Teste de Software

Motivação

- Ocorrência de falhas humanas no processo de desenvolvimento de software é considerável
- Processo de testes é indispensável na garantia de qualidade de software
- Custos associados às falhas de software justificam um processo de testes cuidadoso e bem planejado

Falha, Falta e Erro

- Falha
  - Incapacidade do software de realizar a função requisitada (aspecto externo)
  - Exemplo
    - Terminação anormal, restrição temporal violada

Falha, Falta e Erro

- Falta
  - Causa de uma falha
  - Exemplo
    - Código incorreto ou faltando

Falha, Falta e Erro

- Erro
  - Estado intermediário (instabilidade)
  - Provém de uma falta
  - Pode resultar em falha, se propagado até a saída

Falha, Falta e Erro

Noção de confiabilidade

- Algumas faltas escaparão inevitavelmente
  - Tanto dos testes
  - Quanto da depuração
- Falta pode ser mais ou menos perturbadora
  - Dependendo do que se trate e em qual freqüência irá surgir para o usuário final

Noção de confiabilidade

- Assim, precisamos de uma referência para decidir
  - Quando liberar ou não sistema para uso
- Confiabilidade de software
  - É uma estimativa probabilística
  - Mede a freqüência com que um software irá executar sem falha
    - Em dado ambiente
    - E por determinado período de tempo
- Assim, entradas para testes devem se aproximar do ambiente do usuário final

Dados e Casos de Teste

- Dados de Teste
  - Entradas selecionadas para testar o software
- Casos de Teste
  - Dados de teste, bem como saídas esperadas de acordo com a especificação (Veredicto)
  - Cenários específicos de execução

Eficácia de testes

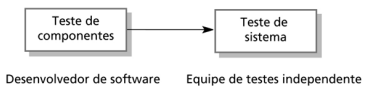
- A atividade de teste é o processo de executar um programa com a intenção de descobrir um erro
- Um bom caso de teste é aquele que apresenta uma elevada probabilidade de revelar um erro ainda não descoberto
- Um teste bem sucedido é aquele que revela um erro ainda não descoberto

O processo de teste

- Teste de componentes
  - 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 de uma equipe independente de teste;
  - Os testes são baseados em uma especificação de sistema.

Fases de teste

Figura 23.1
Fases de teste.



Teste de defeitos

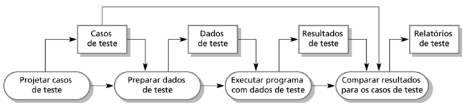
- A meta do teste de defeitos é descobrir defeitos em programas.
- Um teste de defeitos bem sucedido é aquele que faz um programa se comportar de uma maneira anômala.
- Os testes mostram a presença e não a ausência de defeitos.

Metas do processo de teste

- Teste de validação
  - Utilizado para demonstrar ao desenvolvedor e ao cliente do sistema que o software atende aos seus requisitos.
  - Um teste bem sucedido mostra que o sistema opera conforme pretendido.
- Teste de defeitos
  - Utilizado para descobrir faltas ou defeitos no software nos locais em que o comportamento não está correto ou não está em conformidade com a sua especificação;
  - Um teste bem sucedido é aquele que faz o sistema executar incorretamente e, assim, expor um defeito no sistema.

O processo de testes 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. Contudo, testes exaustivos são impossíveis.
- 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;
  - Onde as entradas de usuário são fornecidas, todas as funções devem ser testadas com entradas corretas e incorretas.

Teste de sistema

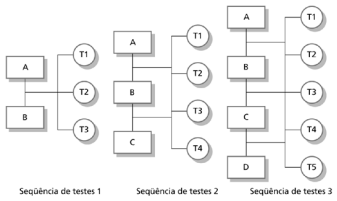
- Envolve a integração de dois ou mais componentes para criar um sistema ou subsistema.
- Pode envolver o teste de um incremento para ser entregue ao cliente.
- Duas fases:
  - Teste de integração – a equipe de teste tem acesso ao código fonte do sistema e o sistema é testado à medida que os componentes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

Teste de integração

- Envolve a construção de um sistema a partir de seus compontes e o teste do sistema resultante dos problemas ocorridos nas interações entre 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 sistemas devem ser integrados incrementalmente.

Teste de integração incremental

Figura 23.3
Testes de integração incremental.



Abordagens de teste

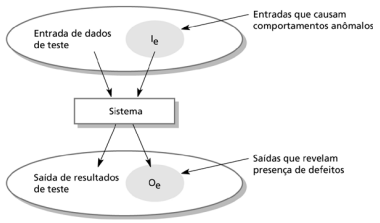
- Validação de arquitetura
  - O teste de integração *top-down* é melhor para descobrir erros na arquitetura do sistema.
- Demonstração de sistema
  - O teste de integração *top-down* permite uma demonstração limitada no estágio inicial do desenvolvimento.
- Implementação de teste
  - Freqüentemente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

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 seus 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 para auxiliá-los a escolher os testes que revelarão defeitos no sistema
  - Escolher entradas que forcem o sistema a gerar todas as mensagens de erro;
  - Projetar entradas que causem overflow dos buffers;
  - Repetir a mesma entrada ou série de entradas várias vezes;
  - Forçar a geração de saídas inválidas;
  - 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 artigo sobre 'Mentalidade de fronteira no Oeste Americano de 1840 a 1880'. Para fazer isto, ela necessita encontrar fontes de uma variedade de bibliotecas. Ela entra no sistema LIBSYS e usa o recurso de busca para descobrir se ela pode acessar os documentos originais da época. Ela descobre fontes em várias bibliotecas de universidades dos EUA e baixa cópias destes documentos. Contudo, para um documento, ela precisa ter a confirmação de sua universidade de que ela é uma aluna legítima e que o uso do 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 que receberá uma mensagem de e-mail quando o documento impresso estiver disponível.

Testes de sistema

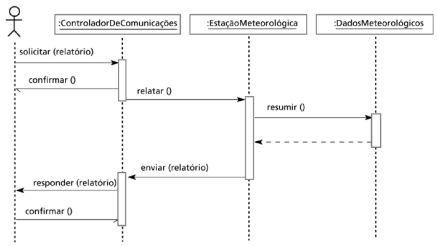
1. Testar o mecanismo de *login* usando *logins* corretos e incorretos para verificar se os usuários válidos são aceitos e os usuários inválidos são rejeitados.
2. Testar o recurso de busca, usando consultas contra fontes conhecidas para verificar se o mecanismo de busca está realmente encontrando documentos.
3. Testar o recurso de apresentação de sistema para verificar se as informações sobre documentos são apresentadas apropriadamente.
4. Testar o mecanismo para solicitar permissão para baixar documentos.
5. Testar a resposta de *e-mail* que indica que o documento baixado está disponível.

Casos de uso

- Casos de uso podem ser uma base para derivar os testes de um sistema. Eles ajudam a identificar as operações a serem testadas e a projetar os casos de teste necessários.
- A partir de um diagrama de seqüência associado, as entradas e saídas a serem criadas para os testes podem ser identificadas.

Diagrama de seqüência de coleta de dados meteorológicos

Figura 23.5 Diagrama de seqüência de coleta de dados meteorológicos.



Teste de desempenho

- Parte do teste de releases pode envolver teste de propriedades emergentes de um sistema, tais como desempenho e confiabilidade.
- Testes de desempenho envolve, geralmente, o planejamento de uma série de testes onde a carga é constantemente aumentada até que o desempenho do sistema se torne inaceitável.
  - Transções em BD
  - Terminais

Teste de estresse

- São exercícios do sistema além de sua carga máxima de projeto. O estresse de um sistema causa, freqüentemente, o surgimento de defeitos.
- O estresse de sistema testa o comportamento de falha, pois os sistemas não devem falhar catastroficamente. O teste de estresse verifica uma perda inaceitável de serviço ou de dados.
- O teste de estresse é particularmente relevante para sistemas distribuídos que podem exibir degradação severa quando uma rede se torna sobrecarregada.

Teste de estresse

- Exemplos
  - Pouca memória ou área em disco, alta competição por recursos compartilhados (ex: vários acessos/ transações no BD ou rede)
  - Exemplo: pode-se desejar saber se um sistema de transações bancárias suporta uma carga de mais de 100 transações por segundo ou se um sistema operacional pode manipular mais de 200 terminais remotos

Tipos de teste

- Teste de segurança e controle de acesso
  - Verifica se todos os mecanismos de proteção de acesso estão funcionando satisfatoriamente
- Teste de integridade de dados
  - Verifica a corretude dos métodos de acesso à base de dados e a garantia das informações armazenadas

Tipos de teste

- Teste de configuração ou portabilidade
  - Verifica o funcionamento adequado do sistema em diferentes configurações de hardware/software
  - O que testar
    - Compatibilidade do software/hardware
    - Configuração do servidor
    - Tipos de conexões com a Internet
    - Compatibilidade com o browser

Tipos de teste

- Teste de instalação e desinstalação
  - Verifica a correta instalação e desinstalação do sistema para diferentes plataformas de hardware/ software e opções de instalação
  - O que testar
    - Compatibilidade do hardware e software
    - Funcionalidade do instalador/desinstalador sob múltiplas opções/condições de instalação
    - GUI do programa instalador/desinstalador

Tipos de teste

- Teste de documentação
  - Verifica se a documentação corresponde à informação correta e apropriada:
    - online
    - escrita
    - help sensível ao contexto
- Teste de ciclo de negócios
  - Garante que o sistema funciona adequadamente durante um ciclo de atividades relativas ao negócio

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 de objeto com vários atributos e métodos;
  - Componentes compostos com interfaces definidas usadas para acessar sua funcionalidade.

Teste de classe de objeto

- A abrangência do teste completo de uma classe envolve
  - Teste de todas as operações associadas com um objeto;
  - Atribuir e interrogar todos os atributos de objeto;
  - Exercício do objeto em todos os estados possíveis.
- A herança torna mais difícil o projeto de testes de classe de objeto quando as informações a serem testadas não são localizadas.

Interface de objeto da estação meteorológica

Figura 23.6
Interface de objeto da estação meteorológica.

Teste da estação meteorológica

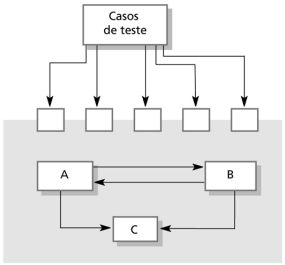
- Necessidade de definir casos de teste para o **relatarClima, calibrar, testar, iniciar, desativar**.
- Usando um modelo de estado, identificar as seqüências de transições de estado a serem testadas e as seqüências de eventos que causam essas transições.
- Por exemplo:
  - Aguardando → Calibrando → Testando → Transmitindo → Aguardando

Teste de interfaces

- Os objetivos são detectar defeitos devido a erros de interface ou suposições inválidas sobre interfaces.
- É particularmente importante para o desenvolvimento orientado a objetos quando os objetos são definidos pelas suas interfaces.

Teste de interfaces

Figura 23.7
Teste de interface.



Tipos de interfaces

- Interfaces de parâmetros
  - Os dados são passados de um procedimento para outro.
- Interfaces de memória compartilhada
  - Um bloco de memória é compartilhado entre procedimentos ou funções.
- Interfaces de procedimentos
  - Um subsistema engloba um conjunto de procedimentos para serem chamados por outros subsistemas.
- Interfaces de passagem de mensagem
  - Os subsistemas solicitam serviços de outros subsistemas.

Erros de interface

- Mau uso de interface
  - Um componente chamador chama um outro componente e faz mau uso de sua interface, por exemplo, parâmetros em ordem errada.
- Mau entendimento de interface
  - Um componente chamador considera suposições sobre o comportamento do componente chamado que estão incorretas.
- Erros de timing
  - Os componentes chamado e chamador operam em velocidades diferentes, e informações desatualizadas são acessadas.

Diretrizes de teste de interfaces

- Projetar testes de tal modo que os parâmetros para um procedimento chamado estejam nos limites extremos de suas faixas.
- Testar sempre os parâmetros de ponteiro com ponteiros nulos.
- Projetar testes que causem a falha do componente.
- Usar teste de estresse em sistemas de passagem de mensagem.
- Em sistemas de memória compartilhada, variar a ordem na qual os componentes são ativados.

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 estrutural.

Teste baseado em requisitos

- Um princípio geral de engenharia de requisitos é que os requisitos devem ser testáveis.
- O teste baseado em requisitos é uma técnica de teste de validação onde você considera cada requisito e deriva um conjunto de testes para esse requisito.

Requisitos do LIBSYS

O usuário será capaz de procurar o conjunto todo inicial da base de dados ou selecionar um subconjunto dele.

O sistema proverá visualizadores apropriados para que o usuário possa ler documentos no depósito de documentos.

A cada solicitação será alocado um identificador único (ORDER_ID) com o qual o usuário será capaz de copiar para a área de armazenamento permanente.

Testes do LIBSYS

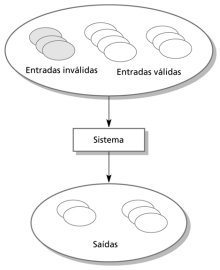
- Iniciar as buscas de usuário para os itens que se conhece a sua presença e para aqueles que não se conhece a sua presença, onde o conjunto de bases de dados inclui uma base de dados.
- Iniciar as buscas de usuário para os itens que se conhece a sua presença e para aqueles que não se conhece a sua presença, onde o conjunto de bases de dados inclui duas bases de dados.
- Iniciar as buscas de usuário para os itens que se conhece a sua presença e para aqueles que não se conhece a sua presença, onde o conjunto de bases de dados inclui mais de uma base de dados.
- Selecionar uma base de dados do conjunto de base de dados e iniciar as buscas de usuário para os itens que se conhece a sua presença e para aqueles que não se conhece a sua presença.
- Selecionar mais de uma base de dados de um conjunto de base de dados e iniciar as buscas para os itens que se conhece a sua presença e para aqueles que não se conhece a sua presença.

Teste de partições

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

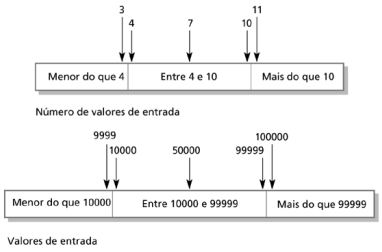
Particionamento de equivalência

Figura 23.8
Particionamento de equivalência.



Partições de equivalência

Figura 23.9
Partições de equivalência.



Especificação de 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.

Rotina de busca – partições de entrada

Tabela 23.1 Partições equivalentes para rotina de busca

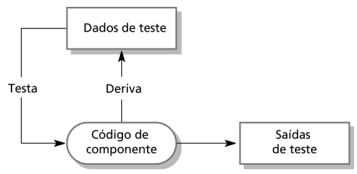
Seqüência	Elemento	
Valor único	Em seqüência	
Valor único	Fora da seqüência	
Mais de 1 valor	Primeiro elemento na seqüência	
Mais de 1 valor	Último elemento na seqüência	
Mais de 1 valor	Elemento médio na seqüência	
Mais de 1 valor	Fora da seqüência	
Seqüência de entradas (T)	**Chave (Key)**	**Saídas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

- Algumas vezes chamado de teste caixa-branca.
- É a derivação de casos de teste de acordo com a estrutura do programa. O conhecimento do programa é usado para identificar casos de teste adicionais.
- O objetivo é exercitar todas as declarações do programa (não todas as combinações de caminhos).

Teste estrutural

Figura 23.11
Teste estrutural.

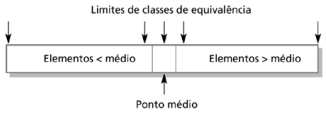


Busca binária – partições de equivalência

- Pré-condições satisfeitas, elemento key no vetor.
- Pré-condições satisfeitas, elemento key não está no vetor.
- Pré-condições não satisfeitas, elemento key no vetor.
- Pré-condições não satisfeitas, elemento key não está no vetor.
- Vetor de entrada tem um valor único.
- Vetor de entrada tem um número par de valores.
- Vetor de entrada tem um número ímpar de valores.

Busca binária – partições de equivalência

Figura 23.13
Classes de equivalência de busca binária.



Busca binária – casos de teste

Tabela 23.2 Casos de teste para a rotina de busca

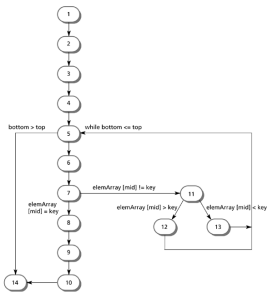
Vetor de entrada (T)	Elemento-chave (Key)	Saída (Found, L)
17	17	verdadeiro, 1
17	0	falso, ??
17, 21, 23, 29	17	verdadeiro, 1
9, 16, 18, 30, 31, 41, 45	45	verdadeiro, 7
17, 18, 21, 23, 29, 38, 41	23	verdadeiro, 4
17, 18, 21, 23, 29, 33, 38	21	verdadeiro, 3
12, 18, 21, 23, 32	23	verdadeiro, 4
21, 23, 29, 33, 38	25	falso, ??

Teste de caminho

- O objetivo do teste de caminho é assegurar que o conjunto de casos de teste é tal que cada caminho pelo programa é executado pelo menos uma vez.
- O ponto de partida do teste de caminho é um fluxograma de programa que mostra os nós que representam as decisões do programa e arcos que representam o fluxo de controle.
- Declarações com condições são, portanto, nós no fluxograma.

Fluxograma da rotina de busca

Figura 23.14
Fluxograma para a rotina de busca.



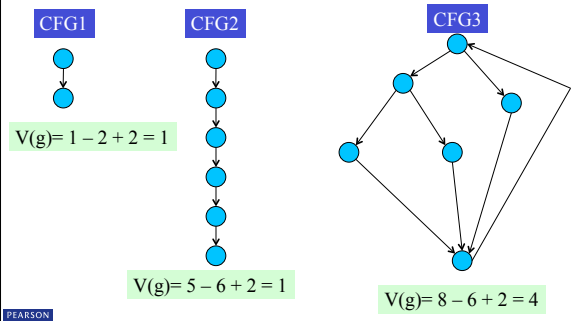
Caminhos independentes

- 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.
- Um analisador dinâmico de programa pode ser usado para verificar se os caminhos foram executados.

Complexidade ciclomática (McCabe)

- Medida do número de caminhos independentes em um programa
- Não depende do tamanho do código, mas dos ramos na estrutura de controle
- É medida por *e* – *n* + *2*, onde *e* é a quantidade de arestas do grafo de controle e *n* é a quantidade de nós do grafo

Complexidade ciclomática



Complexidade ciclomática

- Baixa a moderada (abaixo de 20) indica um programa simples
- Alta (acima de 20) indica um programa complexo
- Muita alta (acima de 50) caracteriza um programa muito difícil de testar

Complexidade ciclomática

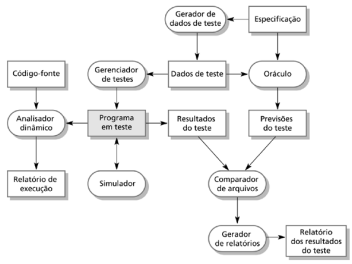
- Sinal de estrutura de controle de fluxo complicada
- Não captura outros aspectos da dificuldade lógica que podem levar a dificuldades no teste
- Poucas evidências de que é uma ferramenta de previsão de esforço de teste mais confiável do que linhas de código

Automação de teste

- Teste é uma fase dispendiosa do processo. Os workbenches de teste fornecem uma variedade de ferramentas para reduzir o tempo necessário e os custos totais de teste.
- Sistemas tais como o JUnit apóiam a execução automática de testes.
- A maioria dos workbenches de teste são sistemas abertos porque as necessidades de teste são específicas da organização.
- Eles são, algumas vezes, difíceis de integrar com workbenches de projeto e análise fechados.

Um *workbench* de testes

Figura 23.15
Workbench de testes.



Adaptação do workbench de testes

- Scripts podem ser desenvolvidos para simuladores de interface de usuário e padrões para geradores de dados de teste.
- Saídas de teste podem ser preparadas manualmente para comparação.
- Comparadores de arquivos para propósitos específicos podem ser desenvolvidos.