Engenharia de Software

Teste de Software

Verificação e validação

Testes de desenvolvimento

Testes de release

Testes de usuário

Desenvolvimento dirigido a testes

Kele Teixeira Belloze

kelebelloze@gmail.com

Introdução

- Os testes são utilizados para:
  - Mostrar que um programa faz o que lhe é destinado fazer;
  - Descobrir os defeitos do programa antes desse ser colocado em uso.

- Quando testamos um software:
  - É executado o programa usando dados artificiais.
  - É verificado os resultados do teste procurando por: erros, anomalias ou informações sobre os atributos que não funcionam do programa.

- **Atenção:** testes podem revelar a presença de erros, porém não podem garantir a sua ausência.

- O teste é parte de um processo mais geral, de verificação e validação, que também inclui técnicas de validação estática.

Objetivos do processo de testes

1. Demonstrar para o desenvolvedor e o cliente que o software atende aos seus requisitos.

   - ✓ Para softwares customizados: deve apresentar pelo menos um teste para cada requisito do documento de requisitos.
   - ✓ Para produtos de software genéricos: deve apresentar testes para todos as características do sistema, além de suas combinações, as quais serão incorporadas no release do produto.

   Tipo de teste: testes de validação.

2. Descobrir situações em que o comportamento do software está incorreto, indesejável ou em desacordo com sua especificação.

   - ✓ Testes de defeitos se concentram em eliminar comportamentos indesejáveis do sistema, tais como: falhas no sistema, interações indesejáveis com outros sistemas, cálculos incorretos e corrompimento de dados.

   Tipo de teste: testes de defeito.

Verificação e validação

- Verificação:

  “Estamos construindo o produto certo?”

  “Estamos construindo o produto de maneira correta?”

  - ✓ O software deve estar em acordo com sua especificação.

- Validação:

  "Estamos construindo certo o produto?".

  - ✓ O software deve fazer o que o usuário realmente necessita.

Verificação e Validação (V & V)

- O objetivo da V & V é estabelecer a confiança de que o sistema "se encaixa no propósito".

- O nível de confiabilidade depende do propósito do software, das expectativas do usuário e do das decisões de marketing.

  - Propósitos do software
    - O nível de confiança depende do quanto o software é crítico para uma organização.

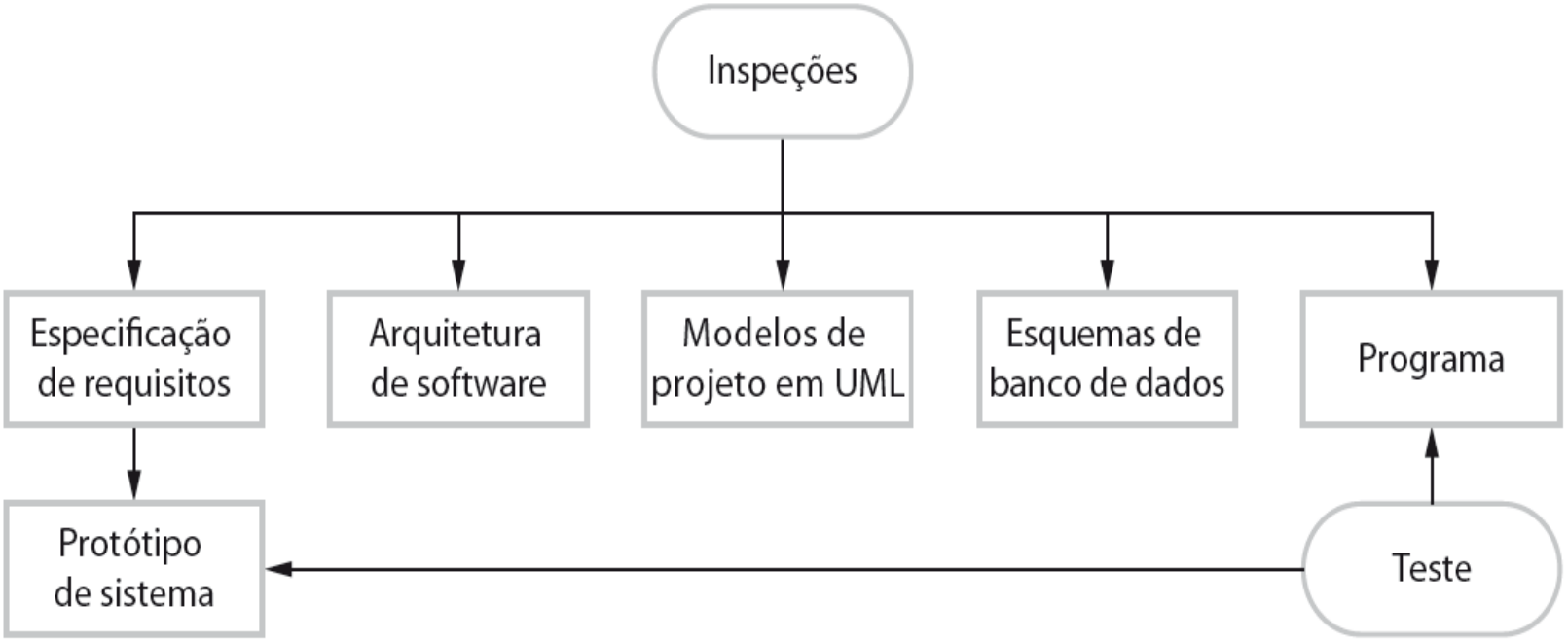
  - Expectativas do usuário
    - Usuários podem ter expectativas baixas em certos tipos de software.

  - Decisões de marketing
    - Lançar um produto antes no mercado pode ser mais importante que encontrar defeitos no programa.

Inspeções e testes

- Inspeções de software
  - ✓ Preocupam-se na análise da representação estática do sistema para descobrir problemas (verificação estática).
- Testes de software
  - ✓ Preocupam-se com a execução e a observação do comportamento do produto (verificação dinâmica)
  - ✓ O sistema é executado com dados de teste e seu comportamento operacional é observado.

Inspeções e testes



Inspeções de software

- Envolvem pessoas examinando o código-fonte com o objetivo de descobrir anomalias e defeitos.
- Não exigem a execução do sistema, podendo então, ser utilizadas antes da implementação.
- Podem ser aplicadas a qualquer representação do sistema: requisitos, dados de projeto, configuração, dados de teste, etc.
- Têm se mostrado uma técnica eficaz para descobrir defeitos de programa.

Vantagens das inspeções

- Durante os testes, os erros podem mascarar outros erros.
  - Como a inspeção é um processo estático, não é necessário se preocupar com as interações entre os erros.
- Versões incompletas de um sistema podem ser inspecionadas sem custos adicionais.
- Buscam defeitos no programa e consideram atributos de qualidade de um programa, como o cumprimento de normas, portabilidade e manutenibilidade.

Inspeções e testes

- São técnicas de verificação complementares.
- Ambas podem ser usadas durante o processo de V &V.
- As inspeções podem verificar a conformidade com uma especificação, mas não conseguem verificar a conformidade com os requisitos reais do cliente.
- As inspeções não podem verificar características não-funcionais, como desempenho, usabilidade, etc.

Políticas de testes

- Testes de sistema exaustivos são impossíveis, assim políticas de teste definem a cobertura necessária dos testes do sistema.

- Exemplos de políticas de testes:

  - ✓ Todas as funções do sistema que são acessados através de menus devem ser testadas.

  - ✓ Devem ser testadas todas as combinações de funções acessadas por meio do mesmo menu.

  - ✓ Todas as funções nas quais a entrada do usuário é fornecida devem ser testadas com entradas corretas e incorretas

Estágios de teste

- Testes de desenvolvimento: o sistema é testado durante seu desenvolvimento para descobrir *bugs* e defeitos.
- Testes de release: uma equipe de testes separada testa uma versão completa do sistema antes que ele seja liberado para os usuários.
- Testes de usuário: os usuários ou potenciais usuários de um sistema testam o sistema em seu próprio ambiente.

Testes de desenvolvimento (1/9)

- Testes de desenvolvimento incluem todas as atividades de testes que são realizadas pela equipe de desenvolvimento do sistema.
  - ✓ Teste de unidade: são testadas as unidades individuais do programa ou classes de objetos.
    - ✓ Os teste de unidade devem se concentrar em testar a funcionalidade dos objetos ou métodos.
  - ✓ Testes de componentes: quando várias unidades individuais são integradas para criar componentes compostos.
    - ✓ O teste de componentes deve se concentrar em testar as interfaces dos componentes.
  - ✓ Teste de sistema: quando algum ou todos os componentes de um sistema são integrados e o sistema é testado como um todo.
    - ✓ Devem se concentrar em testar interações de sistemas.

Testes de unidade (2/9)

- Um teste completo de uma classe envolve:
  - ✓ Testes de todas as operações associadas com um objeto;
  - ✓ Informar e solicitar todos os atributos do objeto;
  - ✓ Exercitar o objeto em todos os estados possíveis.

Teste de Componentes/Teste de Interface (3/9)

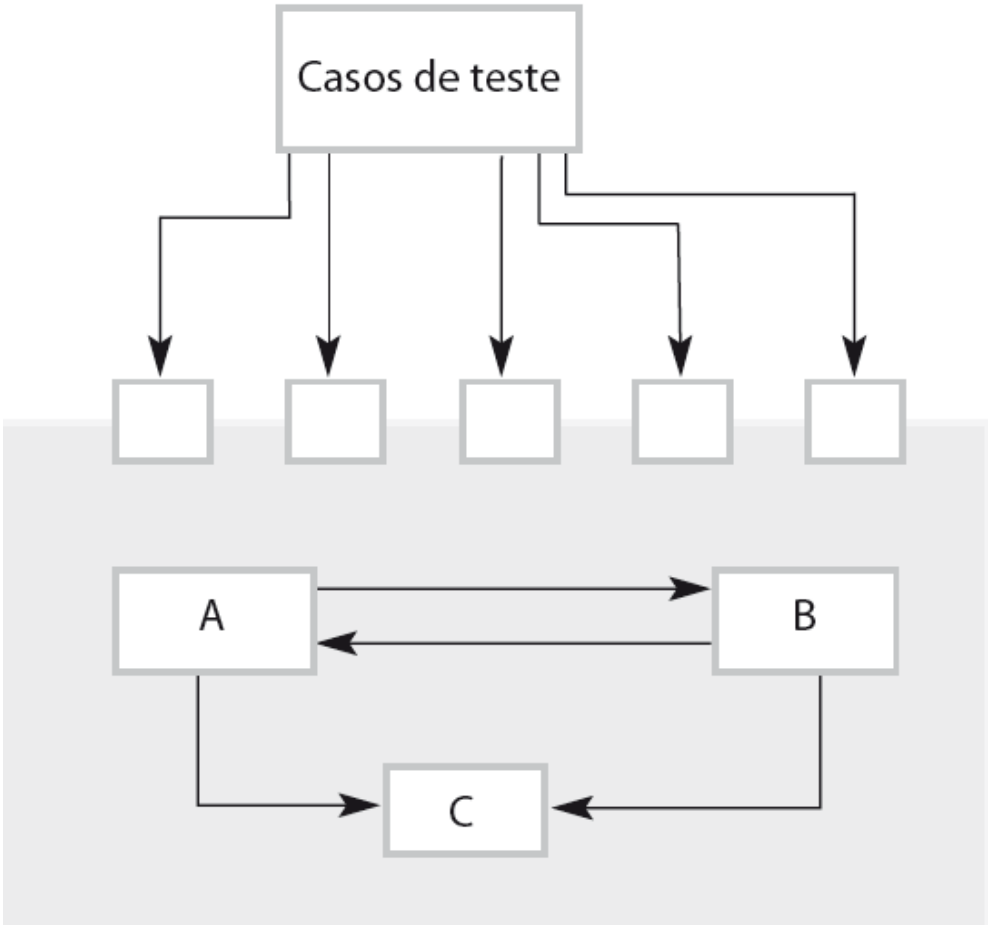
- Geralmente, os componentes de software não são funções ou objetos simples, mas sim componentes compostos de vários objetos que interagem.

- A funcionalidade desses objetos é acessada através da interface do componente definida.

- Portanto, os testes de componentes compostos devem focar em mostrar que a interface do componente se comporta de acordo com sua especificação.

  - ✓ Pode-se supor que já foram concluídos os testes de unidade sobre os objetos individuais dentro do componente.

Processo de Teste de interface (4/9)



Os casos de teste não são aplicados nos componentes individuais, mas sim na interface do componente composto, formado pela combinação dos componentes.

Teste de Interface (5/9)

- Os objetivos do teste de interface são:
  - Detectar defeitos devidos a erros de interface ou,
  - Suposições inválidas sobre as interfaces.

- Tipos de interfaces

  - ✓ Interfaces de parâmetro: dados são passados de um componente para o outro.
  - ✓ Interfaces de memória compartilhada: blocos de memória são compartilhados entre os componentes.
  - ✓ Interfaces de procedimentos: componentes sintetizam um conjunto de procedimentos para serem chamados por outros componentes.
  - ✓ Interfaces de passagem de mensagem: um componente solicita um serviço de outro componente passando uma mensagem para ele.

Erros de interface (6/9)

Erros de interface constituem uma das formas mais comuns de erros em sistemas complexos. Estes são divididos em categorias:

- Mau uso de interface
  - ✓ Um componente chama outro componente e comete um erro no uso da sua interface. Exemplo: parâmetros passados na ordem errada.

- Mau entendimento da interface
  - ✓ Um componente chamador incorpora suposições incorretas sobre o comportamento dos componentes chamados. Exemplo: uma rotina de busca binária pode ser chamada com um vetor não ordenado.

- Erros de *timing*
  - ✓ O componente chamador e o componente chamado operam em velocidades diferentes e são acessadas informações desatualizadas.

Diretrizes de testes de interface (7/9)

1. Projete os testes de modo que os valores dos parâmetros para um procedimento chamado estejam nos extremos de suas escalas.

2. Sempre teste parâmetros de ponteiros com ponteiros nulos.

3. Projete os testes que causem a falha do componente onde este é chamado por meio de uma interface de procedimento.

4. Use testes de estresse em sistemas de passagem de mensagens.

5. Em sistemas de memória compartilhada, varie a ordem em que os componentes são ativados.

Testes de sistema (8/9)

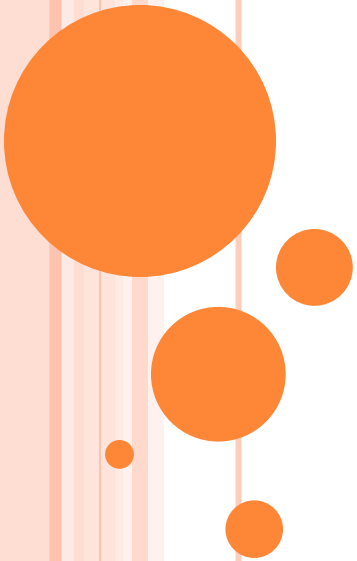
- Testes de sistema durante o desenvolvimento envolvem a integração dos componentes para criar uma versão do sistema e, em seguida, testar o sistema integrado.

- O foco dos testes de sistema é testar as interações entre os componentes.

- Os testes de sistema verificam se os componentes são compatíveis, se interagem corretamente e transferem os dados certos no momento certo por suas interfaces.

Testes de sistema (9/9)

- Nesse estágio podem ser integrados os componentes desenvolvidos por diferentes membros da equipe ou subequipes.
- Os testes de sistema são um processo coletivo, e não um processo individual.
  - ✓ Em algumas empresas, o teste de sistema pode envolver uma equipe de testes separada do envolvimento de projetistas e programadores.



Testes de release

Testes de release

- Teste de release é o processo de testes de uma versão particular do sistema que será distribuído aos clientes.

- O principal objetivo do processo de teste de release é aumentar a confiança do fornecedor de que o sistema atende aos requisitos.

  - ✓ Portanto, os testes de release precisam mostrar que o sistema oferece a funcionalidade, o desempenho e confiabilidade especificados, e que não falha durante o uso normal.

- Geralmente, os testes de release são um processo de teste caixa-preta, em que os testes são derivados somente a partir da especificação do sistema.

Testes de release e testes de sistema

- Testes de release são uma forma de teste do sistema.
- Diferenças importantes:
  - ✓ Uma equipe separada, sem envolvimento com o desenvolvimento do sistema, deve ser responsável pelo testes de release.
  - ✓ Os testes de sistema realizados pela equipe de desenvolvimento devem se concentrar na descoberta de *bugs* do sistema (teste de defeitos).
  - ✓ O objetivo do teste de release é verificar se o sistema atende aos seus requisitos e é bom o suficiente para uso externo. (teste de validação).

Testes de desempenho

- Parte dos testes de release podem envolver ensaios sobre as propriedades emergentes de um sistema, tais como desempenho e confiabilidade.

- Os testes de desempenho envolvem o planejamento de uma série de testes, nos quais a carga é aumentada continuamente até que o desempenho do sistema se torne inaceitável.

  - Testes de estresse são uma forma de testes de desempenho em que o sistema é deliberadamente sobrecarregado para testar seu comportamento até falhar.

Testes de Usuário

Testes de usuário

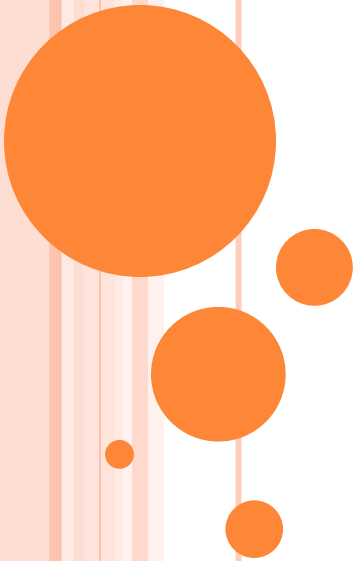
- Testes de usuário ou cliente, é uma etapa no processo de teste em que os usuários ou clientes fornecem informações e conselhos sobre os testes de sistema.

- Testes com usuários são essenciais, mesmo havendo os testes de sistema abrangentes e testes de release.

  - ✓ A razão é que as influências do ambiente de trabalho do usuário tem um efeito importante sobre a confiabilidade, desempenho, usabilidade e robustez de um sistema.
  - ✓ Esses não podem ser replicados em um ambiente de teste.

Tipos de testes de usuário

- Testes alfa
  - Usuários do software trabalham com a equipe de desenvolvimento para testar o software no local do desenvolvedor.
- Testes beta
  - Um release do software é disponibilizado para os usuários para que possam experimentar e levantar os problemas descobertos com os desenvolvedores do sistema.
- Testes de aceitação
  - Clientes testam um sistema para decidir se esse está pronto para ser aceito e implantado no ambiente do cliente. Principalmente para sistemas sob encomenda.



Desenvolvimento dirigido a testes

Desenvolvimento dirigido a testes

- O desenvolvimento dirigido a testes (TDD – *Test Driven Development*) é uma abordagem para o desenvolvimento de programas em que se intercalam testes e o desenvolvimento de código.

- Testes são escritos antes do código e "passar" nos testes é o fator crítico de desenvolvimento.

- O código é desenvolvido de forma incremental, juntamente com um teste para esse incremento.
  - Não se passa para o próximo incremento até que o código desenvolvido passe no teste.

- TDD foi introduzido como parte dos métodos ágeis como o *Extreme Programming*.
  - No entanto, ele também pode ser usado em processos de desenvolvimento dirigido a planos.

Benefícios do TDD

- Cobertura de código
  - ✓ Cada segmento de código que é escrito deve ter pelo menos um teste associado.
- Testes de regressão
  - ✓ Um conjunto de testes de regressão é desenvolvido de forma incremental enquanto um programa é desenvolvido.
- Depuração simplificada
  - ✓ Quando um teste falhar, deve ser óbvio onde está o problema. O código recém-escrito tem de ser verificado e modificado.
- Documentação de sistema
  - ✓ Os próprios testes são uma forma de documentação que descreve o que o código deve fazer.

Testes de regressão

- Testes de regressão testam o sistema para verificar se as mudanças não "quebram" os códigos previamente trabalhados.

- Em um processo de teste manual, os testes de regressão são caros, mas, com testes automatizados, são mais simples.

- Todos os testes são reexecutados toda vez que é feita uma alteração no programa.

- Os testes devem ser executados com 'sucesso' antes da mudança ser executada.

Referência

- PRESSMAN, Roger S., Engenharia de Software – Uma Abordagem Profissional, 7ª edição, São Paulo: Mc Graw Hill, 2011.
- SOMMERVILLE, Ian, Engenharia de Software, 9ª edição, São Paulo: Pearson Education – Addison-Wesley, 2011.