SOFTWARE TESTING

Ian Sommerville, 8° edição – Capítulo 23

Aula de Luiz Eduardo Guarino de Vasconcelos

“A atividade de teste nunca termina. Apenas é transferida do projetista para seu cliente. O engenheiro de software pode realizar testes mais completos e portanto descobrir e corrigir o maior número de erros possíveis antes que os testes do cliente se iniciem.”

Objetivos

- Entender as técnicas de testes que são utilizadas para descobrir defeitos em programas
- Introduzir diretrizes para o teste de interface
- Compreender abordagens específicas para o teste de sistemas orientados a objetos
- Compreender os princípios da operação de apoio de ferramentas CASE para testes

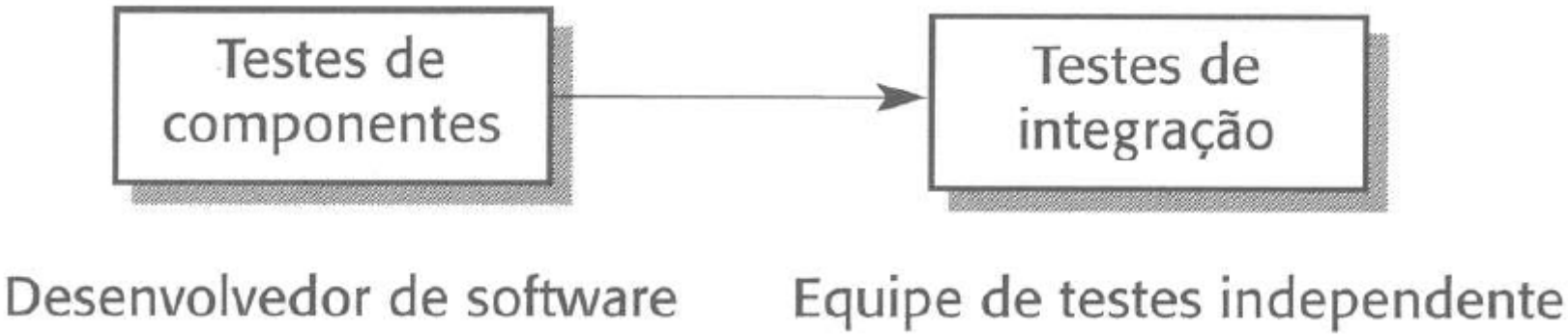
Tópicos abordados

- Testes de componentes e de integração
- Testes de defeitos
- Testes orientados a objetos
- Área de trabalho de teste

O processo de Testes

- Testes de componentes
  - Testes de componentes de programas individuais.
  - Usualmente os programadores assumem a responsabilidade pelo teste de seu código (exceto em caso de sistemas críticos).
  - Testes são derivados da experiência do desenvolvedor.
- Testes de integração
  - Testes de grupos de componentes integrados para formar subsistemas ou sistemas completos.
  - Uma equipe independente de teste faz o teste de integração.
  - Os testes são baseados em uma especificação do sistema.

Fases de Teste



Teste para detecção de defeitos

- O objetivo de testes para a detecção de defeitos é revelar defeitos nos programas.
- Um teste *bem sucedido é aquele que revela a* presença de um defeito (faz com que o programa se comporte de maneira anômala)
- Testes mostram a presença e não a ausência de defeitos.

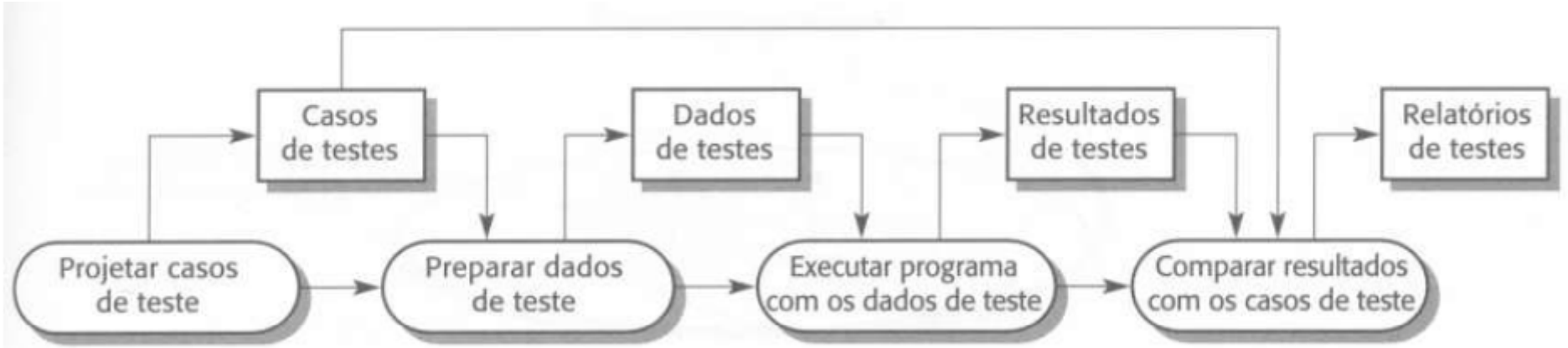
Prioridades de teste

- Somente o teste exaustivo pode mostrar que um programa é livre de defeitos. Contudo, o teste exaustivo é impossível
- Testes devem executar as capacidades do sistema em vez de seus componentes
- Testar capacidades antigas é mais importante que testar novas capacidades
- Testar situações típicas é mais importante que situações adversas (ex. casos de valor limite)

Dados de Teste e Casos de Teste

- Dados de teste: Entradas que foram criadas para testar o sistema
- Casos de teste: Entradas para testar o sistema e as saídas esperadas caso o sistema opere de acordo com sua especificação

Processo de teste para detecção de defeitos



Teste de Caixa Preta

- Uma abordagem de teste onde o programa é considerado uma '**caixa preta**'
- **Testes funcionais**
- Os casos de teste são derivados da especificação do sistema
- Planejamento de testes deve começar no início do processo de software

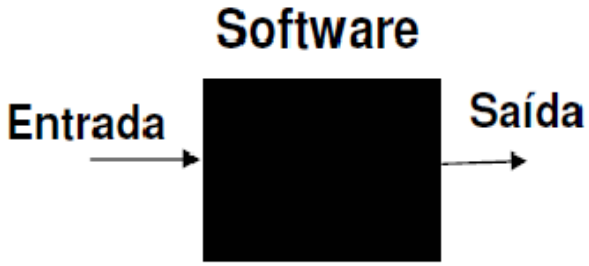
Teste de Caixa Preta

- Valores de borda

  - Utiliza valores extremos válidos (Ex: exorbitância, vetores, null, etc)

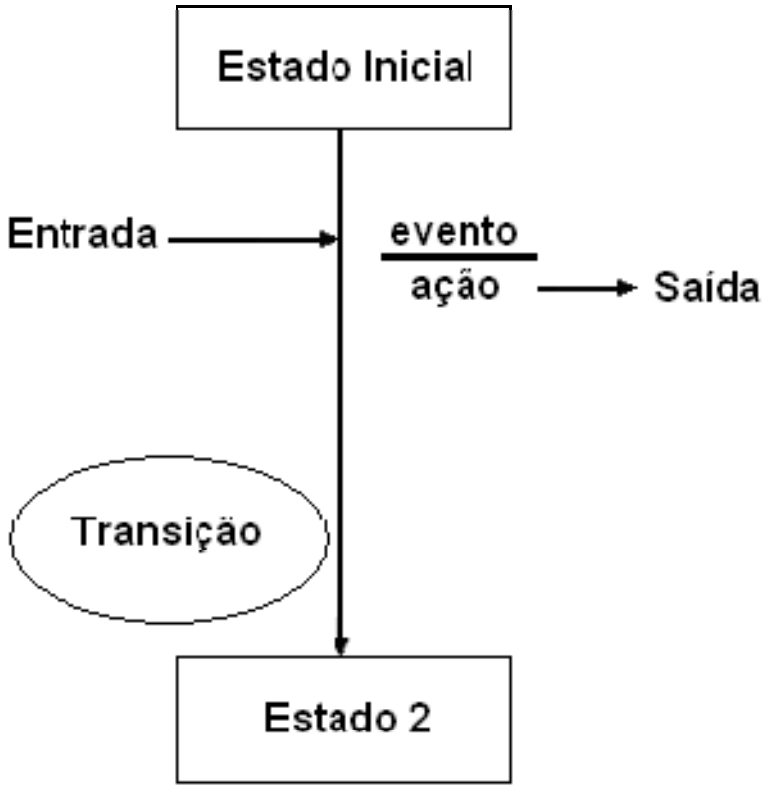
- Testes de Sintaxe

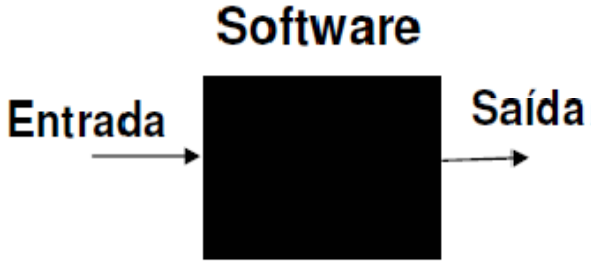
  - Abrange as entradas inválidas do sistema, com o objetivo de observar seu comportamento quando esses valores forem usados (Ex: Uso de 'floats' em campos inteiros)



Teste de Caixa Preta

- Testes de transição
  - Especifica o comportamento de um componente em uma transição de estado ocorrida por uma entrada ou um evento (Ex: assentos de um vôo, variáveis)





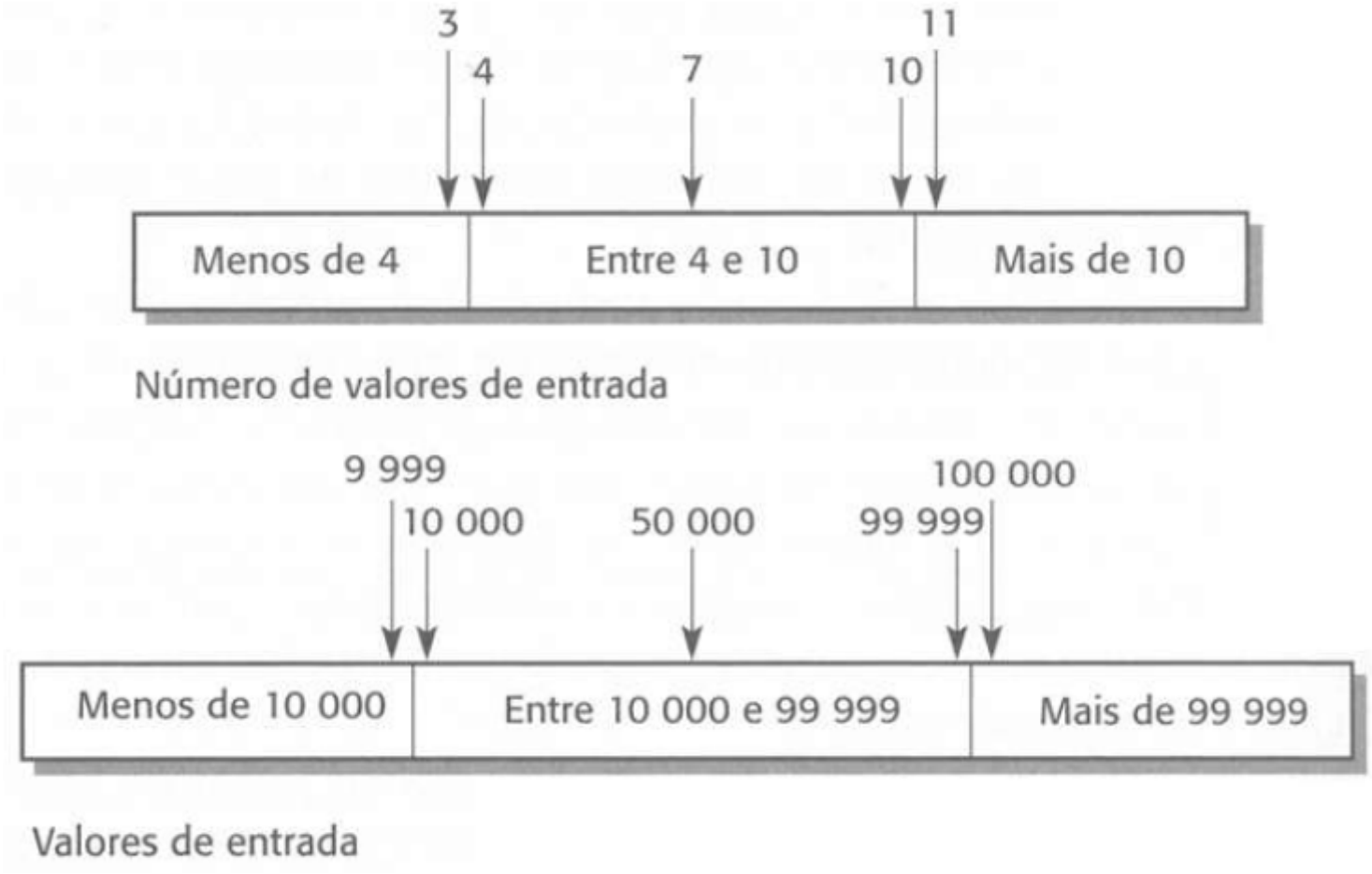
Particionamento de equivalência

- Dados de entrada e resultados de saída freqüentemente se encontram em diferentes classes onde todos os membros de uma classe são relacionados
- Cada uma destas classes é uma partição de equivalência onde o programa se comporta de uma maneira equivalente para cada membro da classe
- Casos de teste devem ser escolhidos a partir de cada partição

Particionamento de equivalência

- Particionar as entradas e saídas em 'conjuntos equivalentes'
  - Se a entrada é um número de 5 dígitos entre 10.000 e 99.999, as partições de equivalência são <10.000, 10.000-99.999, e >99.999
- Escolher casos de teste nos limites destes conjuntos
  - 00000, 09999, 10000, 99999, 100000

Particionamento de equivalência



Diretrizes de Teste (Sequência)

- Testar o software com seqüências que tenham somente um único valor
- Utilizar seqüências, de diferentes tamanhos, em diferentes testes
- Derivar testes de forma que o primeiro, o médio e o último elemento da seqüência sejam acessados
- Testar seqüências de tamanho zero e NULO

Rotina de Busca – Particionamento de Entrada

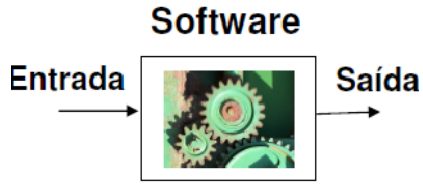
Vetor	Elemento
Valor único	Está na seqüência
Valor único	Não está na seqüência
Mais que 1 valor	Primeiro elemento está na seqüência
Mais que 1 valor	Último elemento está na seqüência
Mais que 1 valor	Elemento médio está na seqüência
Mais que 1 valor	Não está na seqüência

Seqüência de entradas (T)	Chave (key)	Saídas (Found, L)
17	17	Verdadeiro, 1
17	0	Falso, ??
17, 29, 21, 23	17	Verdadeiro, 1
41, 18, 9, 31, 30, 16, 45	45	Verdadeiro, 7
17, 18, 21, 23, 29, 41, 38	23	Verdadeiro, 4
21, 23, 29, 33, 38	25	Falso, ??

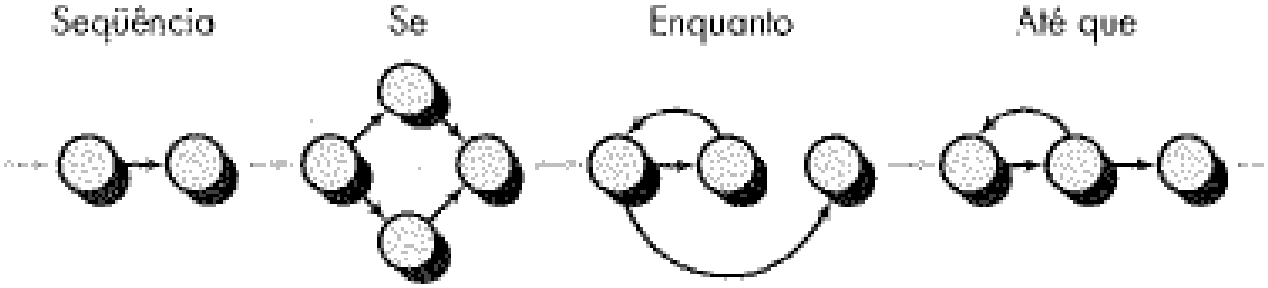
Testes de Estrutura

- Algumas vezes chamado de teste de '**caixa branca**'
- Os casos de teste são derivados de acordo com a **estrutura** do programa. O conhecimento do programa é utilizado para identificar casos de teste adicionais.
- Objetivo é exercitar todas as declarações do programa (e não todas as combinações de caminhos)

Testes de Estrutura



- Fluxo de dados
  - Põe à prova a lógica do programa.



- Teste de laços
  - Faz procedimentos para exercitar laços de vários graus de complexibilidade.

Testes de caminho

- O objetivo do teste de caminho é garantir que o conjunto de testes é tal que cada caminho do programa é executado pelo menos uma vez
- O ponto de partida para o teste de caminho é um grafo de fluxo do programa que mostra nós representando decisões de programa e em arcos representando o fluxo de controle
- Declarações condicionais são portanto nós no grafo de fluxo

Grafo de fluxo do programa

- Descreve o fluxo de controle do programa. Cada ramo em uma declaração condicional é mostrado como um caminho independente e os loops são indicados por setas fazendo o loop de volta para o nó de condição do loop
- Utilizado como uma base para o cálculo da complexidade ciclomática
- Complexidade ciclomática = Numero de arestas – Números de nós + 2

Complexidade ciclomática

- O número de testes de cada caminho independente (atravessa pelo menos um novo ramo) é igual a complexidade ciclomática
- O **número mínimo de casos de teste** para testar todos os caminhos é **igual** a **complexidade ciclomática**
- Útil se usado com cuidado. Não implica suficiência de testes
- Embora todos os caminhos sejam executados, não são executadas todas as combinações de caminhos

Grafo de fluxo para a busca binária

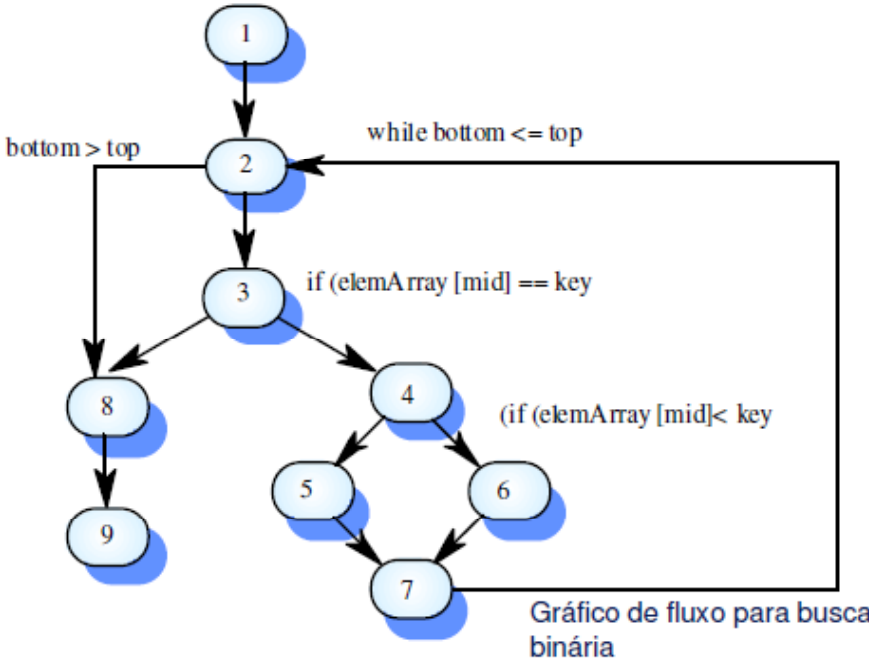


Gráfico de fluxo para busca binária

```
class BinSearch {

// This is an encapsulation of a binary search function that takes an array of
// ordered objects and a key and returns an object with 2 attributes namely
// index - the value of the array index
// found - a boolean indicating whether or not the key is in the array
// An object is returned because it is not possible in Java to pass basic types by
// reference to a function and so return two values
// the key is -1 if the element is not found

    public static void search ( int key, int [] elemArray, Result r )
    {
        int bottom = 0 ;
        int top = elemArray.length - 1 ;
        int mid ;
        r.found = false ; r.index = -1 ;
        while ( bottom <= top )
        {
            mid = (top + bottom) / 2 ;
            if (elemArray [mid] == key)
            {
                r.index = mid ;
                r.found = true ;
                return ;
            } // if part
            else
            {
                if (elemArray [mid] < key)
                    bottom = mid + 1 ;
                else
                    top = mid - 1 ;
            }
        } //while loop
    } // search
} //BinSearch
```

Busca Binária (Java)

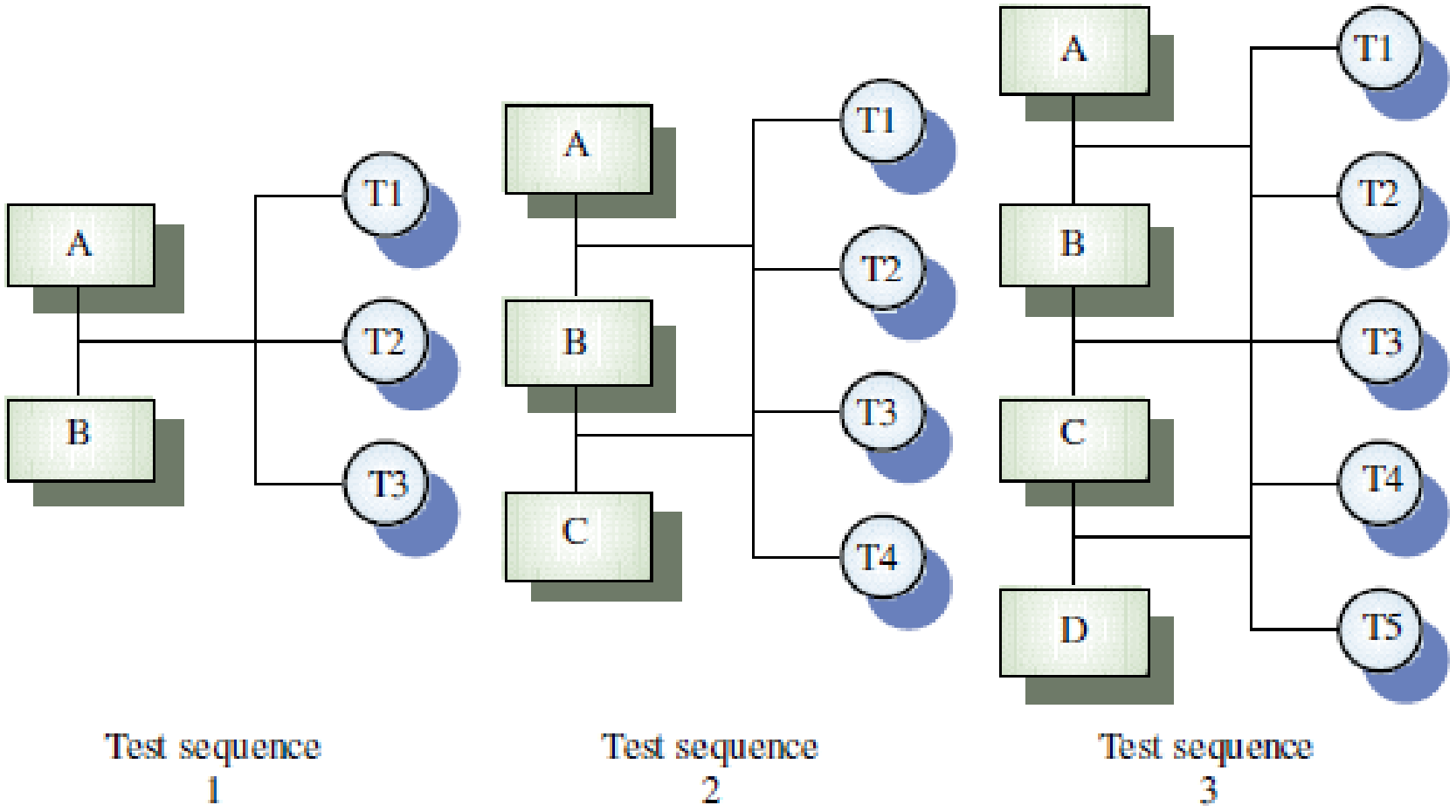
Caminhos independentes

- 1, 2, 3, 8, 9
- 1, 2, 3, 4, 6, 7, 2
- 1, 2, 3, 4, 5, 7, 2
- 1, 2, 3, 4, 6, 7, 2, 8, 9
- Casos de teste devem ser derivados de forma que todos estes caminhos sejam executados
- Um analisador dinâmico de programa pode ser utilizado para verificar se todos os caminhos foram executados

Teste de Integração

- Testa sistemas completos ou subsistemas compostos de componentes integrados
- Testes de integração devem ser 'caixa preta' com testes derivados da especificação do sistema
- Principal dificuldade é a localização de erros
- Testes de integração incremental reduz este problema

Teste de Integração Incremental



Abordagens para Teste de Integração

- Teste top-down
  - Inicia com componentes de alto nível integrando de maneira top-down. Substitui componentes individuais por stubs quando apropriado
- Teste Bottom-up
  - Integra componentes individuais em níveis até o sistema completo estar criado.
- Na prática, a maioria das integrações envolvem uma combinação destas estratégias

Abordagens de Teste

- Validação de arquitetura
  - Teste top-down é melhor em descobrir erros na arquitetura do sistema
- Demonstração do sistema
  - Teste top-down permite uma demonstração limitada no estágio inicial de desenvolvimento
- Implementação de teste
  - Freqüentemente mais fácil com o teste bottom-up
- Observação de teste
  - Problemas com ambas abordagens. Código extra pode ser necessário para observar os testes

Teste de Interface

- Ocorrem quando módulos ou subsistemas são integrados para criar sistemas maiores
- Objetivo é detectar defeitos devido a erros de interface ou suposições inválidas sobre as interfaces
- Particularmente importante para o desenvolvimento orientado a objetos, já que os objetos são definidos por suas interfaces

Teste de estresse

- Executar o sistema além da carga máxima projetada. Os testes de estresse freqüentemente provocam a revelação de defeitos do sistema
- Durante os testes de estresse o sistema não deve falhar de maneira catastrófica. O sistema não deve apresentar corrupção de dados ou a perda inesperada de serviços
- Particularmente importante em sistemas distribuídos que podem apresentar uma grande degradação à medida que a rede se torna sobrecarregada

Testes orientado a objetos

- Os componentes a serem testados são classes instanciadas como objetos
- Como objetos têm uma granularidade maior que funções individuais, os testes de caixa branca devem ser estendidos
- Não há um nível superior óbvio para o sistema para a integração e teste top-down

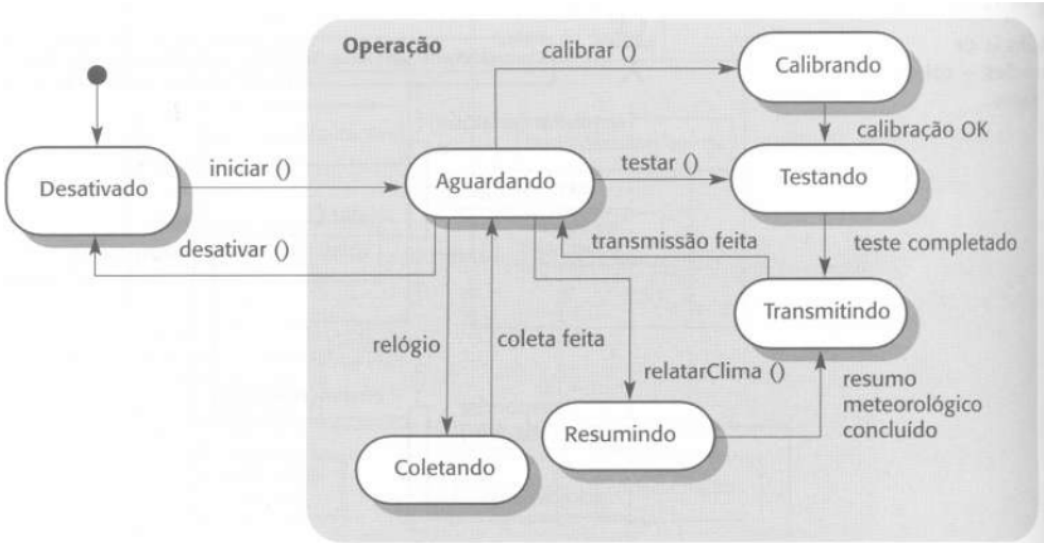
Níveis de Teste

- Testar operações individuais associadas com objetos
- Testar classes de objetos individuais
- Testar agrupamentos de objetos relacionados
- Testar o sistema orientado a objetos

Testes de Classes de Objetos

- Cobertura completa de testes de uma classe envolve
- Testar todas as operações associadas com um objeto
- Estabelecimento e interrogação de todos os atributos do objeto
- Exercício do objeto em todos os estados possíveis
- A herança torna mais difícil projetar testes de classes de objetos já que a informação a ser testada não é localizada

Interface do objeto Estação Meteorológica



- São necessários casos de teste para todas as operações
- Utilizar um modelo de estado para identificar seqüências de transições de estado para teste
- Exemplos de seqüências a serem testadas
- Desativado → Aguardando → Desativado
- Aguardando → Calibrando → Testando → Transmitindo → Aguardando
- Aguardando → Coletando → Aguardando →Resumindo → Transmitindo → Aguardando

Outros Testes

- Teste Alpha
- Teste Beta
- Teste de Regressão
  - Garante funcionamento pleno
- Teste de Aceitação
  - Grupo restrito de usuários finais simulando rotinas

Profissão: Testador

- É uma certificação relativamente nova na área de TI
- Em outros países já é bem difundida
  - Na índia são 10 mil testadores
- Compradores, especialmente estrangeiros, exigem que a solução desenvolvida seja certificada
- Salário Médio
  - Entre R$ 1.900,00 e R$ 5.000,00

Certificação em Testes de Software

Existem várias entidades que emitem a certificação em testes de software:

- **QAI - Quality Assurance Institute**

A prova está dividida em 4 partes sendo duas de múltipla escolha e duas partes Dissertativas. Seu custo é de 350 dólares.

- **CMST - Certified Manager of Software Testing**

A prova requer conhecimento prático. Está dividida em 4 partes e as respostas tem de ser construídas. Seu custo é de 600 dólares.

- **ISTQB (International Software Testing Qualifications Board)**

Com duração de uma hora, a prova contém 40 questões de múltipla escolha. Para ser aprovado, necessita-se de no mínimo 65 % de acertos. Tem custo de 350 reais.

- **CTM – Certified Test Manager**

A prova tem custo de 120 dólares + um treinamento obrigatório. O número de questões e o tempo de duração variam. Requer mínimo de 80 % de acertos para aprovação

Certificação em Testes de Software

Existem várias entidades que emitem a certificação em testes de software:

- **ALATS (Associação Latino-Americana de Teste de Software)**

A prova contém 100 questões de múltipla escolha e tem duração de 3 horas.
O preço é de 300 reais e requer no mínimo 75 % de acertos

http://www.alats.org.br.

- **IBQTS (Instituto Brasileiro de Qualidade em Testes de Software)**

Contém 50 questões de múltipla escolha e 3 horas de duração.. Seu preço é de 300 reais e requer no mínimo 70% de acertos

http://www.ibqts.com.br.

“A atividade de teste nunca termina. Apenas é transferida do projetista para seu cliente. O engenheiro de software pode realizar testes mais completos e portanto descobrir e corrigir o maior número de erros possíveis antes que os testes do cliente se iniciem.”

Pontos-chave

- É mais importante testar as partes do sistema mais comumente utilizadas do que as partes que são exercitadas raramente.
- Partição de equivalência é uma maneira de derivar casos de teste. Partições são conjuntos de dados onde o programa deve se comportar de maneira equivalente.
- Teste de caixa preta é baseado na especificação do sistema. Não precisa analisar o código fonte.
- Teste estrutural baseia-se na análise do programa para determinar os caminhos a serem executados e a seleção de casos de teste.

Pontos-chave

- Os testes de integração testes de integração se concentram no teste das interações entre os componentes.
- Para testar as classes de objetos testar as classes de objetos, deve-se testar todas as operações, atributos e estados.