

Cartão de Suporte: Teste de Software

Prof. **Lesandro Ponciano** - lesandrop@pucminas.br
Departamento de Engenharia de Software e Sistemas de Informação

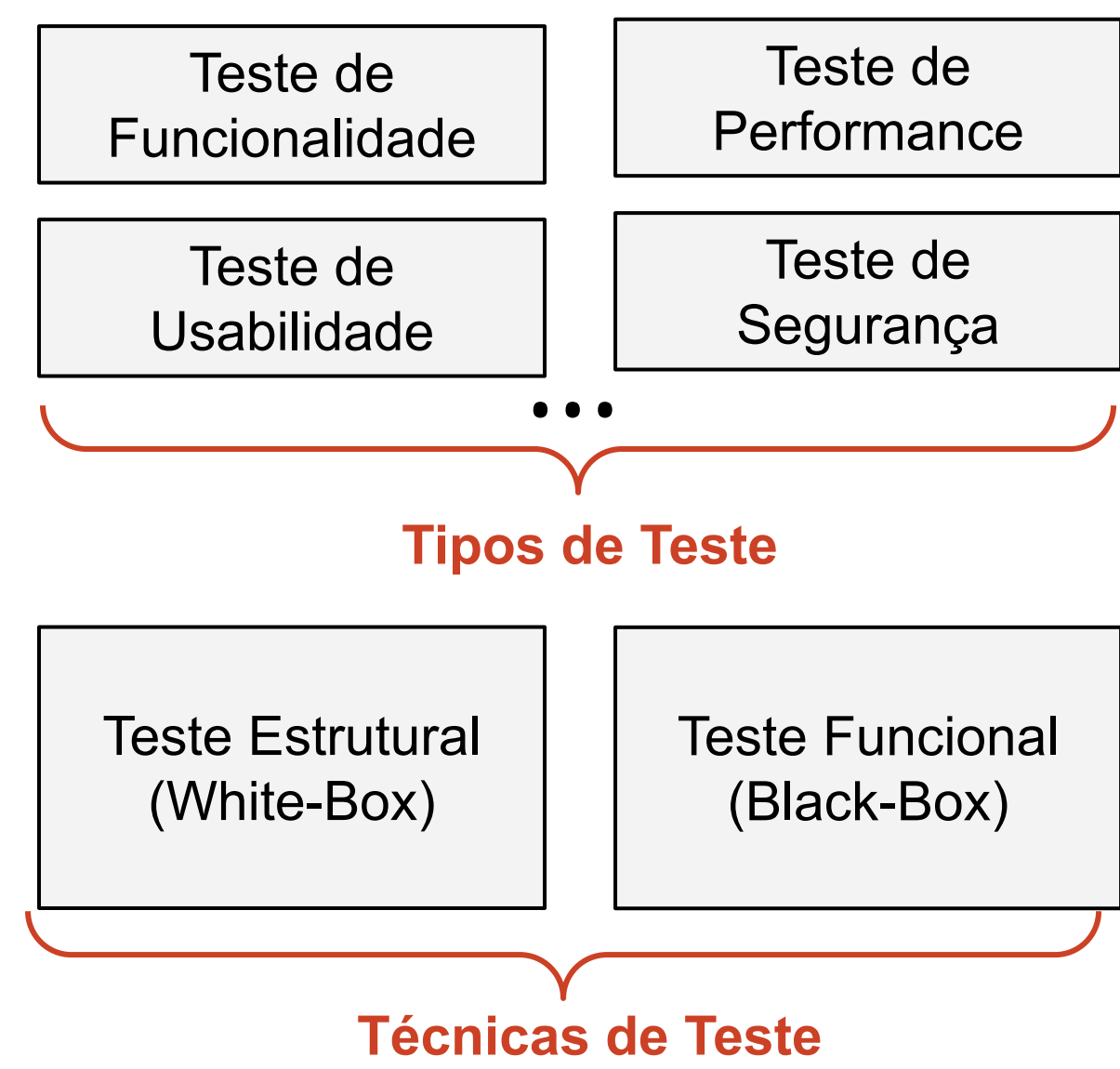
Conceitos Básicos

- **Engano**: Uma das causas da introdução de um defeito no software durante a implementação
- **Defeito**: A parte do software que possui uma implementação incorreta
- **Erro**: Ocorre em tempo de execução, quando o estado de execução do software desvia da especificação
- **Falha**: Resultado incorreto produzido pelo software

- **Verificação**
 - Estou implementado o software corretamente?
 - É isso que está especificado?
- **Validação**
 - Estou implementando o software certo?
 - É isso que o cliente quer?
- **Teste** (verificação dinâmica)
 - Execução do software para verificar se ele atende ao especificação

- **Caso de teste**
 - Pré-condições, que é o estado obrigatório do software antes do início do teste
 - Entradas a serem fornecidas como teste
 - Ação que faz o software cumprir o que será testado
 - Resultados esperados que software gere
 - Pós-condições, que é o estado obrigatório do software após o teste

Tipologia (Dimensões) de Testes



Teste de Aceitação

- Visa verificar se o software está pronto e se pode ser utilizado pelos usuários
- Pode ser formal, informal, alfa ou beta

Teste de Integração

- Visa verificar se duas ou mais unidades funcionam juntas
- Se concentra nas interfaces de comunicação entre unidades

Teste do Sistema

- O sistema é testado como um todo
- Visa verificar a compatibilidade, interação e troca de dados entre componentes

Teste de Unidades

- Para testar classes individuais
- Feito pelo próprio programador da classe
- Testa toda a interface da classe

Teste Manual

- Executado sem apoio ferramental

Teste Automatizado

- Executado com apoio ferramental, como Junit, PyTest, Selenium, etc

Teste Alfa

- Feito com participação de usuários e desenvolvedores, geralmente ambiente controlado

Teste Beta

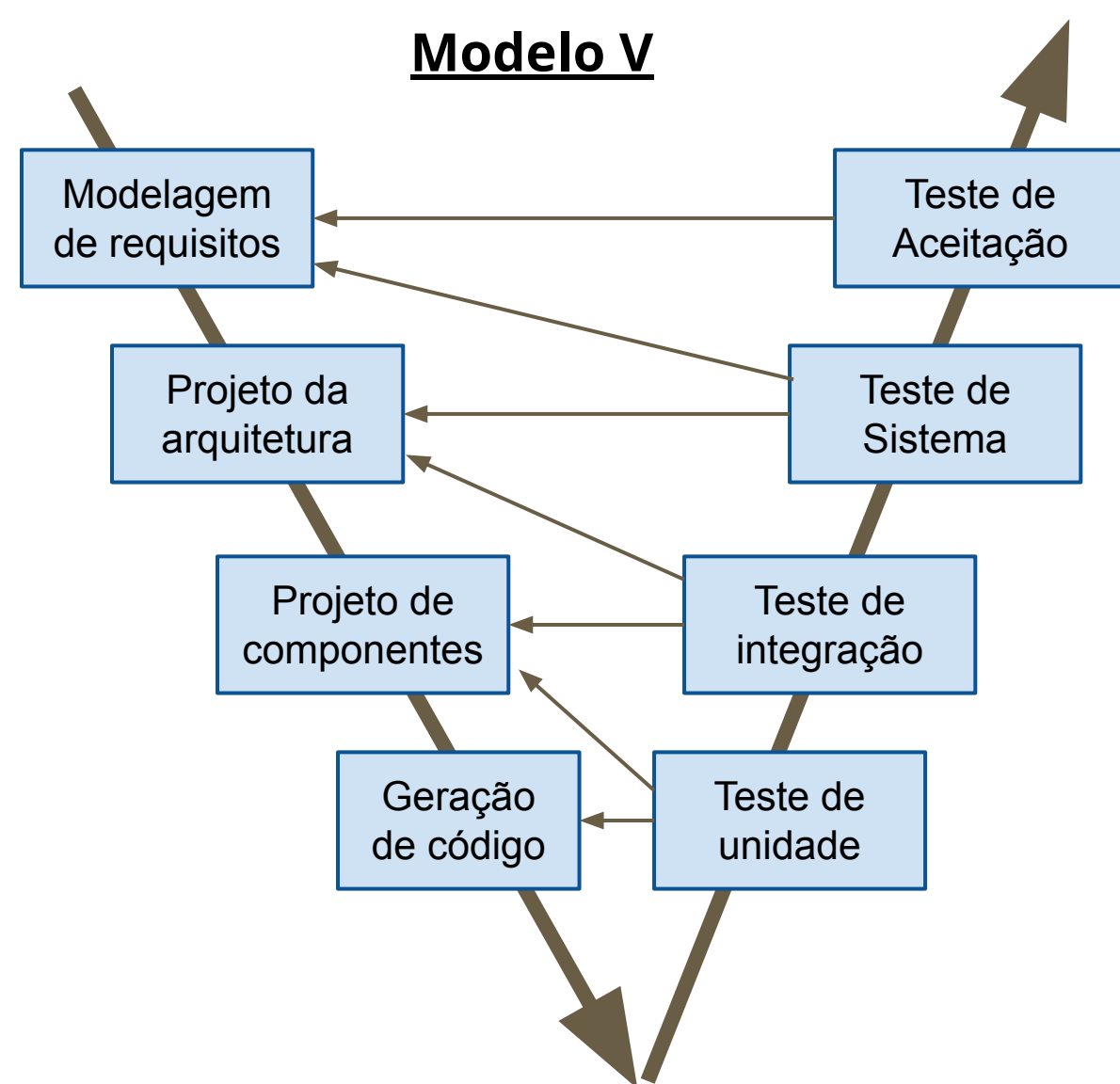
- Feito pelo usuário no sistema implantando no ambiente de uso

Teste de Regressão

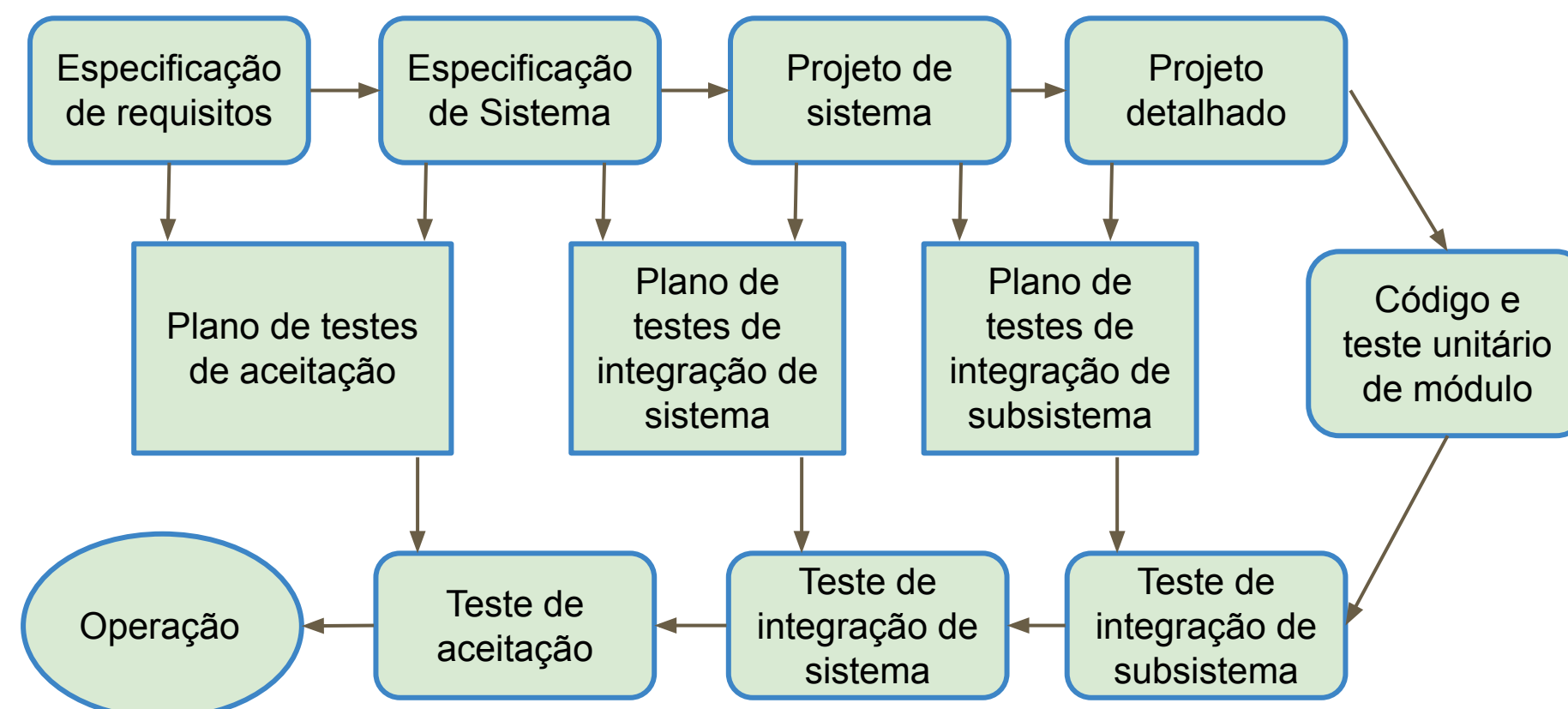
- Reexecução de casos de teste após uma mudança para garantir que o software não regrediu

Teste no Processo de Desenvolvimento de Software

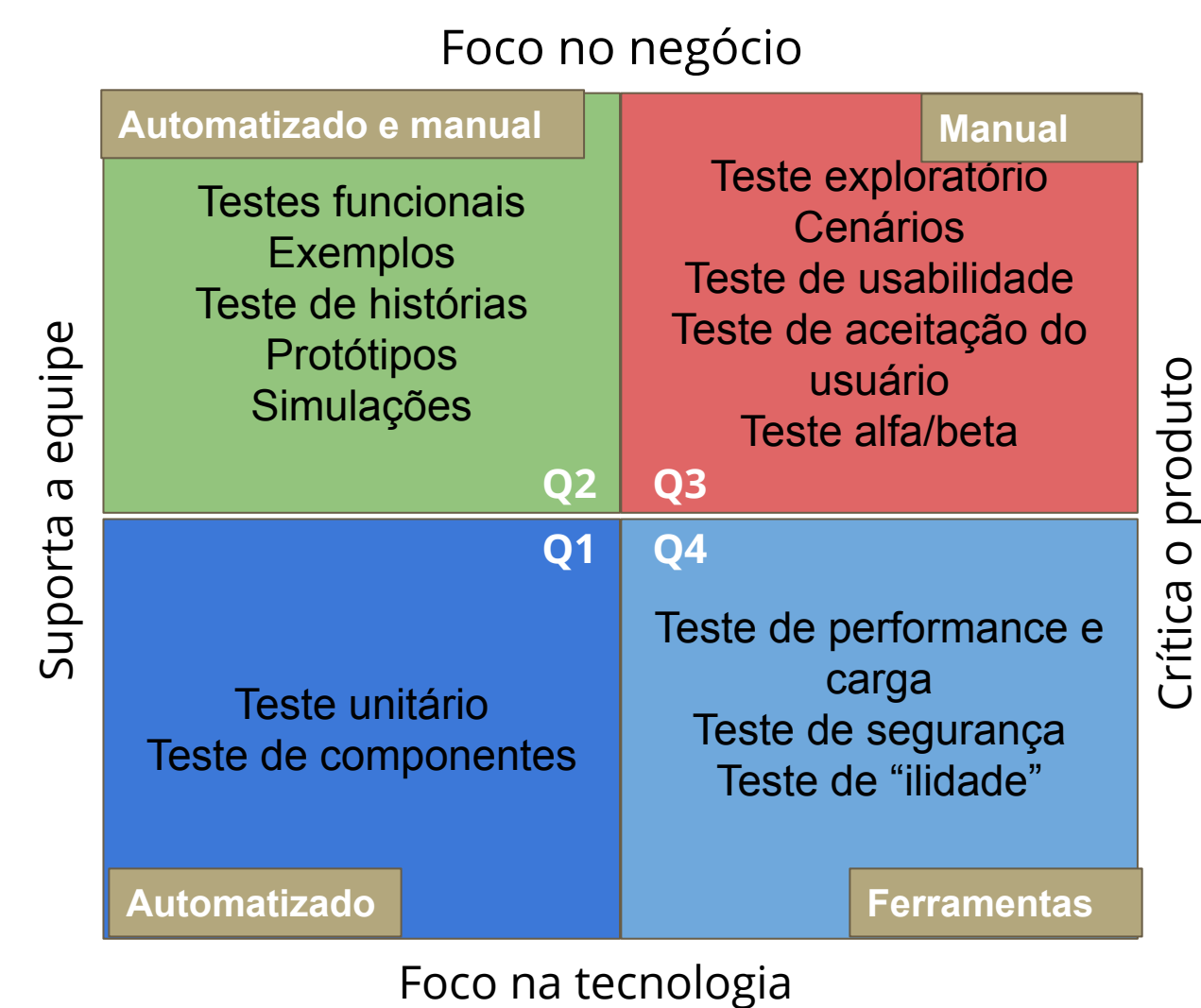
Modelo V



Processos dirigidos a planos

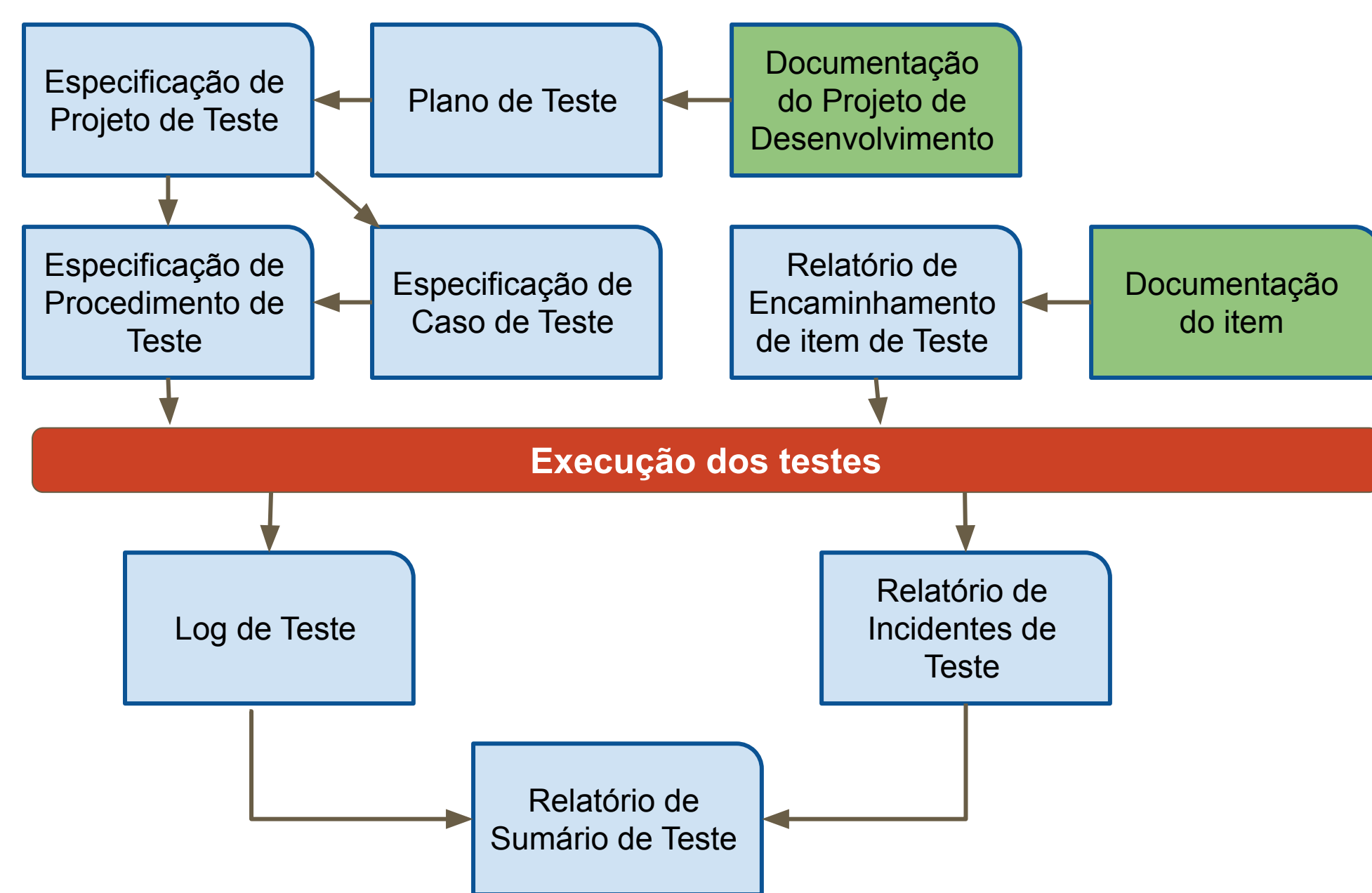


Quadrantes do Teste Ágil

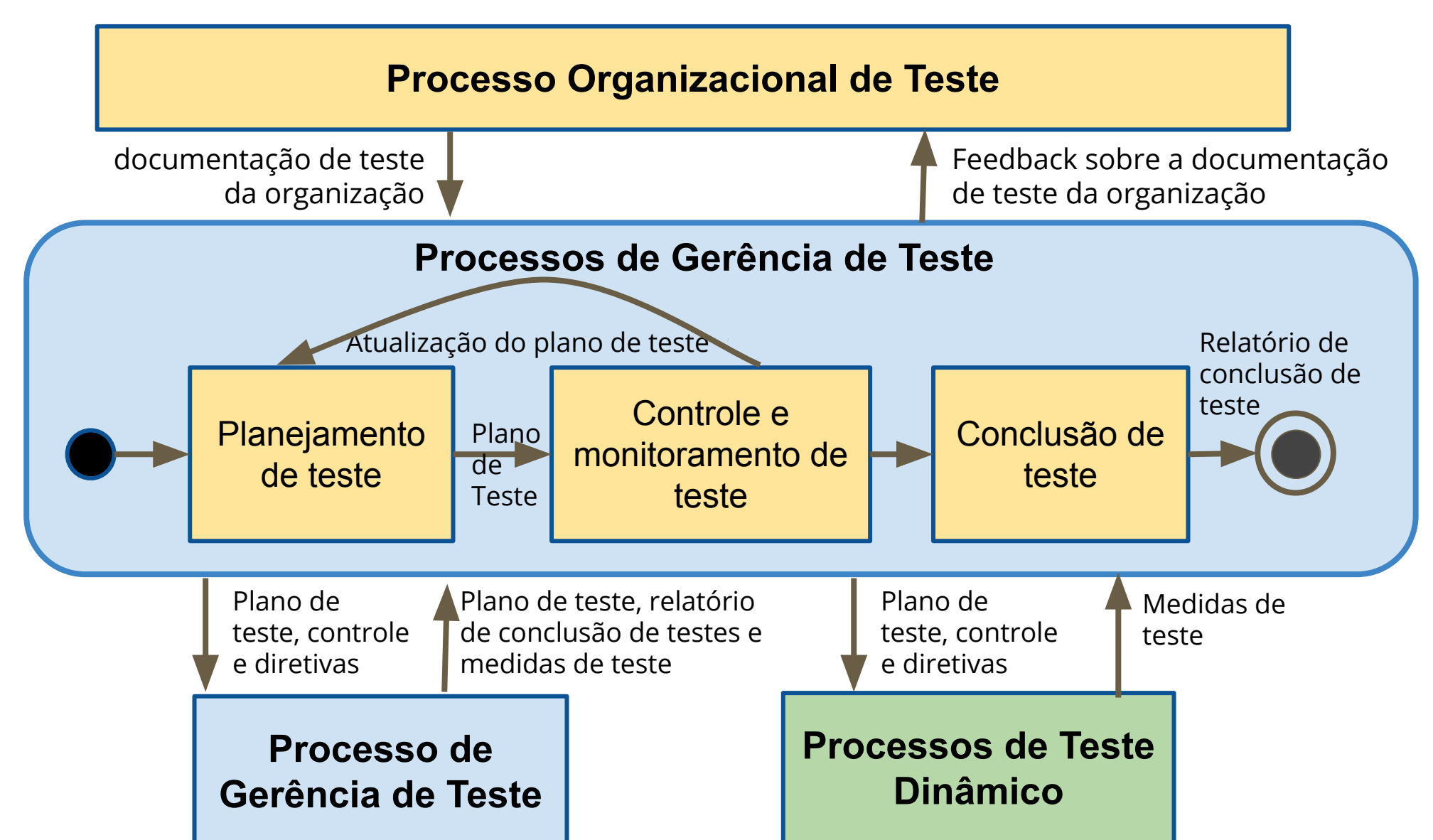


Padrões para Gestão do Processo de Teste de Software

IEEE 829-2008 - Standard for Software and System Test Documentation



ISO/IEC/IEEE 29119 - Software and systems engineering - Software testing



Necessidade de critérios para definir casos de teste

Quantos casos de teste possíveis existem para se testar cada funcionalidade de um sistema?

- Exemplo: 65536 casos de teste para testar um método que recebe um inteiro, em uma máquina de 16 bits

- Seja **P** um programa a ser testado.
- Seja **D(P)** o domínio de todos os casos de teste para **P**.
- Testar todo o domínio **D(P)** pode ser inviável
- A busca é por um conjunto **T** (sendo **T ⊂ D(P)**), bastante reduzido em relação a **D(P)** mas que, de certa maneira, representa cada um dos elementos de **D(P)**

Uma estratégia de teste determina o critério que deve ser seguido para se gerar um conjunto reduzido de casos de teste capaz de identificar defeitos se eles existirem.

Cartão de Suporte: Teste de Software



PUC Minas

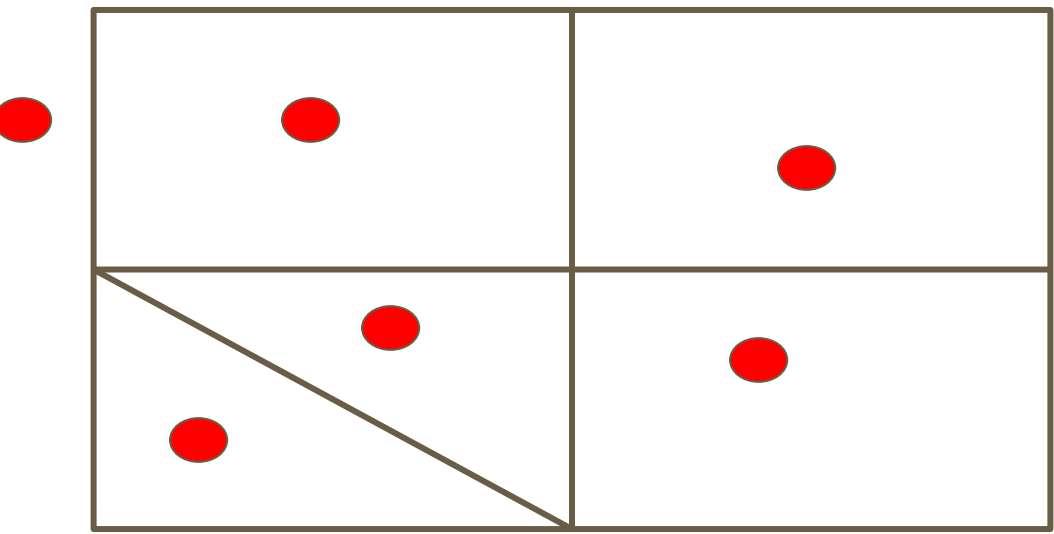
Instituto de Ciências Exatas e Informática

Curso de Engenharia de Software

Teste Funcional (Black-Box)

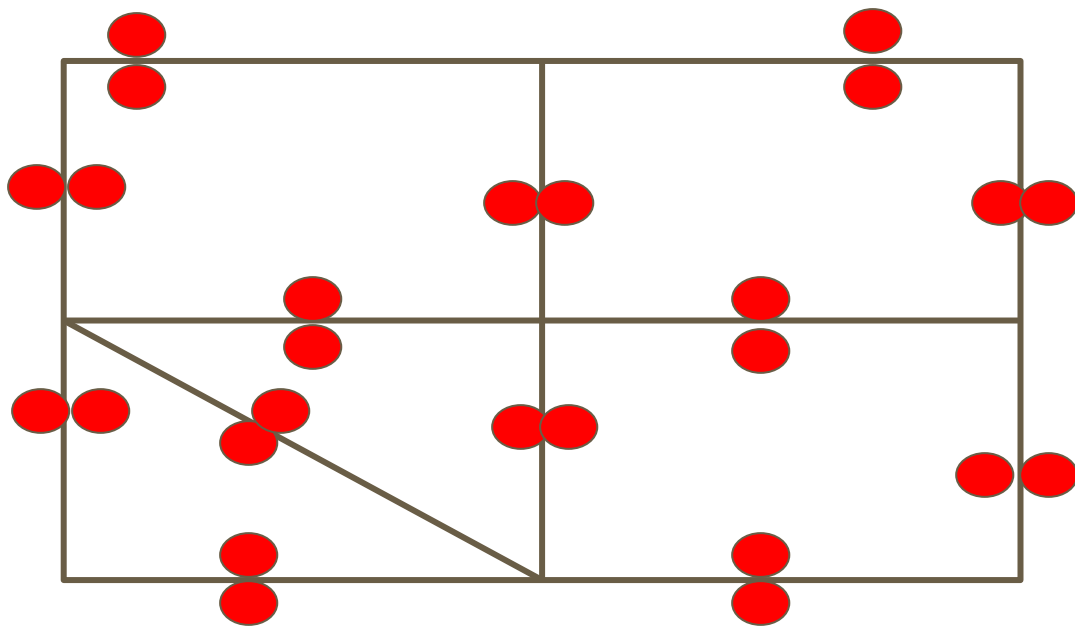
F

Particionamento de Equivalência



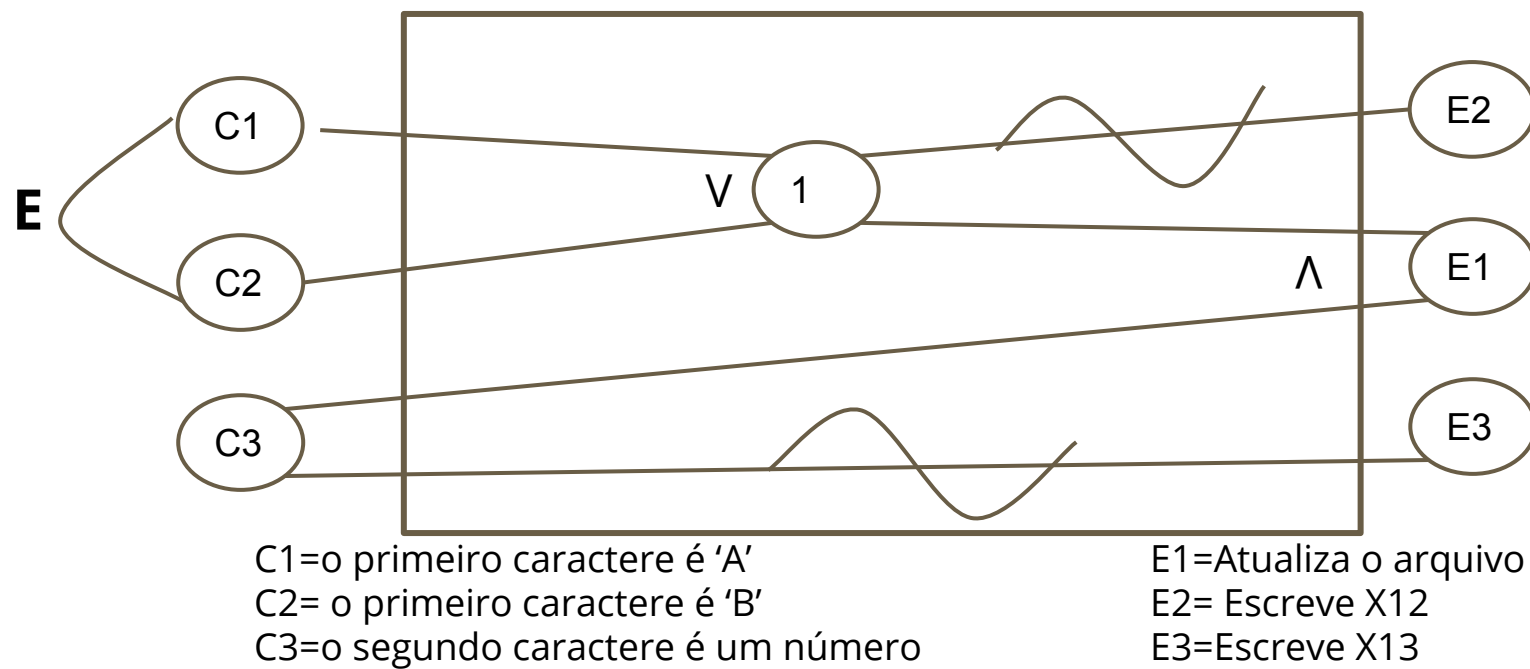
- Divide o domínio de entrada do software em classes de dados (classes de equivalências)
- Casos de teste são derivados a partir das classes de equivalência (válidas e inválidas)

Análise de Valores Limites



- Complementa o Particionamento de Equivalência
- Considera que os limites de uma classe de equivalência são fontes propícias a defeitos
- Casos de teste nos limites de classes de equivalência

Grafo de Causa-Efeito

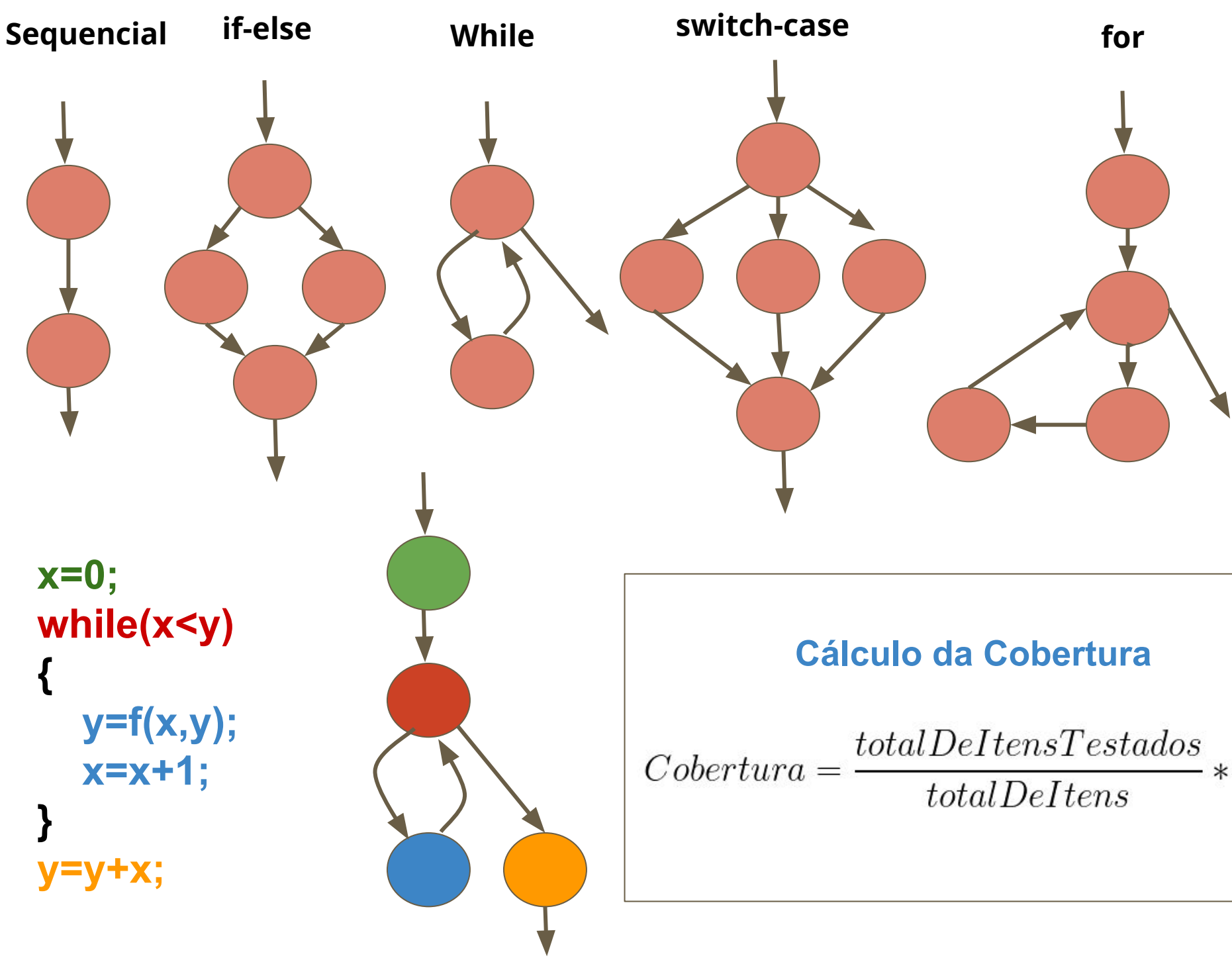


- Grafo que representa conjuntos de condições sobre entradas (causas) e as ações correspondentes do sistema (efeitos).
- A partir do grafo, monta-se uma tabela de decisão, e a partir desta, os casos de teste

Teste Estrutural (White-Box)

G

Grafo de Fluxo de Controle (Control Flow Graph - CFG)



Cálculo da Cobertura

$$Cobertura = \frac{totalDeItensTestados}{totalDeItens} * 100$$

Cobertura de comandos (statements)

- Casos de teste que exercitem todos os comandos software

Cobertura de decisões (branches)

- Casos de teste que exercitem todas as decisões em valores verdadeiros e falsos
 - If, while, for, switch-case, do

Cobertura de condições

- Casos de teste que exercitem todos os ramos do grafo (arestas) pelo menos uma vez

Cobertura de caminhos

- Casos de teste que exercitem todos os caminhos possíveis no grafo, do início ao fim
- Inclui cobrir todos os comandos, decisões e condições

Caminho Linearmente Independente

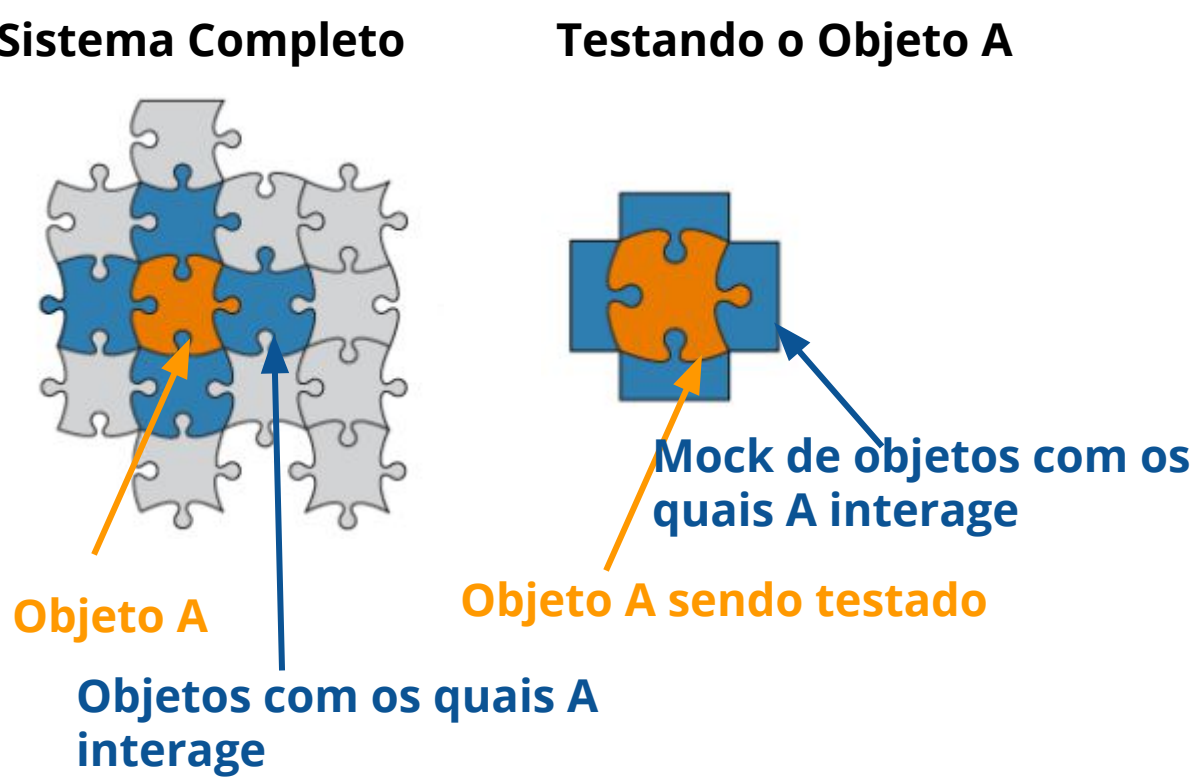
- Qualquer caminho que introduz pelo menos um novo conjunto de comandos ou uma nova condição
- no grafo, significa incluir pelo menos uma aresta que não tenha sido atravessada antes de o caminho ser definido

Complexidade Ciclomática

- Número de caminhos linearmente independentes
- $V(G) = E - N + 2$
 - E é o número de ramos do grafo
 - N é o número de nós do grafo
- $V(G) = P + 1$
 - P é o número de nós predados do grafo, aqueles que têm duas ou mais arestas saindo dele

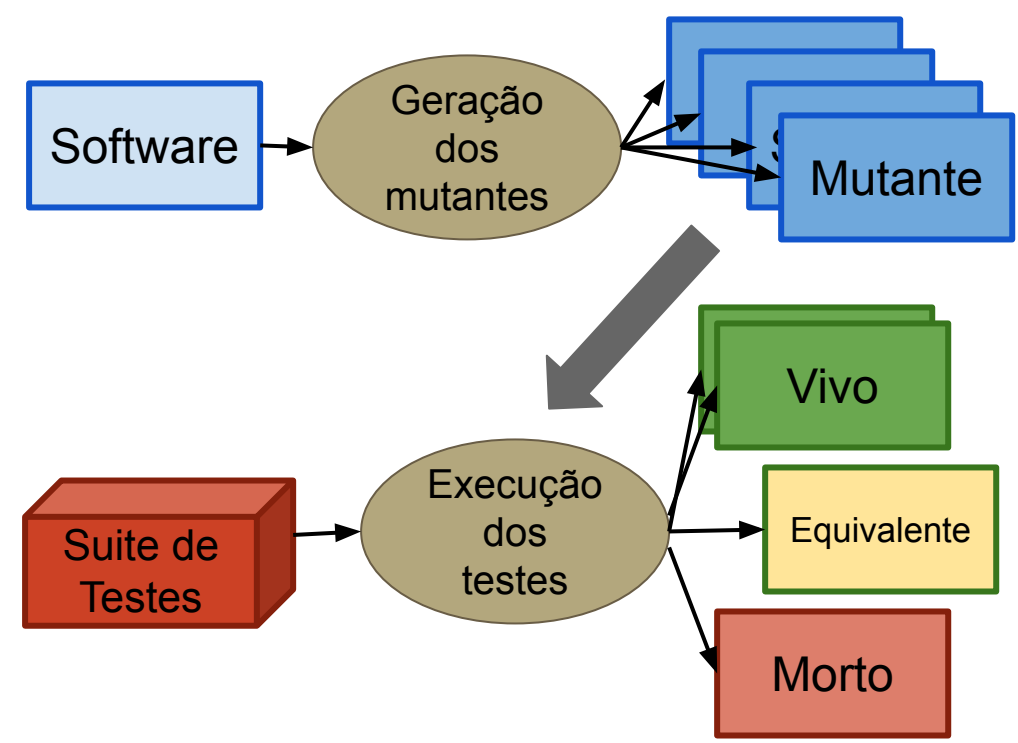
Teste com Mock Object

Objetos *mock* imitam objetos reais, simulam o comportamento de objetos reais complexos



Teste de Mutação

Casos de teste baseados em erros frequentes, variações sintáticas no código.



$MutationScore = 100 * D / (N - E)$, onde D = #mutantes mortos; N = # mutantes; E = #mutantes equivalentes

Métricas e Indicadores

H

Software	Testes
<ul style="list-style-type: none">• Fatia de dados (<i>data slices</i>)• Aglutinação (<i>stickiness</i>)• Número de módulos chamados (<i>fan-out</i>)• Número de módulos que chamam o módulo em consideração (<i>fan-in</i>)• Complexidade ciclomática	<ul style="list-style-type: none">• Cobertura (comandos, decisões, condições, caminhos)• Fator de Teste = N° Linha dos testes / N° Linhas do sistema• Mutation score• Número de asserções por método• Tempo de execução dos testes
Processo de Teste	
<ul style="list-style-type: none">• Total de Defeitos Detectados durante o desenvolvimento (DD)• Total de Defeitos Removidos durante o desenvolvimento (DR)• Total de Falhas Encontradas pelo Usuário (FU)• Eficácia na Detecção de Defeitos = $DD / (DD + FU) \times 100$	

Desenvolvimento Guiado Por Testes (TDD), Code Smell e Refatoração

Code Smell e Catálogo de Refatoração



- Código Duplicado
- Método Longo
- Classes Grandes
- Lista Longa de Parâmetros
- Alterações por Motivos Divergentes
- Cirurgia com Rifle
- Inveja dos Dados
- Agrupamento de Dados
- Obsessão por Tipos Primitivos
- Switches numerosos ou duplicados
- Hierarquias paralelas de herança
- Classes ociosas
- Generalidade especulativa
- Atributos temporários
- Cadeia de Mensagens
- Intermediários
- Intimidade inadequada
- Classes Alternativas com Interfaces Diferentes
- Biblioteca de classes incompleta
- Classes de Dados
- Herança Recusada

Depuração (debugging)

I

Quando um caso de teste cumpre sua função de revelar o defeito, entra em cena a atividade **depuração**

Estratégias de depuração

- por Força Bruta
- por Indução
- por Dedução
- por Backtracking
- por Teste

Considerações

Este cartão é um resumo de alguns dos tópicos abordados na disciplina TS lecionada pelo prof. Lesandro Ponciano na PUC Minas. Trata-se de um material complementar e de consulta rápida durante as aulas. Não pode e não deve ser usado como única fonte de estudo para as avaliações da disciplina. Não pode ser usado durante avaliações sem consulta.

Referências

- SOMMERVILLE, Ian. Engenharia de Software - 9a edição. Pearson ISBN 9788579361081
- DELAMARO, Márcio; MALDONADO, José; JINO, Mario. Introdução ao teste de software. Elsevier Brasil, 2016.
- MYERS, Glenford J. et al (2004) "The Art of Software Testing." 2ed. New York, NY, USA: John Wiley & Sons.