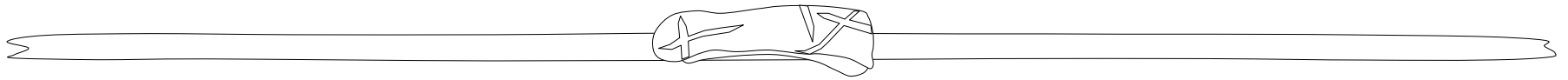


Testes de Software



Aula 14

Sumário

- # Objectivos e Princípios dos Testes
- # Software fácil de testar
- # Desenho de casos de teste
- # Testes
 - ▣ Caminho básico
 - ▣ Estrutura de controlo
 - ▣ Caixa preta
 - ▣ Ambiente, arquitectura e aplicações
 - ▣ OO

Objectivos

- # Descobrir erros
- # Um bom teste tem uma alta probabilidade de detectar erros não descobertos
- # Um teste tem sucesso SE descobrir um erro não detectado

Princípios

- # Os testes deviam detectar desde erros de código até a satisfação de requisitos
- # Devem ser planificadas
- # Isolar módulos "suspeitos" e testá-los aprofundadamente
- # Começar pelos módulos e terminar no sistema inteiro
- # Não é possível fazer testes exaustivos
- # Realizadas por equipas independentes

Software fácil de testar

- # Operatividade (quanto melhor funcionar, + eficientemente se pode testar)
 - Tem poucos erros
 - Os erros não bloqueiam a execução dos testes
 - O produto evoluciona em fases
- # Observabilidade (testas só o quê ves)
 - Saída diferente para cada entrada
 - Estados e variáveis do sistema podem ser consultados durante a execução
 - Estados e variáveis do sistema antigos podem ser consultados (registos de transacção)
 - Todos os factores que afectam os resultados são visíveis
 - Um resultado incorrecto identifica-se com facilidade
 - Os erros internos detectam-se automaticamente
 - O código fonte é acessível

Software fácil de testar (ii)

- # Controlabilidade (quanto + controlável, + automatizável e otimizável)
 - Todos os resultados possíveis gerados através da combinação de entradas
 - Estados e variáveis do sistema são controláveis pelo programador
 - Formatos de E/S são estruturados e consistentes
 - Os testes podem especificar-se, automatizar-se e reproduzir-se convenientemente
- # Modularidade (controlando o âmbito dos testes, os problemas podem ser isolados)
 - O sistema está construído em módulos independentes
 - Os módulos podem ser testados independentemente

Software fácil de testar (iii)

- # Simplicidade (há menos que testar)
 - ▣ Simplicidade funcional (características mínimas)
 - ▣ Simplicidade estrutural (que impede propagação de falhos)
 - ▣ Simplicidade do código (selecção de standards, boas práticas de programação)
- # Estabilidade (menos mudanças)
 - ▣ Mudanças infrequentes
 - ▣ Mudanças controladas
 - ▣ Mudanças não invalidam testes
 - ▣ O software recupera bem dos falhos

Software fácil de testar (iv)

- # Facilidade de compreensão (+informação, melhores testes)
 - ▣ O desenho foi compreendido
 - ▣ Dependências entre componentes internos foi compreendida
 - ▣ Mudanças ao desenho foram comunicadas
 - ▣ Documentação técnica acessível instantaneamente, está bem organizada, es exacta, específica e detalhada.

Desenho de casos de teste

- # Caixa preta
- # Caixa branca, garante o exercício de:
 - ▣ todos os caminhos independentes são utilizados
 - ▣ Todas as vertentes falsas e verdadeiras das decisões lógicas
 - ▣ Todos os ciclos nos seus limites
 - ▣ Todas as estruturas internas de dados
- # Permite ver
 - ▣ Erros lógicos e supostos incorrectos
 - ▣ Fluxo lógico real dum programa
 - ▣ Erros tipográficos

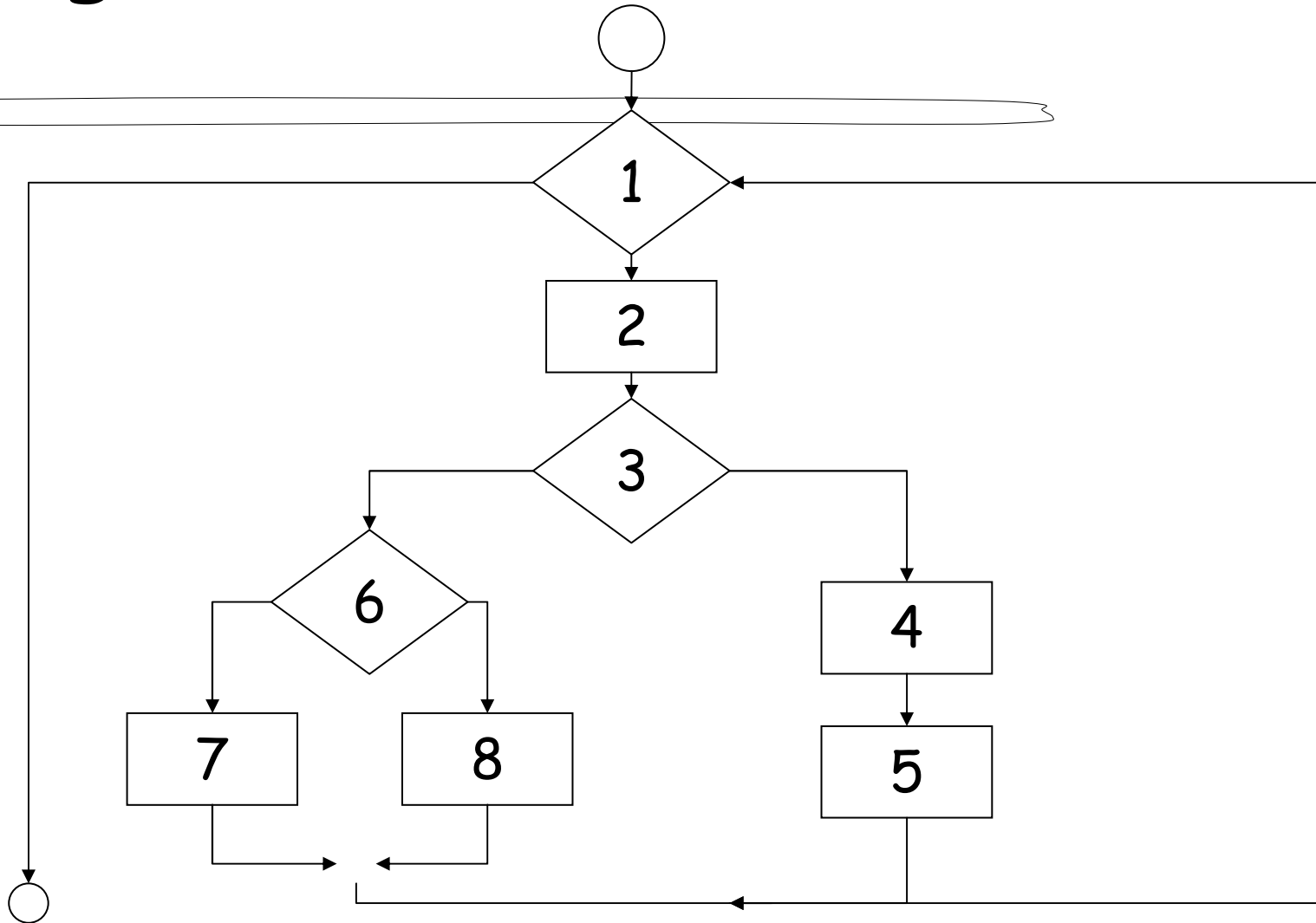
Teste do caminho básico

- # Método de caixa branca
- # Medida da complexidade lógica
- # Mede o n° de caminhos independentes
- # Os casos de teste obtidos por este método garantem que cada instrução é executada pelo menos 1 vez

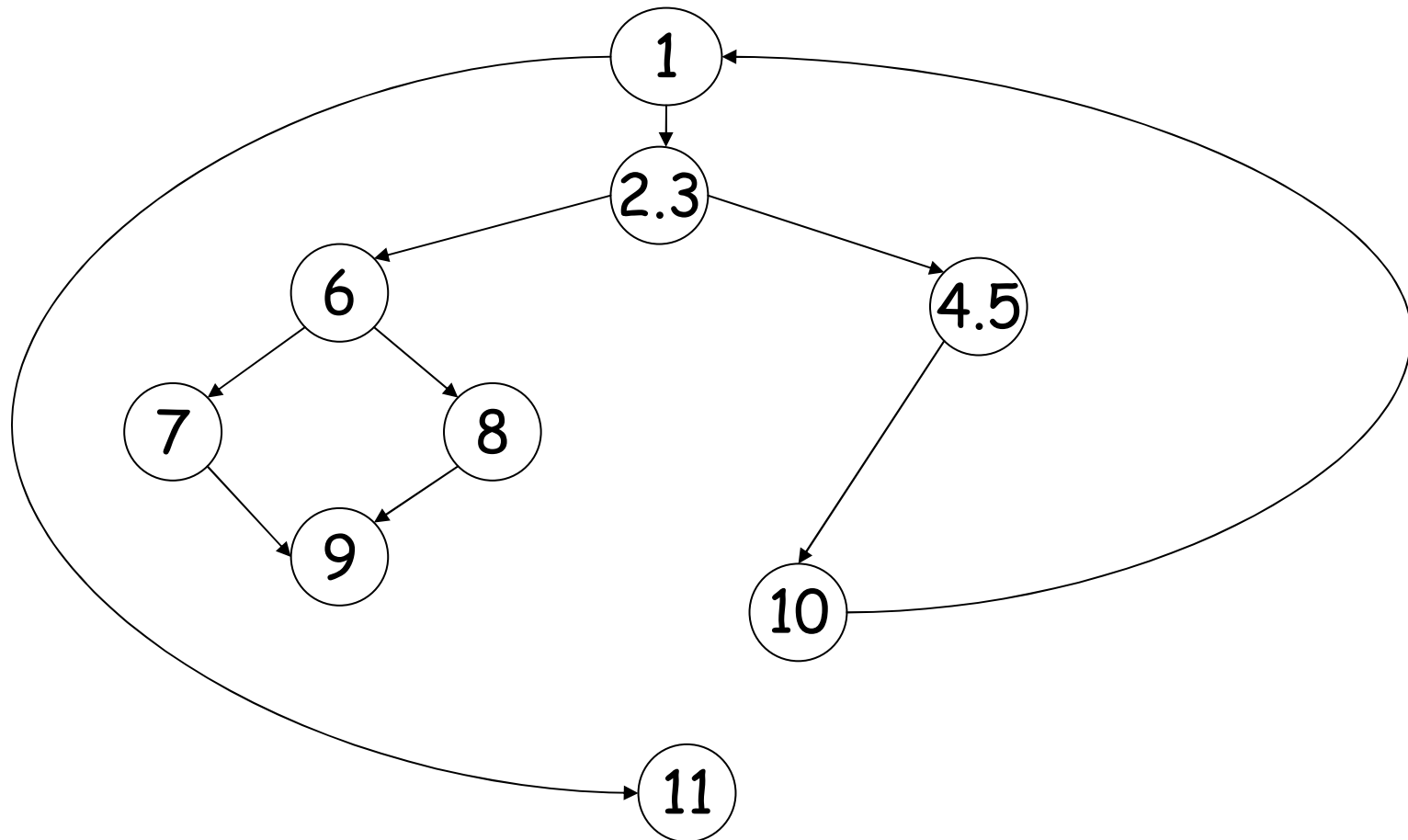
Complexidade ciclomática

- # Define o n° de caminhos independentes (conjunto básico) do programa e dá um limite superior para o n° de testes

Diagrama de fluxo



Grafo de fluxo



Caminhos linearmente independentes

- # Caminho 1: 1-11
- # Caminho 2: 1-2-3-4-5-10-1-11
- # Caminho 3: 1-2-3-6-8-9-10-1-11
- # Caminho 4: 1-2-3-6-7-9-10-1-11
- # Definem um conjunto básico
- # O conjunto básico não é único

Calculo da complexidade

N° de regiões dum grafo

$$\# V(G) = A - N + 2 = 11 - 9 + 2 = 4$$

$$\# V(G) = P + 1 = 3 + 1 = 4$$

Método

1. Usando o diagrama de fluxo ou o código como base, desenha-se o grafo de fluxo
2. Determina-se a complexidade ciclomática para saber o n° de caminhos independentes
3. Determina-se cada um dos caminhos independentes
4. Prepara-se testes que "forcem" cada um desses caminhos

Testes da estrutura de controlo

- # Teste da condição
- # Teste de fluxos de dados: caminhos de teste segundo a localização das definições e usos das variáveis
- # Testes de ciclos
 - ▣ Passar por alto
 - ▣ Passar 1 vez
 - ▣ Passar 2 vezes
 - ▣ Fazer m passos com $m < n$
 - ▣ Fazer $n-1$, n e $n+1$

Testes de "caixa preta"

- # Métodos baseados em grafos
- # Partição equivalente
 - Ex: teste de processamento dos dados tipo caracter
- # Análise de valores limites
- # Testes de comparação

Testes de ambientes, arquitecturas e aplicações

- # Testes de interfaces
- # Testes de arquitectura cliente/servidor
- # Testes da documentação e facilidades de ajuda
- # Testes de sistemas a tempo-real

Testes OO

- # Exactidão dos modelos AOO e DOO
- # Consistência dos modelos AOO e DOO
- # Testes de unidade \approx testes de classes
- # Testes de integração
 - Threads
 - Baseadas no uso (dependência de classes)
 - Baseadas no use-cases
- # Testes de validação baseados nas acções e saídas visíveis para o utilizador