

**Engenharia de Software I – 2017.2**

# **Introdução a Verificação, Validação e Teste de Software**

**Ricardo A. Ramos**

# Organização

- Introdução
- Teste de Software
  - Terminologia e Conceitos Básicos
  - Técnicas e Critérios de Teste
  - Automatização da Atividade de Teste
  - Estudos Empíricos
- Perspectivas

# Introdução

## ➤ Qualidade de Software

*Conformidade com requisitos funcionais e de desempenho, padrões de desenvolvimento documentados e características implícitas esperadas de todo software profissionalmente desenvolvido.*

- Corretitude
- Confiabilidade
- Testabilidade

# Introdução

- Garantia de Qualidade de Software
  - Conjunto de atividades técnicas aplicadas durante todo o processo de desenvolvimento
  - Objetivo
    - Garantir que tanto o processo de desenvolvimento quanto o produto de software atinjam os níveis de qualidade especificados
  - VV&T – Verificação, Validação e Teste

# Introdução

- **Validação:** Assegurar que o produto final corresponda aos requisitos do usuário

*Estamos construindo o produto certo?*

- **Verificação:** Assegurar consistência, completitude e corretitude do produto em cada fase e entre fases consecutivas do ciclo de vida do software

*Estamos construindo corretamente o produto?*

- **Teste:** Examina o comportamento do produto por meio de sua execução

# Terminologia

- Defeito ➡ Erro ➡ Falha
  - Defeito: deficiência mecânica ou algorítmica que, se ativada, pode levar a uma falha
  - Erro: item de informação ou estado de execução inconsistente
  - Falha: evento notável em que o sistema viola suas especificações

# Defeitos no Processo de Desenvolvimento

- A maior parte é de origem humana
- São gerados na comunicação e na transformação de informações
- Continuam presentes nos diversos produtos de software produzidos e liberados (10 defeitos a cada 1000 linhas de código)
- A maioria encontra-se em partes do código raramente executadas

# Defeitos no Processo de Desenvolvimento

- Principal causa: tradução incorreta de informações
- Quanto antes a presença do defeito for revelada, menor o custo de correção do defeito e maior a probabilidade de corrigi-lo corretamente
- Solução: introduzir atividades de VV&T ao longo de todo o ciclo de desenvolvimento

# Teste e Depuração

## ➤ Teste

*Processo de execução de um programa com o objetivo de revelar a presença de erros.*

*Contribuem para aumentar a confiança de que o software desempenha as funções especificadas.*

## ➤ Depuração

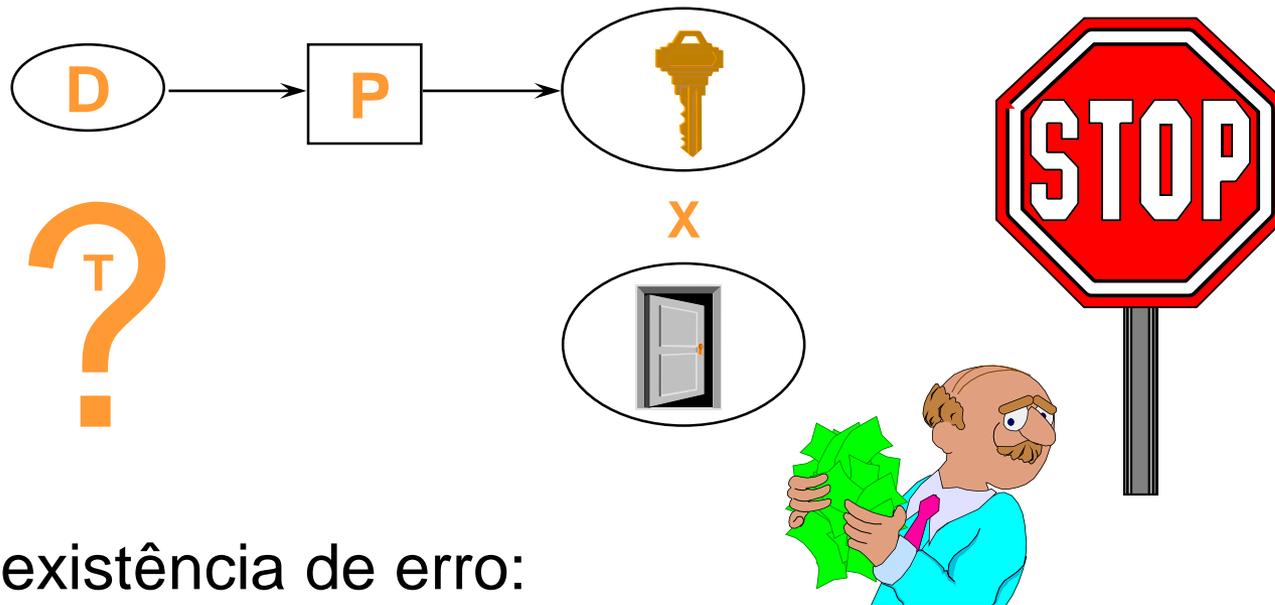
*Conseqüência não previsível do teste. Após revelada a presença do erro, este deve ser encontrado e corrigido.*

# Teste de Software

- Fundamental em todos os ramos de engenharia
  - Software: produto da Engenharia de Software
- Atividade essencial para ascensão ao nível 3 do Modelo CMM/SEI
- Atividade relevante para avaliação da característica funcionalidade (ISO 9126,14598-5)

# Teste de Software

**Objetivo: revelar a presença de erros**



- Inexistência de erro:
  - Software é de alta qualidade?
  - Conjunto de casos de teste  $T$  é de baixa qualidade?

# Teste de Software

- Defeitos e erros não revelados
  - Falhas se manifestam durante a utilização pelos usuários
  - Erros devem ser corrigidos durante a manutenção
- Alto custo

# Teste de Software

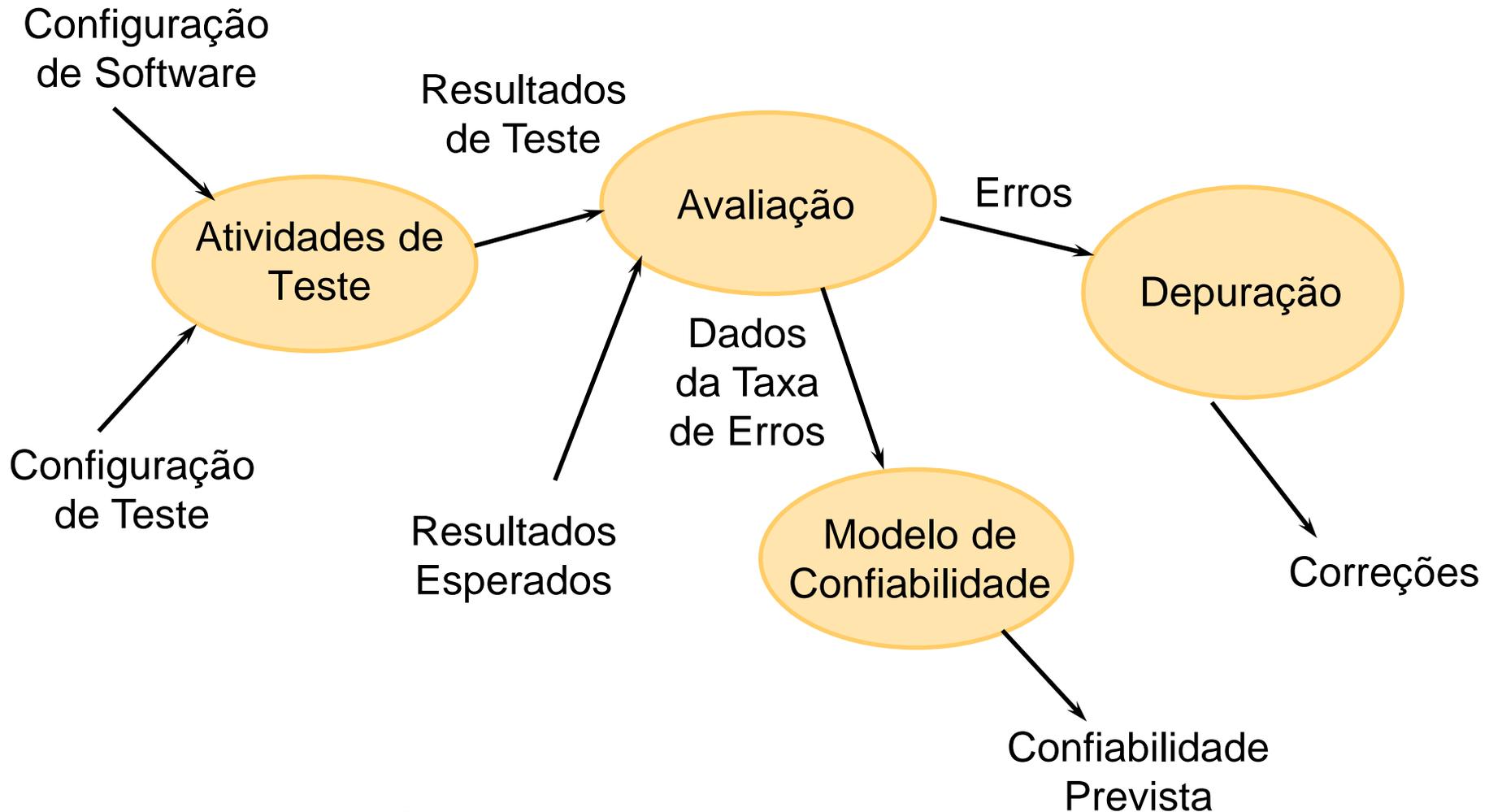
- Falhas graves
  - Qualidade e confiabilidade suspeitas
  - Modificação do projeto
  - Novos testes
- Erros de fácil correção
  - Funções aparentemente funcionam bem
  - Qualidade e confiabilidade aceitáveis
  - Testes inadequados para revelar a presença de erros graves
    - Novos testes

# Teste de Software

## ➤ Limitações

- Não existe um algoritmo de teste de propósito geral para provar a corretude de um programa
- Em geral, é indecidível se dois caminhos de um mesmo programa ou de diferentes programas computam a mesma função
- É indecidível se existe um dado de entrada que leve à execução de um dado caminho de um programa; isto é, é indecidível se um dado caminho é executável ou não

# Teste de Software



# Teste de Software

- Fases de Teste
  - Teste de Unidade
    - Identificar erros de lógica e de implementação em cada módulo do software, separadamente
  - Teste de Integração
    - Identificar erros associados às interfaces entre os módulos do software
  - Teste de Sistema
    - Verificar se as funções estão de acordo com a especificação e se todos os elementos do sistema combinam-se adequadamente

# Teste de Software

- Etapas do Teste
  - Planejamento
  - Projeto de casos de teste
  - Execução do programa com os casos de teste
  - Análise de resultados

# Teste de Software

- Caso de teste
  - Especificação de uma entrada para o programa e a correspondente saída esperada
    - Entrada: conjunto de dados necessários para uma execução do programa
    - Saída esperada: resultado de uma execução do programa
      - Oráculo
  - Um bom caso de teste tem alta probabilidade de revelar um erro ainda não descoberto

# Teste de Software

- Projeto de casos de teste
  - O projeto de casos de teste pode ser tão difícil quanto o projeto do próprio produto a ser testado
  - Poucos programadores/analistas gostam de teste e, menos ainda, do projeto de casos de teste
  - O projeto de casos de teste é um dos melhores mecanismos para a prevenção de defeitos
  - O projeto de casos de teste é tão eficaz em identificar erros quanto a execução dos casos de teste projetados

# Técnicas e Critérios de Teste

- **Técnica Funcional**
  - Requisitos funcionais do software
    - Critério Particionamento em Classes de Equivalência
- **Técnica Estrutural**
  - Estrutura interna do programa
    - Critérios Baseados em Fluxo de Dados
- **Técnica Baseada em Erros**
  - Erros mais freqüentes cometidos durante o processo de desenvolvimento de software
    - Critério Análise de Mutantes

# Automatização da Atividade de Teste

## ➤ Ferramentas de Teste

*Para a aplicação efetiva de um critério de teste faz-se necessário o uso de ferramentas automatizadas que apoiem a aplicação desse critério.*

- Contribuem para reduzir as falhas produzidas pela intervenção humana
  - Aumento da qualidade e produtividade da atividade de teste
  - Aumento da confiabilidade do software
- Facilitam a condução de estudos comparativos entre critérios

# Automatização da Atividade de Teste

- Critérios Estruturais: Fluxo de Dados
  - *Asset, Proteste* – programas em Pascal
  - *xSuds* – programas em C, C++ e Cobol
  - *Poke-Tool* – programas em C, Cobol e Fortran
- Critérios Baseados em Mutação
  - *Mothra* – programas em Fortran
  - *Proteum* – programas em C (unidade)
  - *Proteum/IM* – programas em C (integração)
  - *Proteum/RS* – especificações

# Automatização da Atividade de Teste

- *xSuds* (Software Understanding & Diagnosis System)
  - *xAtac*: teste
  - *xSlice*: depuração
  - *xVue*: manutenção
  - *xProf*: melhoria de performance
  - *xDiff*: comparação de código

*Estado da Arte X Estado da Prática*

# Técnica Funcional (Caixa Preta)

- Baseia-se na especificação do software para derivar os requisitos de teste
- Aborda o software de um ponto de vista macroscópico
- Envolve dois passos principais:
  - Identificar as funções que o software deve realizar (especificação dos requisitos)
  - Criar casos de teste capazes de checar se essas funções estão sendo executadas corretamente

# Técnica Funcional

- Problema
  - Dificuldade em quantificar a atividade de teste: não se pode garantir que partes essenciais ou críticas do software foram executadas
  - Dificuldade de automatização
- Critérios da Técnica Funcional
  - Particionamento em Classes de Equivalência
  - Análise do Valor Limite
  - Grafo de Causa-Efeito

# Técnica Estrutural (Caixa Branca)

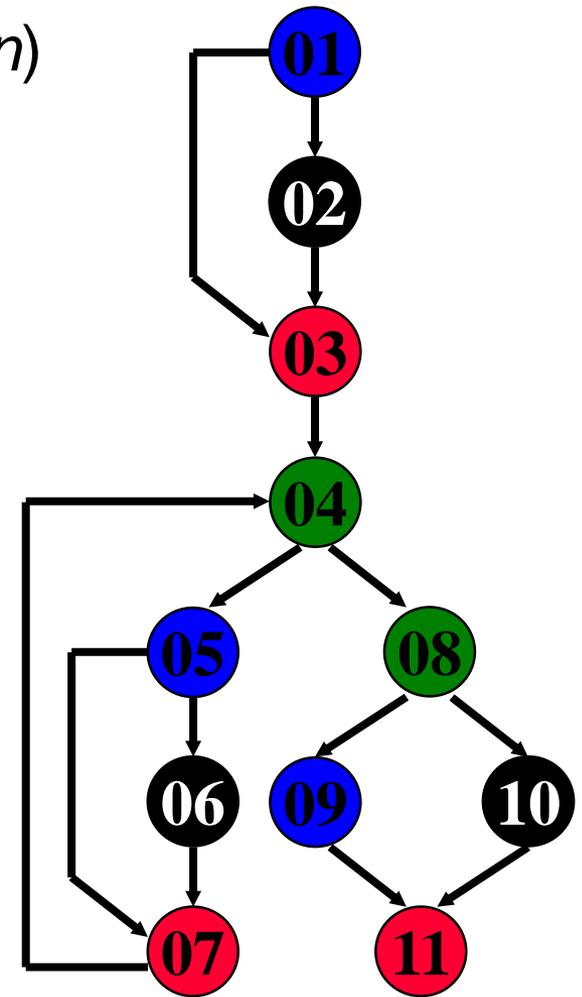
- Baseada no conhecimento da estrutura interna (implementação) do programa
- Teste dos detalhes procedimentais
- A maioria dos critérios dessa técnica utiliza uma representação de programa conhecida como grafo de programa ou grafo de fluxo de controle

# Técnica Estrutural

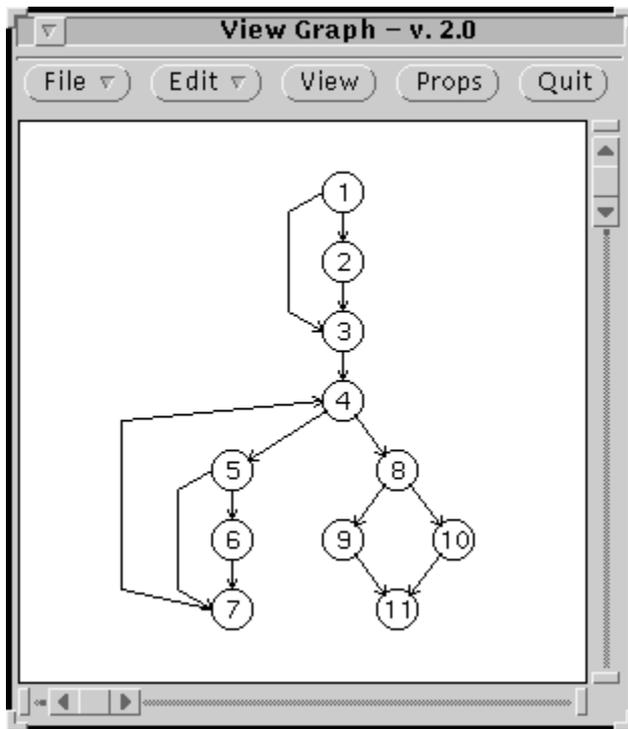
- Grafo de Programa
  - Nós: blocos “indivisíveis”
    - Não existe desvio para o meio do bloco
    - Uma vez que o primeiro comando do bloco é executado, os demais comandos são executados seqüencialmente
  - Arestas ou Arcos: representam o fluxo de controle entre os nós

# Identifier.c (função *main*)

```
/* 01 */ {  
/* 01 */     char  achar;  
/* 01 */     int   length, valid_id;  
/* 01 */     length = 0;  
/* 01 */     printf ("Identificador: ");  
/* 01 */     achar = fgetc (stdin);  
/* 01 */     valid_id = valid_s(achar);  
/* 01 */     if (valid_id)  
/* 02 */         length = 1;  
/* 03 */     achar = fgetc (stdin);  
/* 04 */     while (achar != '\n')  
/* 05 */     {  
/* 05 */         if (!(valid_f(achar)))  
/* 06 */             valid_id = 0;  
/* 07 */         length++;  
/* 07 */         achar = fgetc (stdin);  
/* 07 */     }  
/* 08 */     if (valid_id && (length >= 1) && (length < 6) )  
/* 09 */         printf ("Valido\n");  
/* 10 */     else  
/* 10 */         printf ("Invalido\n");  
/* 11 */ }
```



# Técnica Estrutural



Grafo de Programa do *identifier*  
Gerado pela *View-Graph*

## Grafo de Programa

- Detalhes considerados
  - nó
  - arco
  - caminho
    - simples (2,3,4,5,6,7)
    - completo (1,2,3,4,5,6,7,4,8,9,11)
  - fluxo de controle

# Técnica Baseada em Erros

- Os requisitos de teste são derivados a partir dos erros mais freqüentes cometidos durante o processo de desenvolvimento do software
- Critérios da Técnica Baseada em Erros
  - Semeadura de Erros
  - Teste de Mutação
    - Análise de Mutantes (unidade)
    - Mutação de Interface (integração)

# Teste de Mutação

## ➤ Hipótese do Programador Competente

*Programadores experientes escrevem programas corretos ou muito próximos do correto.*

## ➤ Efeito de Acoplamento

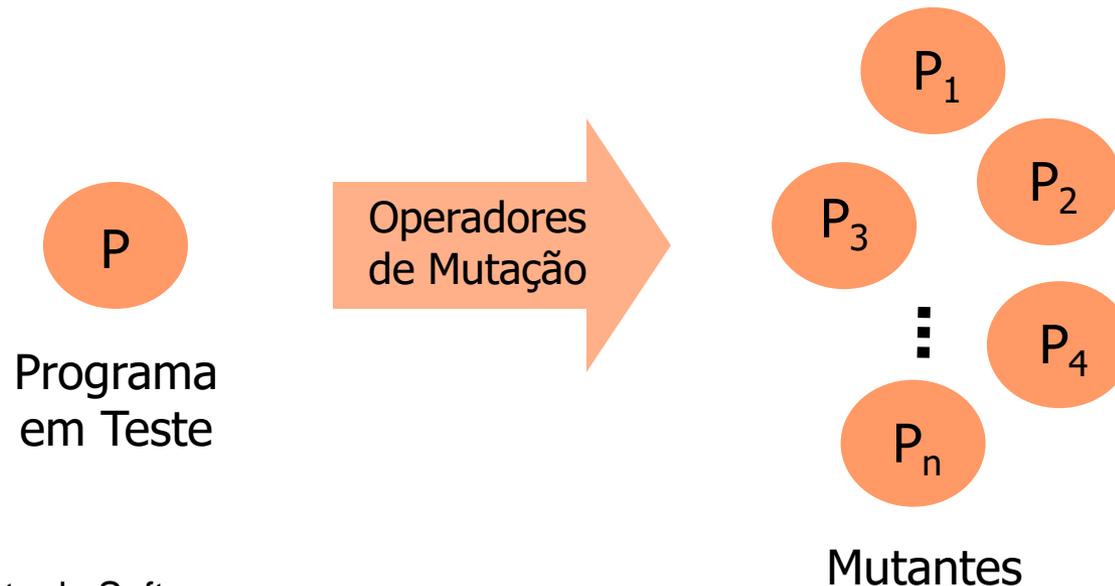
*Casos de teste capazes de revelar erros simples são tão sensíveis que, implicitamente, também são capazes de revelar erros mais complexos.*

# Análise de Mutantes

## ➤ Passos da Análise de Mutantes

### 1- Geração de Mutantes

*Para modelar os desvios sintáticos mais comuns, **operadores de mutação** são aplicados a um programa, transformando-o em programas similares: **mutantes**.*



# Análise de Mutantes

- Seleção dos operadores de mutação
  - Abrangente
    - Capaz de modelar a maior parte dos erros
  - Pequena cardinalidade
    - Problemas de custo
      - Quanto maior o número de operadores utilizados, maior o número de mutantes gerados

# Análise de Mutantes

## ➤ Exemplo de Mutantes

### Mutante Gerado pelo Operador OLAN

```
if (valid_id * (length >= 1) && (length < 6) )  
    printf ("Valido\n");  
else  
    printf ("Invalido\n");
```

### Mutante Gerado pelo Operador ORRN

```
if (valid_id && (length >= 1) && (length <= 6) )  
    printf ("Valido\n");  
else  
    printf ("Invalido\n");
```

# Exercício

1- Para a função abaixo

- a. elabore o grafo do programa
- b. determine os caminhos completos
- c. crie casos de teste que executem estes caminhos

A função recebe três valores e verifica se eles podem formar um triângulo. Três lados formam um triângulo quando um lado é menor que a soma dos outros dois.

```
function verificaTriangulo (a, b, c : real): String;
begin
if ((a < b + c) AND (b < a + c) AND (c < a + b)) then
    if (a = b) AND (b = c) then
        verificaTriangulo:='Triangulo equilatero' {Três lados iguais}
    else if ((a = b) OR (a = c) OR (b = c)) then
        verificaTriangulo:='Triangulo isosceles' {Dois lados iguais}
    else
        verificaTriangulo:='Triangulo escaleno'
    else verificaTriangulo:='Nao é um triangulo';      {Não satisfaz a propriedade}
end;
```

# Exercício

## 2 - Para a função abaixo

- elabore o grafo do programa
- determine os caminhos completos
- crie casos de teste que executem estes caminhos

/\* Este programa escrito em C lê uma linha de texto e converte conjuntos de caracteres brancos em um unico caractere. Sugira casos de teste para o programa \*/

```
void eliminaBranco(char linha[40])
{
    int i,j,tamanho;
    i=0; tamanho=strlen(linha);
    while(i<tamanho)
    {
        if(isspace(linha[i])) /* verifica se é "branco" */
        {
            if(isspace(linha[i+1])) /* verifica se o próximo é "branco" */
                for (j=i;j<tamanho;j++ )
                    linha[j]=linha[j+1]; /* copia o proximo caractere para a posicao vazia*/
            else
                i++;
        }
        else
            i++;
    }
} // fim do eliminaBranco()
```