (inválido no limite)



# Fronteiras

---

- A técnica da seleção de dados pelas fronteiras é muito indicada para investigar bom funcionamento de
  - Arrays
  - Vetores
  - Algoritmos de busca/ordenação
  - Etc.



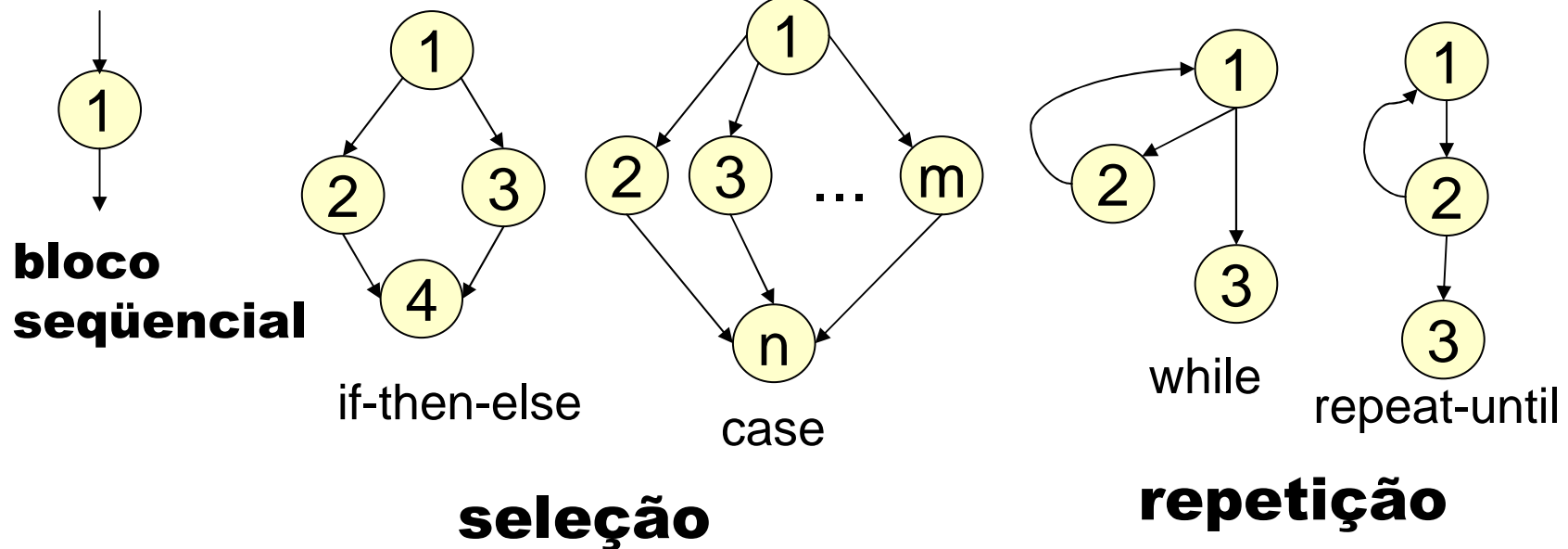
# Teste de Caixa Branca

---

- Casos de teste são gerados a partir da implementação
- Não se pode avaliar o grau de cobertura de uma funcionalidade pelo teste de caixa preta
- A idéia é gerar dados de teste que permitam exercitar algum critério em relação ao código (cobertura)

# Grafo de fluxo de controle

nó = bloco de comandos seqüenciais  
aresta ou ramo = transferência de controle

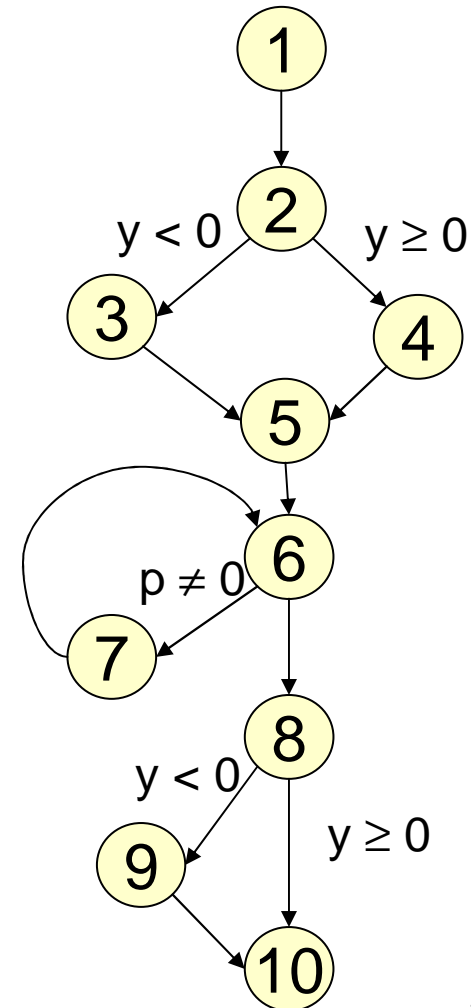




# Grafo de fluxo de controle: Exemplo

## Cálculo de $xy$

```
1. read x, y;  
2. if y < 0  
3.   then p := 0 - y  
4.   else p := y;  
5. z := 1.0;  
6. while p ≠ 0 do  
7.   begin  
8.     z := z * x; p := p - 1;  
9.   end;  
10. write z;  
    end;
```



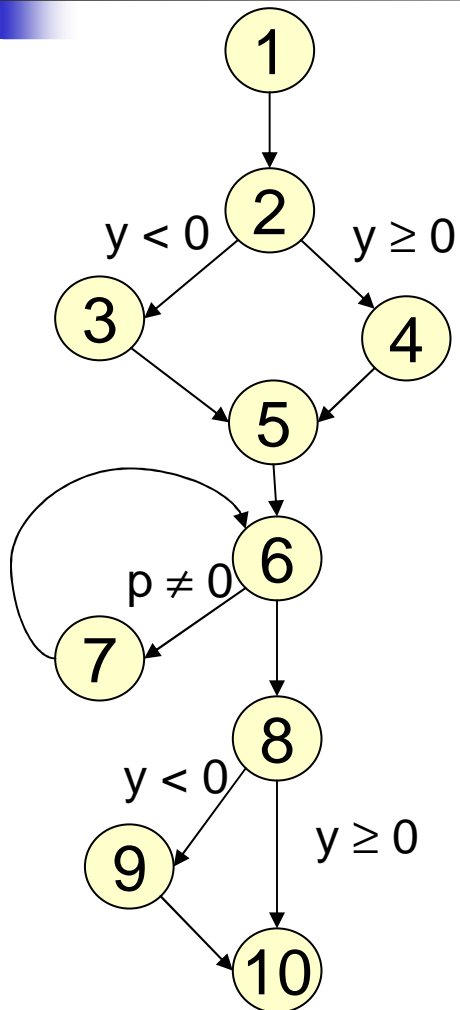


# Critérios de cobertura

---

- Tipos
  - Cobertura de instruções
  - Cobertura de decisões
  - Cobertura de condições
  - Cobertura de caminhos

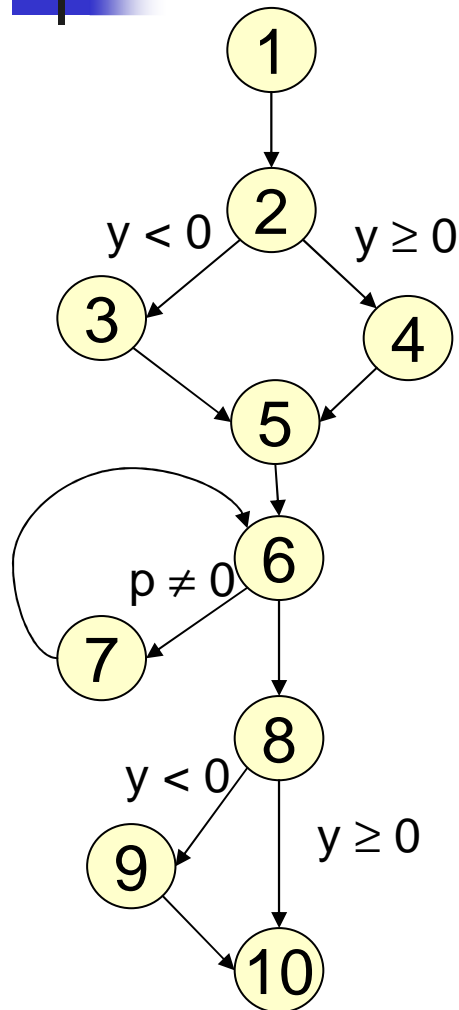
# Cobertura de instruções



**Critério: cada instrução deve ser executada pelo menos 1 vez**

nós	predicados	dados
{1,2,3,5,6,7,6, 8,9,10}	$\forall x, y < 0$	(4, -1)
{1,2,4,5,6,8,10}	$\forall x, y = 0$	(4, 0)

# Cobertura de decisões

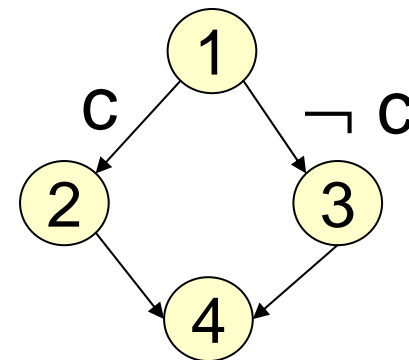


**Critério: cada ramo deve ser percorrido pelo menos 1 vez**

ramos	predicados	dados
{(1,2), (2,3), (3,5), (5,6), (6,7), (7,6), (6,8), (8,9), (9,10)}	$\forall x, y < 0$	(4, -1)
{(1,2), (2,4), (4,5), (5,6), (6,7), (7,6), (6,8), (8, 10)}	$\forall x, y = 0$	(4, 0)

# Cobertura de decisões

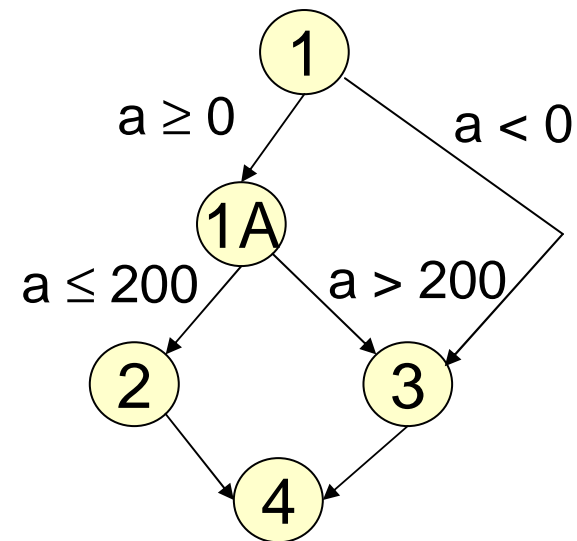
1. if  $\overbrace{a \geq 0 \text{ and } a \leq 200}^C$   
2. then  $m := 1$   
3. else  $m := 3$   
4. "Dep. em a..."



ramos	dados
$\{(1,2), (2,4)\}$	$a = 5$
$\{(1,3), (3,4)\}$	$a = -5$

# Cobertura de decisões/condições

1. if  $a \geq 0$  and  $a \leq 200$   
2. then  $m := 1$   
3. else  $m := 3$   
4. "Dep. em a..."



ramos	dados
$\{(1,1A), (1A, 2), (2,4)\}$	$a = 5$
$\{(1,1A), (1A,3), (3,4)\}$	$a = 500$
$\{(1,3), (3,4)\}$	$a = -5$

**Critério: todas as condições devem ser avaliadas para valores V/F em cada (sub)decisão**



# Teste de Abstrações de Dados

---

- Testar os métodos de maneira conjugada
- Testes devem explorar propriedades esperadas para a composição de métodos
- Sequências de testes devem ser originadas tanto pelo teste de caixa preta quanto o de caixa branca

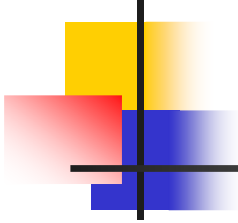


# Exemplo

---

- Suponha uma classe com métodos inserir, remover, consultar, etc.
- Então, seqüências como:
  - consultar(x) -> inserir(x) -> remover(x)
  - inserir(x) -> inserir(x)
  - inserir(x) -> consultar(x)
  - remover(x) -> consultar(x)





# Automação de Testes Integrada

---

- Testar é essencial como referencial de qualidade
- Falhas humanas são comuns
- Realizar todos os testes, associados a artefatos modificados, é imprescindível



# Bibliografia

---

- Liskov, B. et al. Program Development in Java (Cap. 10)
- Sommerville, I. Software Engineering (Cap. 20)