Gerência de Projetos e Qualidade de Software

Prof. Walter Gima

OBJETIVO

- Compreender uma série de técnicas de testes, que são utilizadas para descobrir defeitos em programas
- Conhecer as diretrizes que apóiam os testes de interfaces de componentes
- Compreender abordagens específicas dos testes de componentes e dos testes de integração para sistemas orientados a objetos
- Compreender os princípios da operação de apoio a ferramentas CASE para testes

INTRODUÇÃO

- Os testes de software tem como base uma compreensão intuitiva de como esses componentes deveriam operar

- Sob a perspectiva dos testes, os sistemas orientados a objetos diferem dos sistemas orientados a funções:
  - Nos sistemas orientados a funções, existe uma distinção nítida entre as unidades básicas de programas (funções) e seus módulos. Em sistemas orientados a objetos, não há tal distinção. Os objetos podem ser simples entidades, como uma lista, ou entidades complexas

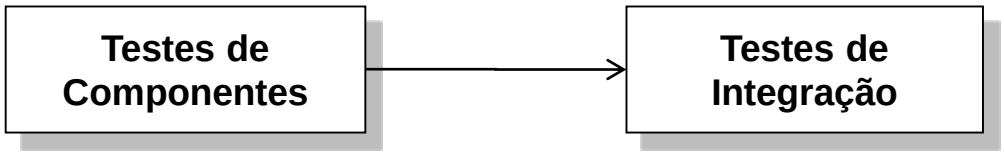
  - Freqüentemente, não existe hierarquia de objetos definida, como nos sistemas orientados a funções. As estratégias de integração, como a integração top-down e botton-up são muitas vezes inadequadas

Qualidade de Software

FASES

- Teste de Componentes
  - Testar o funcionamento dos componentes claramente identificáveis
    - Funções ou grupos de métodos de objetos

- Testes de integração
  - Componentes são integrados para formar subsistemas ou o sistema completo
  - Os testes devem focalizar as interações entre os componentes e a funcionalidade e o desempenho dos sistema como um todo



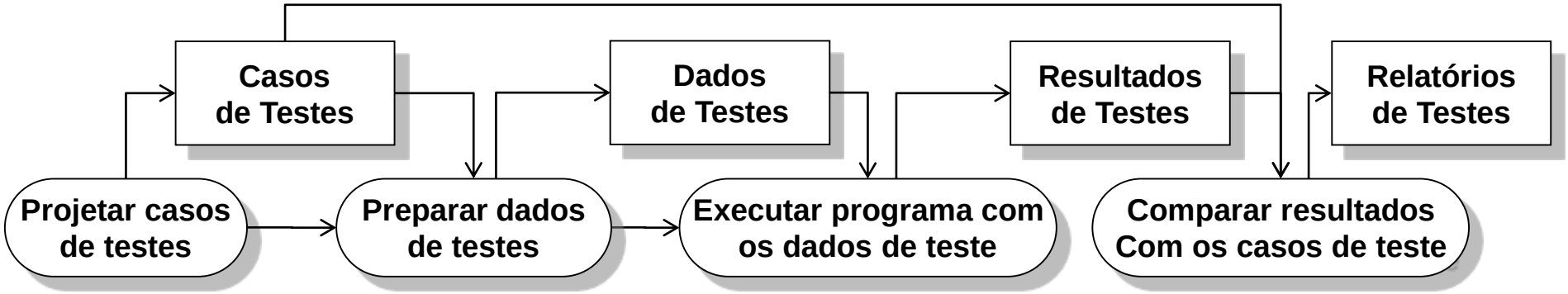
SOMMERVILLE, Figura 20.1, Pag. 376 – Fases de testes

TESTES

- A função é expor defeitos latentes em um sistema de software antes de sistema ser entregue
- Isto contrasta com os testes de validação, que se destinam a demonstrar que um sistema cumpre com suas especificações
- Um teste bem-sucedido para a detecção de defeitos é aquele que faz com que o sistema opere incorretamente e expõe um defeito existente

PROCESSO

- Os testes têm de ser baseados em um subconjunto casos de testes:
  - Todas as funções de sistema que são acessadas por meio de menus devem ser testadas
  - As combinações de funções que são acessadas pelo mesmo menu devem ser testadas
  - Quando a entrada do usuário é fornecida, todas as funções devem ser testadas com entradas corretas e incorretas



SOMMERVILLE, Figura 20.2, Pag. 377 – Processo de testes para detecção de defeitos

TESTES

- Os testes funcionais, ou testes de “caixa preta”, são uma abordagem na qual os testes são derivados da especificação de programa ou de um componente

- O sistema é uma “caixa preta” cujo comportamento somente pode ser determinado estudando-se suas entradas e saídas relacionadas

- A esse método também se dá o nome de testes funcionais, porque o testador está preocupado somente com a funcionalidade, e não com a implementação do software

PARTIÇÃO

- Os dados de entrada de um programa se dividem em diferentes classes, com características comuns (números positivos, negativos etc.)
- Os programas se comportam de maneira comparável para todos os membros de uma classe. Devido a esse comportamento equivalente, essas classes podem ser chamadas de partições de equivalência, ou domínios
- Uma abordagem dos testes para a detecção de defeitos baseia-se em identificar partições de equivalência que tenham de ser manuseadas por um programa
- Os casos de teste são projetados de modo que as entradas e as saídas fiquem dentro dessas partições
- As partições de equivalência de entradas são dados em que os membros do conjunto são processados de maneira equivalente
- As partições de equivalência de saída são saídas de programa que tem características comuns e podem ser consideradas uma classe distinta

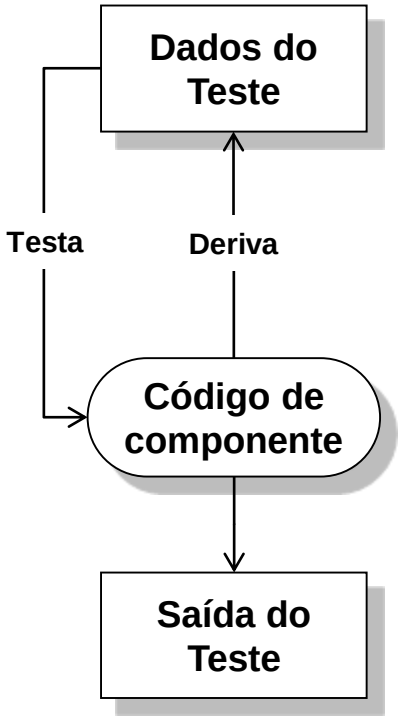
PARTIÇÃO

- Diretrizes de testes
  - Teste o software com seqüência que tenham somente um único valor. Os programadores pensam nas seqüência como constituídas de diversos valores e, algumas vezes, incluem essa suposição em seus programas. Como conseqüência, o programa pode não trabalhar adequadamente quando apresentado com uma seqüência de valor único
  - Utilize diferentes seqüência, de diferentes tamanhos, em diferentes testes. Isso diminui as chances de que um programa com defeito produza uma saída correta, devido a algumas características acidentais de entrada
  - Derive testes de maneira que o primeiro, o médio e o último elemento da seqüência sejam acessados. Essa abordagem revela problemas nos limites da partição

TESTE

- São testes derivados do conhecimento da estrutura e da implementação do software. Também pode ser chamada de testes de “caixa branca”, “caixa de vidro” ou “caixa clara”

- São aplicados às unidades de programa pequenas, como sub-rotinas, ou às operações associadas com um objeto. O testador pode analisar o código e utilizar conhecimento sobre a estrutura de um componente, a fim de derivar os dados para o teste. A análise do código pode ser utilizada para descobrir quantos casos de teste são necessários para garantir que todas as declarações no programa ou componente sejam executadas pelo menos uma vez durante o processo de teste



TESTES

- São uma estratégia de teste de estrutura, cujo objetivo é exercitar cada caminho de execução independente, por meio de um componente ou programa
- Se cada caminho independente for executado, então todas as declarações no componente devem ter sido executadas pelo menos uma vez
- Além do mais, todas as declarações condicionais são testadas para os casos verdadeiros e os falsos
- Se todos os caminhos forem executados, pode-se estar certo que:
  - Todas as declarações no método foram executadas pelo menos uma vez
  - Todos os ramos foram exercitados para as condições verdadeiras e falsas

TESTES

- Depois de descobrir o número de caminhos independentes no código pelo cálculo de complexidade ciclomática, o próximo passo será projetar casos de teste para executar cada um desses caminhos. O número mínimo de casos de teste requerido para testar todos os caminhos do programa é igual à complexidade ciclomática

- Durante a compilação, instruções adicionais de código-objeto são acrescentadas ao código gerado. Estas contam o número de vezes que cada declaração de programa foi executada. Depois que o programa foi executado, um perfil de execução pode ser impresso, que mostra quais partes do programa foram ou não executadas, utilizando casos de teste particulares

- Portanto, esse perfil de execução revela seções do programa que não foram testadas

TESTES

- Uma vez testados os componentes individuais de programa, eles devem ser integrados para criar um sistema parcial ou completo

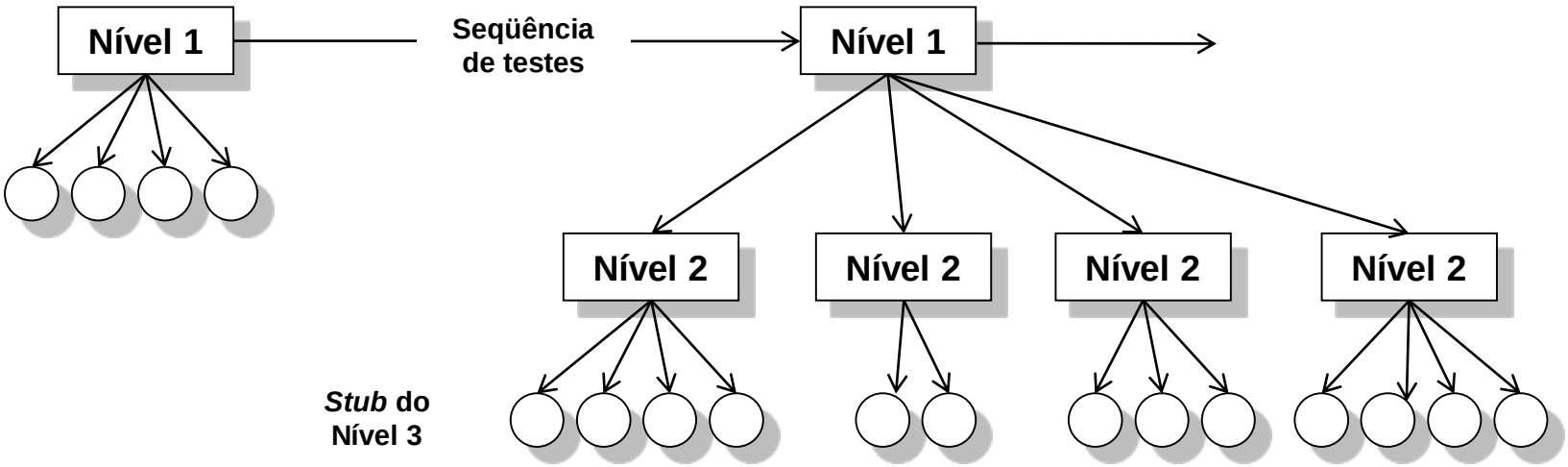
- Esse processo de integração envolve construir o sistema e testar o sistema resultante quanto a problemas que surjam a partir das interações de componentes

- Os testes de integração devem ser desenvolvidos a partir da especificação do sistema e começar assim que as versões de alguns dos componentes do sistema estejam disponíveis

- A principal dificuldade que surge nos testes de integração é localizar erros descobertos durante o processo

TESTE DE INTEGRAÇÃO TOP-DOWN

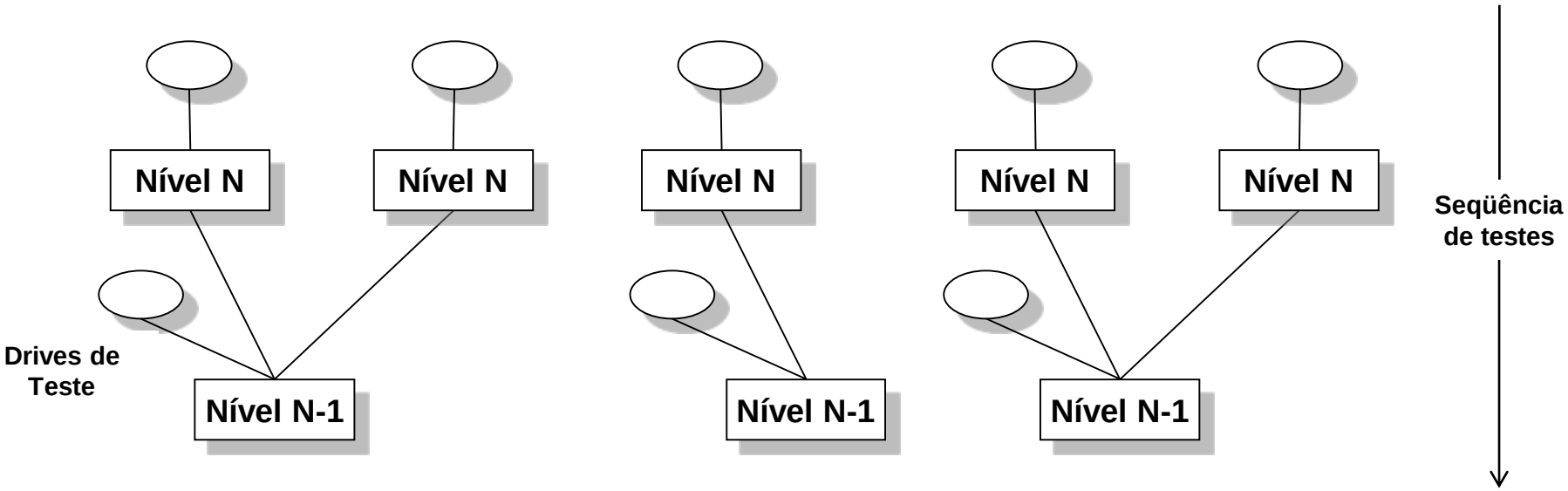
- A integração dos componentes de alto nível de um sistema são integrados e testados antes que seus projetos e sua implementação tenham sido completados
- O desenvolvimento se inicia com os componentes de alto nível e vai descendo na hierarquia de componentes



SOMMERVILLE, Figura 20.14, Pag. 387 – Testes de integração top-down e botton-up

TESTE

- Os componentes de nível inferior são integrados e testados antes que os componentes de nível superior tenham sido desenvolvidos
- Envolvem integrar e testar os módulos de nível inferior na hierarquia e, então, subir na hierarquia de módulos, até que o módulo final seja testado



SOMMERVILLE, Figura 20.14, Pag. 387 – Testes de integração top-down e botton-up

TESTES

- Ocorrem quando módulos ou subsistemas são integrados para criar sistemas maiores
- Cada módulo ou subsistema tem uma interface definida, que é chamada por outros componentes do programa
- O objetivo dos testes de interface é detectar erros que possam ter sido introduzidos no sistema, em razão de erros ou suposições inválidas sobre as interfaces

- Tipos de erros
  - Interfaces de parâmetros
  - Interfaces de memória compartilhada
  - Interfaces de procedimento
  - Interface de passagem de mensagem

TESTES

- Os erros de interface são um dos mais comuns em sistemas complexos. Eles podem ser:
  - Mau uso da interface
    - Um componente chama outro componente e comete um erro no uso de sua interface
  - Mau entendimento da interface
    - Um componente que chama interpreta mal a especificação da interface do componente chamado e faz suposições sobre seu comportamento
  - Erros *de timing*
    - Ocorrem em sistemas em tempo real, que utilizam uma memória compartilhada ou uma interface de passagem de mensagens

DIRETRIZES PARA TESTES DE INTERFACE

- Examine o código a ser testado e relacione cada chamada para um componente externo. Projete testes em que o valor dos parâmetros par aos componentes externos esteja nos limites extremos. Esses valores extremos podem revelar inconsistências da interface

- Quando ponteiros são passados por meio de uma interface, sempre teste a interface com parâmetros de ponteiros nulos

- Quando um componente é chamado por meio de uma interface de procedimento, projete testes que devam provocar a falha do componente. Suposições diferentes de falhas é um dos equívocos mais comuns de especificação

DIRETRIZES

- Utilize testes de estresse em sistemas de passagem de mensagens. Projete testes que gerem muito mais mensagens do que é possível ocorrer na prática. Problemas de *timing* podem ser revelados dessa maneira

- Quando vários componentes interagem por meio de memória compartilhada, projete testes que variem a ordem em que esses componentes são ativados

TESTE

- Os testes de estresse continuam além da carga máxima de projeto do sistema, até que o sistema falhe

- Funções
  - Testa o comportamento de falha do sistema
  - Causa estresse no sistema e pode acarretar a detecção de defeitos que, normalmente não se manifestariam

- São testes relevantes para sistemas distribuídos, com base em uma rede de processadores

TESTES

- Os testes de componentes, onde os componentes de sistema são testados individualmente, e os testes de integração, em que coleções de componentes são integradas em subsistemas e no sistema final para testes são aplicáveis para sistemas orientados a objetos

- Contudo, existem diferenças:

  – Objetos como componentes individuais são, muitas vezes, maiores do que funções isoladas
  – Os objetos que são integrados em subsistemas são, em geral, indevidamente acoplados, e não há um nível superior óbvio para o sistema
  – Se objetos forem reutilizados, os testadores podem não ter nenhum acesso ao código-fonte do componente para análise

TESTES

- Em um sistema orientado a objetos, quatro níveis de testes podem ser identificados:
  - Testar as opções individuais associadas com objetos
    - Podem ser utilizadas testes de “caixa preta” ou “caixa branca”
  - Testar classes de objetos individuais
    - Testes de seqüência das operações
  - Testar agrupamentos de objetos
    - Testes como base de cenários
  - Testar o sistema orientado a objetos
    - A verificação e validação aos requisitos

TESTES

- Quando objetos são testados, a cobertura completa de testes deve incluir:
  - O teste isolado de todas as operações associadas com o objeto
  - O estabelecimento e a interrogação de todos os atributos associados com o objeto
  - O exercício do objeto em todos os estados possíveis. Isso significa que todos os eventos que provoquem uma mudança de estado no objeto devem ser simuladas

INTEGRAÇÃO

- Teste de Cluster
  - Quando os sistemas orientados a objetos são desenvolvidos, os níveis de integração são menos distintos
  - Operações e dados são integrados para formar objetos e classes de objetos
  - O teste dessas classes de objetos corresponde ao teste de unidade
  - Deve-se criar grupos de classes que agem em combinação para fornecer um conjunto de serviços a serem testados juntos

INTEGRAÇÃO

- Existem três possíveis abordagens para os testes de integração, que podem ser utilizadas:
  - Testes de use-case baseados em cenários
    - Podem ser baseados nessas descrições de cenário e nos clusters de objetos criados para lidar com use-cases que se relacionam com aquele modo de uso
  - Testes de threads
    - Baseiam-se no testa da resposta do sistema a uma entrada particular ou a um conjunto de eventos de entrada
  - Testes e interação de objetos
    - Sugere que um nível intermediário de testes de integração pode ter como base a identificação de caminhos de método-mensagem

WORKBENCHES

- Ferramentas de teste de software
  - Gerenciador de testes
    - Gerencia a execução dos testes
  - Gerador de dados de teste
    - Gera dados de teste para o programa a ser testado
  - Oráculo
    - Gera previsões dos resultados esperados para o teste

WORKBENCHES

- Ferramentas de teste de software
  - Comparador de arquivos
    - Compara os resultados dos testes de programa com os resultados de testes precedentes e relata as diferenças entre eles
  - Gerador de relatórios
    - Fornece uma definição de recursos de geração para os resultados de testes
  - Analisador dinâmico
    - Adiciona código a um programa para contar o número de vezes que cada declaração foi executada
  - Simulador
    - Simulam a máquina em que o programa deve ser executado

WORKBENCHES

- Adaptações dos sistemas de testes
  - Novas ferramentas podem precisar ser acrescentadas para testar características especificas da aplicação
  - Scripts podem ter de ser escritos para simuladores de interface com o usuário e padrões definidos para geradores de dados de teste
  - Conjuntos de resultados esperados dos testes podem ser preparados manualmente, se nenhuma versão prévia do programa estiver disponível para servir como oráculo
  - Comparadores de arquivo de propósito especial podem ser escritos, incluindo o conhecimento da estrutura dos resultados de teste em arquivos

Dúvidas ?

walter.gima@anhanguera.com