# *Teste de Software Estrutural ou "Caixa-Branca"*

Disciplina de Engenharia de Software
prof. Andrey Ricardo Pimentel
andreyrp@hotmail.com

# Contexto da Aula

- Introdução a ES
- Qualidade
- Métricas de Software
- Planejamento do Projeto
- Análise e Projeto - UML
- Testes
  - Técnicas de testes
    - Teste funcional
    - Teste estrutural
    - Teste erro
  - Estratégias de testes
- Manutenção
- Gerenciamento da Configuração
- Engenharia Reversa e Reengenharia
- Reuso de Software
- Desenvolvimento Web
- ES e Software Livre

# *Tópicos*

- Introdução
- Teste Estrutural
- Teste de Caminho Básico
- Teste de Estrutura de Controle
- Conclusões
- Exercícios

# *Introdução*

- Teste de software é o processo de executar programas com o objetivo de encontrar defeitos
- Qualidade de software é a satisfação dos requisitos funcionais, de desempenho e normas de desempenho explicitamente declaradas
- É uma atividade essencial para se garantir a qualidade do software
- É uma das últimas atividades que fará a revisão do produto.

# *Introdução*

- Falhas em sistemas críticos:
  - Therac 25
  - Ariane 5 (erro em uma conversão de ponto flutuante de 64 bits para inteiro de 16 bits)
  - Titan IV e Titan IV b
  - Boeing 757 na Colombia
- Teste de software gasta 40% do esforço
- Objetivos do teste: Revelar erros ainda não descobertos

# *Introdução*

- Características de um bom teste:
  - Um bom teste tem alta probabilidade de encontrar um erro
  - Um bom teste não é redundante
  - Um bom teste não deve ser muito simples nem muito complexo
- Técnicas de teste:
  - Funcional (caixa preta)
  - Estrutural (caixa branca)
  - Baseada em erros

# *Teste Estrutural ou Caixa Branca*

- É um método de projeto de testes que usa a estrutura de controle do projeto procedimental para derivar casos de teste (Pressman, 2006)
- Baseia-se num minucioso exame dos detalhes procedimentais
- Caminhos lógicos do do software são testados
- Não é viável testar todos os caminhos lógicos de um programa (teste exaustivo)

# *Teste Exaustivo*

- Programa Pascal com 100 linhas e dois ciclos aninhados que executam entre 1 e 20 vezes cada um dependendo do dado da entrada.
- Dentro do ciclo interior 4 construções se-então-senão.
- $10^{14}$ caminhos possíveis de execução
- Se cada caso de teste for executado por um processador “mágico” de testes em 1 mseg
- 3170 anos para completar os testes.

# Teste de Caminho Básico

- Técnica de teste de caixa branca
- McCabe (1976)
- Conjunto Básico de caminhos de execução
- Os casos de teste derivados para executar os caminhos básicos tem a garantia de executar cada instrução pelo menos uma vez

# Casos de Teste

- Um caso de teste é composto de um dado de entrada (dado de teste) e de uma saída esperada
- Um bom caso de teste é aquele que tem alta probabilidade de revelar um defeito ainda não descoberto

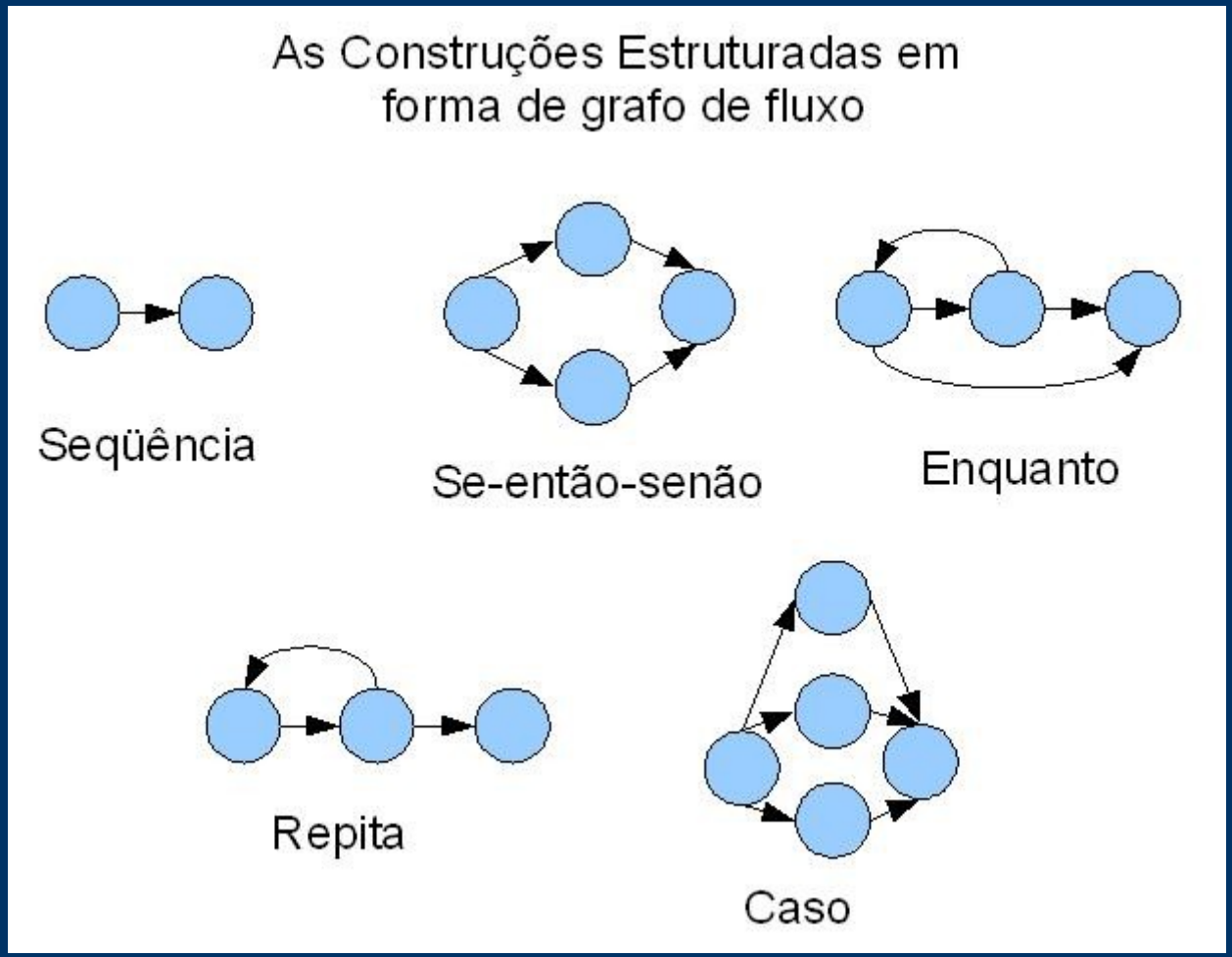
# Casos de Teste

- Os casos de teste no teste estrutural devem:
  - Garantir que todos os caminhos independentes de um módulo tenham sido exercitados pelo menos uma vez
  - Exercitem todas as decisões lógicas em seus lados verdadeiro e falso
  - Executem todos os ciclos nos seus limites e dentro de seus intervalos operacionais
  - Exercitem as estruturas de dados internas

# Notação Grafo de Fluxo

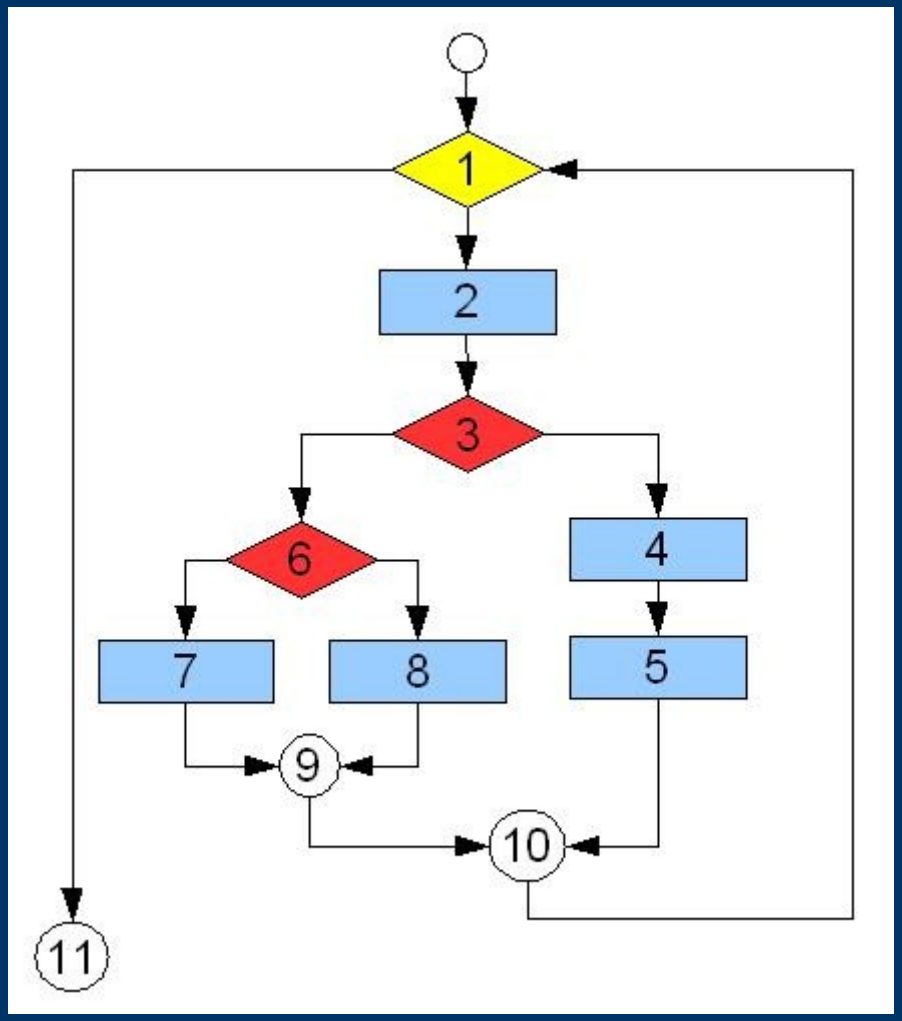
- O grafo de fluxo mostra o fluxo de controle
- Nós representam um ou mais processos
- Arestas representam o fluxo de controle
- Regiões do grafo são áreas limitadas pelas arestas e nós (incluindo a área fora do grafo)

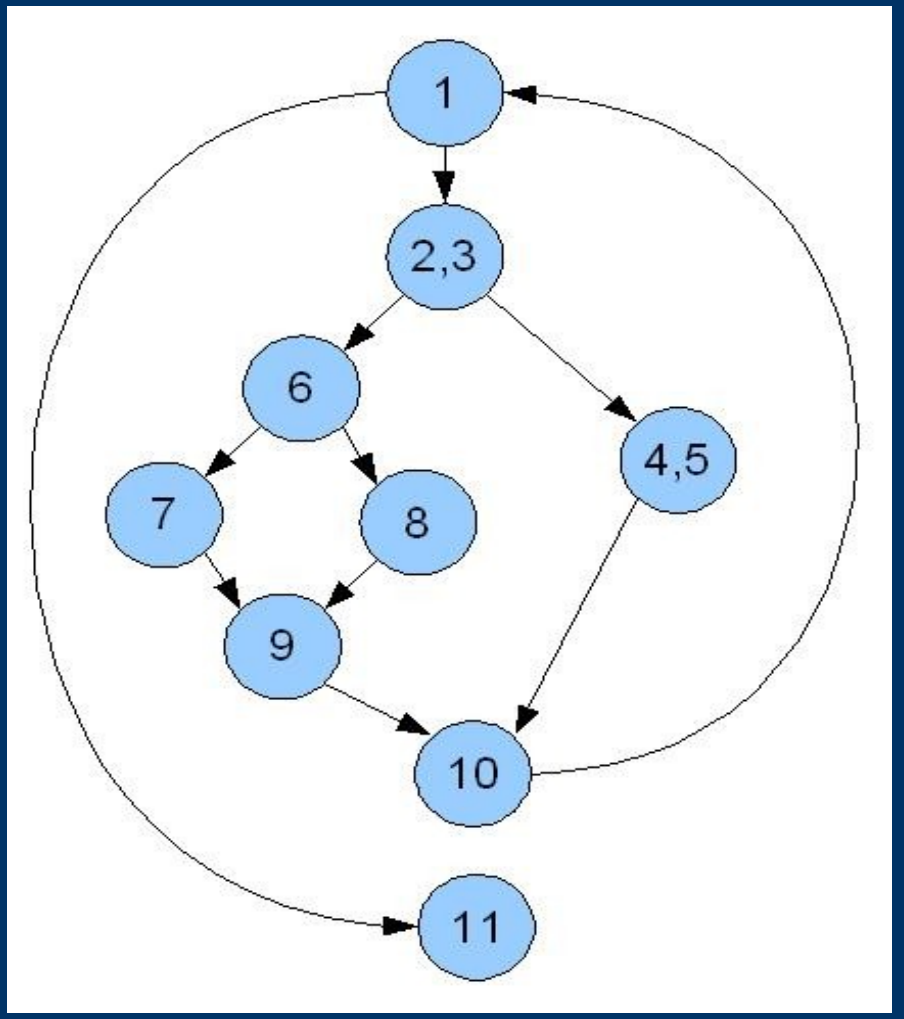
# Notação Grafo de Fluxo



# Notação Grafo de Fluxo

Derivando o grafo de fluxo a partir de um fluxograma

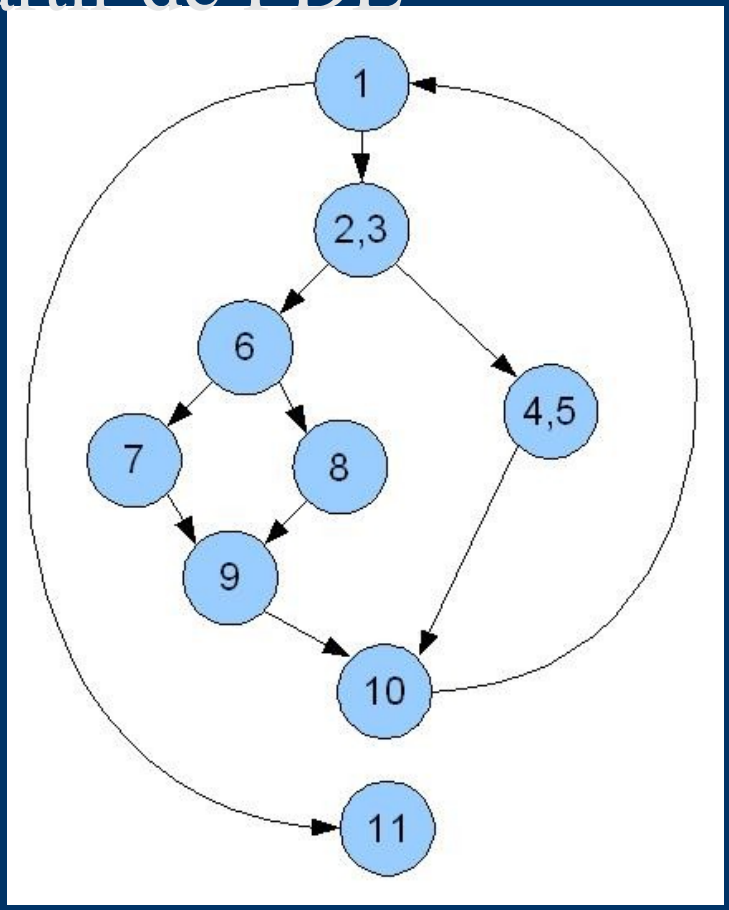


# Notação Grafo de Fluxo

Derivando o grafo de fluxo a partir de PDL

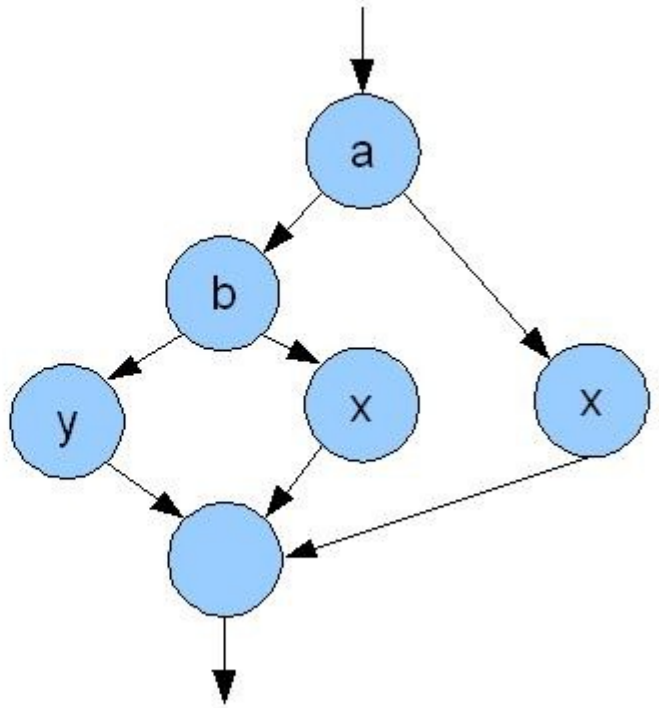
```
1:   enquanto existir registro faça
2:       leia registro
3:       se registro.campo1 = 0 então
4:           processar registro e armazenar em buffer
5:           incrementar contador
6:       senão se registro.campo2 = 0 então
7:           resetar contador
         senão
8:           processar registro e armazenar em arquivo
9:       fimse
10:      fimse
11:  fimenquanto
```



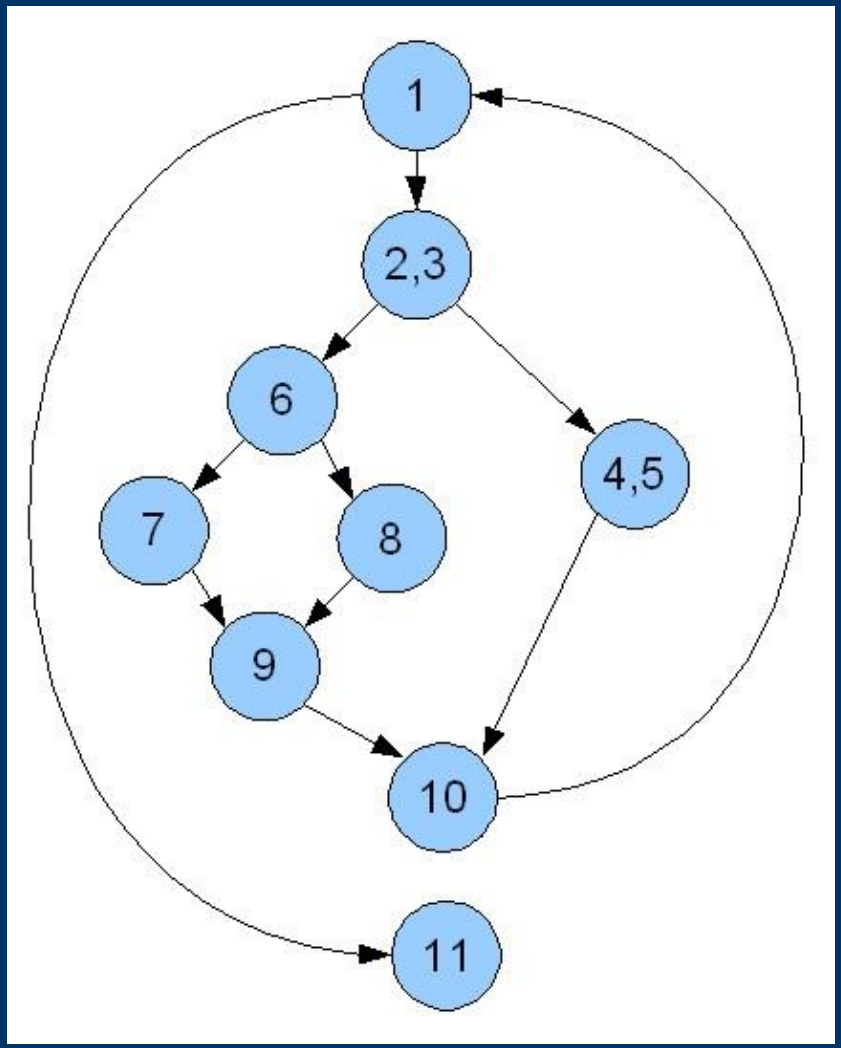
# *Notação Grafo de Fluxo*

Representando condições complexas em grafo de fluxo

```
...
se a ou b então
        procedimento
x
senão
        procedimento
y
fimse
...
```



# Caminhos Independentes




- Caminhos independentes:
  (1)1-11
  (2)1-2-3-4-5-10-1-11
  (3)1-2-3-6-8-9-10-1-11
  (4)1-2-3-6-7-9-10-1-11

  O caminho:
  1-2-3-4-5-10-1-2-3-6-8-6-10-1-11
  Não é independente

# *Complexidade Ciclomática*

- Como saber quantos caminho procurar?
- Complexidade Ciclomática
- O número de regiões do grafo de fluxo corresponde à complexidade ciclomática
- V(G) = E – N + 2
  - E : número de ramos do grafo
  - N : numero de nós do grafo
- V(G) = P + 1
  - P : número de nós predicados do grafo
- Nó predicado é o que tem duas ou mais arestas saindo dele

# *Derivação de Casos de Teste*

- Usando o projeto ou código como base, desenhe o grafo de fluxo correspondente
- Determine a complexidade ciclomática do grafo de fluxo correspondente
- Determine um conjunto base de caminhos linearmente independentes
- Prepare os casos de teste que vão forçar a execução de cada caminho do conjunto.

# *Exemplo*

- Derivar os casos de teste para um programa que calcula a média das entradas válidas, usando o método do caminho básico.

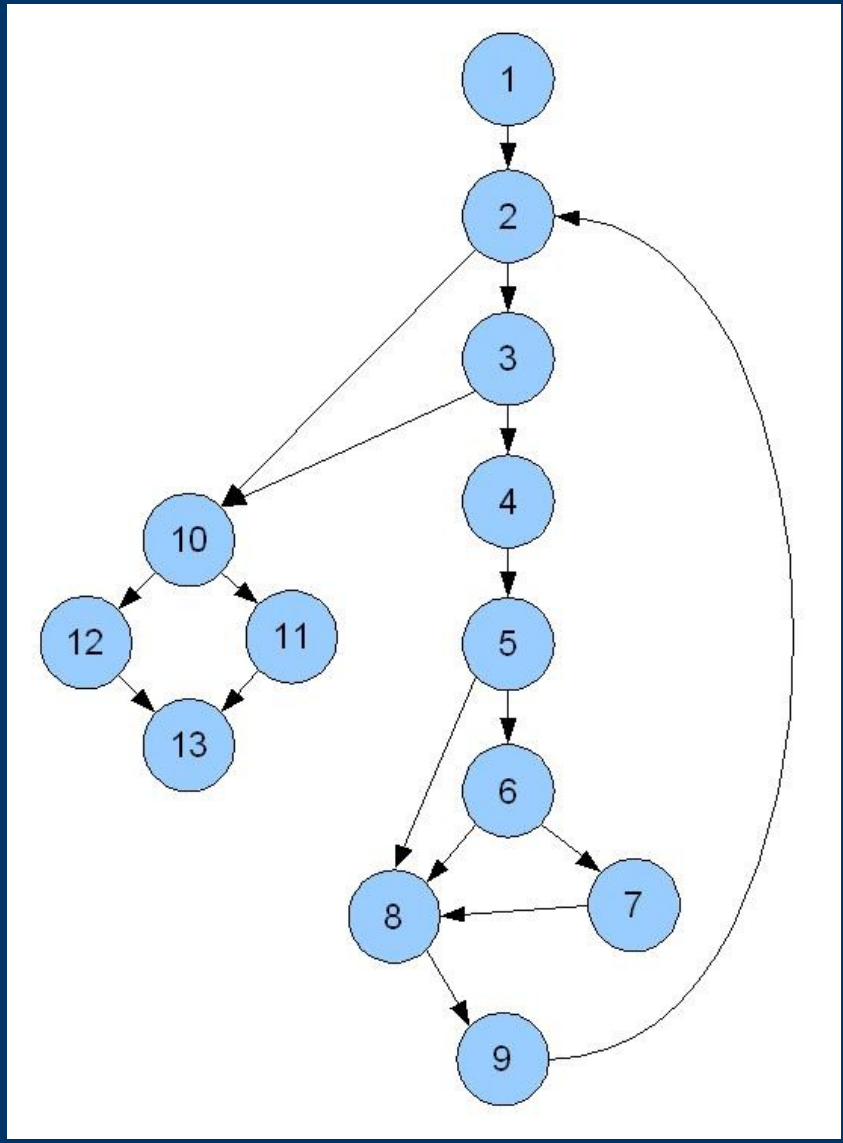
```
Procedimento media
INTERFACE ACEITA valor, min, max
INTERFACE RETORNA media, entradas, validas

var
   valor[1..100] vetor de real
   media, entradas, validas, min, max, soma: real
   i : inteiro
inicio
   i = 1
   totalEntradas = 0
   totalValidas = 0
   soma = 0
   enquanto valor[i]<>-999 e entradas<100 faça
      entradas=entradas+1
      se valor[i]>=min e valor[i]<=max então
         validas=validas+1
         soma=soma+valor[i]
      senão pule
      fimse
      i=i+1
   fimenquanto
   se validas>0 então
      media=soma/validas
   senão
      media=-999
   fimse
fim
```

1 2 3 4 5 6 7 8 9 10 11 12 13

# Exemplo

- Passo 1: Desenhe o grafo de fluxo correspondente.



# Exemplo

- Passo 2: Calcule a complexidade ciclomática.
- V(G) = 6 regiões
- V(G) = 17 arestas – 13 nós + 2 = 6
- V(G) = 5 nós predicados + 1 = 6

# *Exemplo*

- Passo 3: Determine um conjunto base de caminhos independentes.
- Caminho 1: 1-2-10-11-13
- Caminho 2: 1-2-10-12-13
- Caminho 3: 1-2-3-10-11-13
- Caminho 4: 1-2-3-4-5-8-9-2...
- Caminho 5: 1-2-3-4-5-6-8-9-2...
- Caminho 6: 1-2-3-4-5-6-7-8-9-2...

# *Exemplo*

- Passo 4: Prepare os casos de teste que vão forçar a execução de cada caminho
- O caminho 1 só pode ser testado como parte dos caminhos 4, 5 e 6
- Caminho 2: valor (i) = -999; resultados esperados: média = -999 e os outros valores com os valores iniciais.
- Caminho 6: valor (i) = entrada válida; resultados esperados: média correta baseada em n valores e totais apropriados.

# Teste de Estrutura de Controle

- O teste do caminho básico é simples e eficaz, mas nem sempre é suficiente.
- Outras variações:
  - Teste de condição
  - Teste de fluxo de dados
  - Teste de ciclo

# Teste de Condição

- Método de projeto de teste que exercita as condições booleanas de um módulo de programa
- Condição Simples:
  - E1 <operador relacional>E2
- Condição Composta:
  - Operadores Booleanos E, OU e NÃO
- O método de teste de condição focaliza o teste de cada condição para garantir que não contém erros

# Teste de Fluxo de Dados

- Seleciona caminhos de teste de acordo com as definições e dos usos das variáveis do programa (potenciais usos)
  - DEF(S) = {X | comando S contém definição de X}
  - USO(S) = {X | comando S contém uso de X}
- Cadeia DU (definição-uso)
  - A definição de X no comando S é viva no comando S' se existir um caminho entre S e S' sem outra definição de X
  - Cadeia DU de X: [X, S, S']
- Cada cadeia DU deve ser coberta pelo menos uma vez

# *Teste de Ciclos*

- Focaliza a validade das construções dos ciclos
- Ciclos: simples, concatenados, aninhados e desestruturados.
- Testes para Ciclos simples:
  - Pule o ciclo completamente
  - Apenas uma passagem pelo ciclo
  - Duas passagens
  - m passagens pelo ciclo, onde m < n
  - n – 1, n, n+1 passagens

# Teste de Ciclos

- Testes para Ciclos aninhados:
  - Comece no ciclo mais interno, outros ciclos nos valores mínimos
  - Teste o ciclo mais interno com os outros ciclos nos valore mínimos, incluindo valores fora do intervalo e excluídos
  - Trabalhe em direção ao exterior passando para o ciclo seguinte com os ciclos externos em valores mínimos e os internos em valores típicos
  - Continue até que todos os ciclos tenham sido testados

# Exemplo

- Programa que calcula o fatorial de um número
- Planejar os testes de ciclo para este programa.

```
Programa fatorial
var: N, FAT, I : inteiro

inicio
    leia N
    se N for número e N>=0 então
        FAT = 1
        I = 2
        enquanto I <= N faça
            FAT = FAT * I
            I = I + 1
        fimenquanto
        escreva FAT
    senão
        escreva "entrada inválida"
    fim se
fim
```

# *Ferramentas Automatizadas*

- PokeTool (UNICAMP)
  - Testes
- PROTEUM e PROTEUM/IM (USP)

# *Exercícios*

- Construa os seguintes algoritmos e projete os casos de teste usando a técnica do caminho mínimo e o teste de ciclo.
  - Um algoritmo que lê um número e imprime a lista dos seus divisores
  - Um algoritmo que lê dois números e calcula o máximo divisor comum pelo método de Euclides.
  - Um algoritmo que lê as 4 notas de um aluno e diga se ele passou por média, está em final ou reprovou.

# Conclusões

- Métodos Estruturais se baseiam na estrutura de controle do programa
- Técnica de caminho básico
  - Grafo de fluxo
  - Caminhos independentes
  - Complexidade ciclomática
- Técnica de estrutura de controle
  - Teste de condição
  - Teste de fluxo de dados
  - Teste de ciclo

# Conclusões

- Funções ou métodos mais simples podem ser testados com métodos funcionais (caixa preta) enquanto que funções ou métodos mais complexos podem ser melhor testados com métodos estruturais (caixa branca)

# Referências


- Pressman, R. S. Engenharia de Sofware, 6a. ed. McGraw Hill, 2006
- Barbosa, E.; Maldonado, J.C.; Vincenzi, A.; Delamaro, M.; Souza, S.; Jino, M.. Introdução ao teste de Software. Curso ministrado no XIV congresso da SBES, 2000.