Teste de Software

Escopo de um Teste

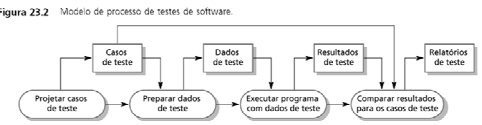
- Teste de componentes ou de unidade
  - Teste de componentes individuais de programa
  - Geralmente é de responsabilidade do desenvolvedor do componente (exceto algumas para sistemas críticos)
  - Os testes são derivados da experiência do desenvolvedor
- Teste de sistema
  - Teste de grupos de componentes integrados para criar um sistema ou um subsistema
  - A resposabilidade é, frequentemente, de uma equipe independente de teste
  - Baseados em uma especificação de sistema

Metas do Processo de Teste

- Teste de validação
  - Demonstra ao desenvolvedor e ao cliente que o software atende aos seus requisitos
- Teste de defeitos
  - Utilizado para descobrir defeitos no software
  - Um teste bem sucedido é aquele que faz o sistema se comportar incorretamente
  - Mostram a presença de defeitos, não sua ausência

O Processo de Teste de Software

Figura 23.2 Modelo de processo de testes de software.



Políticas de Teste

- Somente testes exaustivos podem mostrar que um programa está livre de defeitos
  - Inexequível na prática!
- As políticas de teste definem a abordagem a ser usada na seleção de testes de sistema:
  - Todas as funções acessadas por meio de menus devem ser testadas;
  - As combinações de funções acessadas por meio dos mesmos menus devem ser testadas;
  - Todas as funções devem ser testadas com entradas corretas e incorretas
    - Máximos, mínimos e valores intermediários

Teste de Sistema

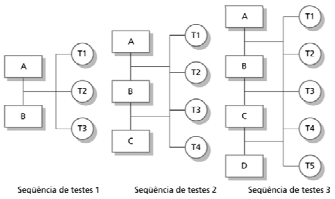
- Envolve a integração de dois ou mais componentes para criar um sistema ou subsistema.
- Pode envolver o teste de um incremento para ser entregue ao cliente.
- Duas fases:
  - Teste de integração – a equipe de teste tem acesso ao código fonte do sistema e este é testado à medida que os componentes são integrados.
  - Teste de *releases* – a equipe de teste testa o sistema completo a ser entregue como uma caixa-preta.

Teste de Integração

- Teste do sistema do ponto de vista das interações entre seus componentes
- Integração *top-down*
  - Desenvolver o esqueleto do sistema e preenchê-lo com componentes
- Integração *bottom-up*
  - Integrar componentes de infra-estrutura e, em seguida, adicionar componentes funcionais
- Para simplificar a localização de erros, os sub-sistemas devem ser integrados incrementalmente

Teste de Integração Incremental

Figura 23.3
Testes de integração incremental.

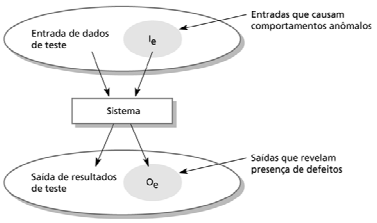


Teste de Releases

- É o processo de teste de um release de sistema que será distribuído aos clientes
- A meta primária é aumentar a confiança do fornecedor de que o sistema atende aos requisitos
- Teste de releases é, geralmente, um teste caixa-preta ou funcional
  - É baseado somente na especificação de sistema;
  - Os testadores não têm conhecimento da implementação do sistema.

Teste Caixa-Preta

Figura 23.4
Teste caixa-preta.



Diretrizes de Teste

- Diretrizes são recomendações para a equipe de teste
  - Variam de projeto para projeto
- Auxilia na escolha dos testes que revelarão defeitos no sistema
- Exs.:
  - Escolher entradas que forcem o sistema a gerar todas as mensagens de erro
  - Projetar entradas que causem *overflow* dos *buffers*
  - Repetir uma entrada ou série de entradas várias vezes
  - Forçar resultados de cálculo a serem muito grandes ou muito pequenos

Diretrizes de Teste

- Diretrizes são recomendações para a equipe de

Como vocês elaboraram os testes em seu projeto?

Testaram para um grande número de cenários?

Testaram cenários de erro?

Testaram condições-limite do sistema?

- Forçar resultados de cálculo a serem muito grandes ou muito pequenos

Cenário de Teste

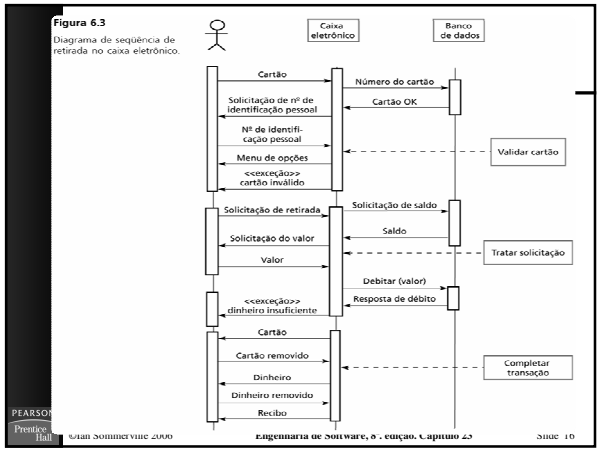
Uma estudante na Escócia, que estuda história Americana, recebeu a tarefa para escrever um artig
sobre 'Mentalidade de fronteira no Oeste Americano de 1840 a 1880'. Para fazer isto, ela necessit
encontrar fontes de uma variedade de bibliotecas. Ela entra no sistema LIBSYS e usa o recurso de busc
para descobrir se ela pode acessar os documentos originais da época. Ela descobre fontes em vári
bibliotecas de universidades dos EUA e baixa cópias destes documentos. Contudo, para um document
ela precisa ter a confirmação de sua universidade de que ela é uma aluna legítima e que o uso d
documento é para fins não comerciais. A estudante então usa o recurso presente no LIBSYS que pode
solicitar tal permissão, e registra a sua solicitação. Se for concedida, o documento será baixado para o
servidor registrado da biblioteca e impresso. Ela recebe uma mensagem do LIBSYS contando a ela qu
receberá uma mensagem de e-mail quando o documento impresso estiver disponível.

Exemplos Testes de Sistema

1.

Casos de Uso

- Podem servir de base para derivar os testes de um sistema
- Ajudam a identificar as funcionalidades a ser testadas e a projetar os casos de teste necessários
- Diagramas de sequência associados fornecem informações adicionais
  - Entradas e saídas
  - Sequência de operações



Teste de Desempenho

- Normalmente caixa branca
- Visa identificar *bottlenecks* de desempenho
- Não tem o objetivo de identificar *bugs*
  - O sistema já está estável o suficiente
- Usa-se uma série de testes onde a carga é constantemente aumentada
  - Até que o desempenho do sistema se torne inaceitável
  - Neste ponto, tenta-se identificar ineficiências

Exemplo: Aplicação *Web*

- Parâmetros a ser medidos:
  - Carga esperada em termos de usuários concorrentes ou conexões HTTP
  - Tempo de resposta aceitável
- *Bottlenecks* podem estar em vários níveis:
  - Aplicação, Servidor Web, Banco de dados, Sistema operacional, Rede, Hardware
- Encontrados os *bottlenecks*, busca-se estratégias para mitigar o problema

Fonte: http://agiletesting.blogspot.com/2005/02/performance-vs-load-vs-stress-testing.html

Teste de Carga

- Parte dos testes de desempenho
- O sistema se comporta adequadamente quando sob a carga normal esperada?
  - E quando a carga chega ao limite o normal?
- Usa-se abordagens extremas
  - Mas o objetivo não é quebrar o sistema!
  - Exs.:
    - Abrir um arquivo muito grande em um editor de texto
    - Aplicar maior carga de usuários esperada a um RPG *on-line*

Teste de Estresse

- Exercita o sistema além de sua capacidade máxima de prover serviço
- Normalmente resulta na ocorrência de defeitos
- Serve para verificar se:
  - O sistema falha catastroficamente?
  - O sistema se recupera de forma apropriada?
- *"Where performance testing demands a controlled environment and repeatable measurements, stress testing joyfully induces chaos and unpredictability."*

Fonte: http://agiletesting.blogspot.com/2005/02/performance-vs-load-vs-stress-testing.html

Teste de Componentes

- Teste de componente ou unitário é o processo de teste de componentes individuais isolados.
- É um processo de teste de defeitos.
- Os componentes podem ser:
  - Funções individuais ou métodos de um objeto;
  - Classes com vários atributos e métodos;
  - Componentes compostos com interfaces definidas usadas para acessar sua funcionalidade.

Teste de Classes

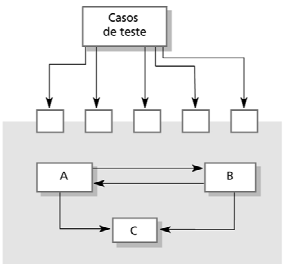
- A abrangência do teste completo de uma classe envolve
  - Teste de todas as operações associadas com um objeto;
  - Atribuir a e obter os valores de todos os atributos;
  - Exercício do objeto em todos os estados possíveis.
- A herança torna mais difícil o projeto de testes de classes

Teste de Interfaces

- Os objetivos são detectar defeitos devido a erros de interface ou suposições inválidas sobre interfaces
  - Normalmente, pré-condições
  - Invocações implícitas (padrão *Observer*)
- É particularmente importante para o desenvolvimento orientado a objetos

Teste de Interfaces

Figura 23.7
Teste de interface.



Erros de Interface

- Mau uso de interface
  - Ex. Parâmetros em ordem errada
  - Normalmente detectados pelo compilador
- Mau entendimento de interface
  - Ex. Suposições incorretas sobre o comportamento do componente chamado
- Ordem de invocação
  - Ex. Operação A() precisa ser invocada antes da operação B()
- Erros de temporização
  - Os componentes chamado e chamador operam em velocidades diferentes

Projeto de casos de teste

- Envolve o projeto de casos de teste (entradas e saídas) usados para testar o sistema.
- A meta do projeto de casos de teste é criar um conjunto de testes que sejam eficazes em validação e teste de defeitos.
- Abordagens de projeto:
  - Teste baseado em requisitos;
  - Teste de partições;
  - Teste de caminhos

Teste Baseado em Requisitos

- Requisitos devem ser testáveis.
- Para cada requisito, deriva-se um conjunto de testes
  - Foco em testes de validação

Requisitos do LIBSYS

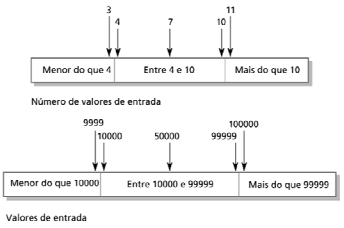
Testes do LIBSYS

Teste de Partições

- Dados de entrada e resultados de saída caem freqüentemente em classes diferentes
  - Todos os membros de uma classe são relacionados.
- Cada classe é uma partição de equivalência, onde o programa se comporta da mesma forma para cada membro da classe
- Teste de **defeito**!
- Casos de teste devem ser escolhidos a partir de cada partição.

Partições de Equivalência

Figura 23.9
Partições de equivalência.



Especificação de uma Rotina de Busca

Figura 23.10
Especificação de uma rotina de busca.

```
procedimento Search (Key : ELEM ; T: SEQ of ELEM ;
Found : in out BOOLEAN; L: in out ELEM_INDEX) ;
Precondição
    – a seqüência tem pelo menos um elemento
    T'FIRST <= T'LAST
Pós-condição
    – o elemento é encontrado e referenciado por L
    ( Found and T (L) = Key)
ou
    – o elemento não está na seqüência
    ( not Found and
    not (exists i, T'FIRST >= i <= T'LAST, T (i) = Key))
```


Rotina de busca – partições de entrada

- Entradas que estão de acordo com as pré-condições.
- Entradas onde uma pré-condição não é atendida.
- Entradas onde o elemento key é um membro da seqüência.
- Entradas onde o elemento key não é um membro da seqüência.

Diretrizes de teste (seqüências)

- Testar o software com seqüências que têm apenas um valor único.
- Usar seqüências de tamanhos diferentes em testes diferentes.
- Derivar testes de tal modo que o primeiro, o médio e o último elementos da seqüência sejam acesssados.
- Testar com seqüências de comprimento zero.

Rotina de Busca – Partições de Entrada

Tabela 23.1 Partições equivalentes para rotina de busca

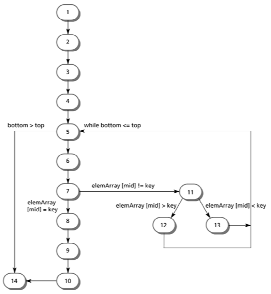
Seqüência	Elemento	
Valor único	Em seqüência	
Valor único	Fora da seqüência	
Mais de 1 valor	Primeiro elemento na seqüência	
Mais de 1 valor	Último elemento na seqüência	
Mais de 1 valor	Elemento médio na seqüência	
Mais de 1 valor	Fora da seqüência	
Seqüência de entradas (T)	**Chave (Key)**	**Saídas (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, ??

Teste de Caminhos

- Visa assegurar que cada caminho do programa é executado pelo menos uma vez
- O ponto de partida é o grafo de fluxo de controle do programa
- Declarações com condições são nós desse grafo
  - Diferentes decisões representam arestas

Grafo de Fluxo de Controle da Rotina de Busca

Figura 23.14
Fluxograma para a rotina de busca.



Caminhos Independentes

- 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 14
- 1, 2, 3, 4, 5, 14
- 1, 2, 3, 4, 5, 6, 7, 11, 12, 5, …
- 1, 2, 3, 4, 6, 7, 2, 11, 13, 5, …
- Casos de teste devem ser derivados de tal modo que todos os caminhos sejam executados
  - Analisadores estáticos podem ajudar a descobrir os caminhos
- Um analisador dinâmico de programa pode ser usado para verificar se os caminhos foram executados