Testes de Software
(Capítulo 8 Sommerville)

slide 569

Tópicos abordados

- Testes de desenvolvimento
- Desenvolvimento dirigido a testes
- Testes de *release*
- Testes de usuário

Testes de programa

- Os testes são destinados a
  - mostrar o que um programa faz,
  - o que pretende fazer e
  - para **descobrir os defeitos** do programa antes desse ser colocado em uso.
- Ao testar o software, você executa um programa usando **dados artificiais.**
- Você verifica os resultados do teste para erros, anomalias ou informações sobre os atributos não funcionais do programa.
- **Podem revelar a presença de erros, NÃO a sua ausência**.
- O **teste é parte de um processo de verificação e validação** mais geral, que também inclui técnicas de validação estática.

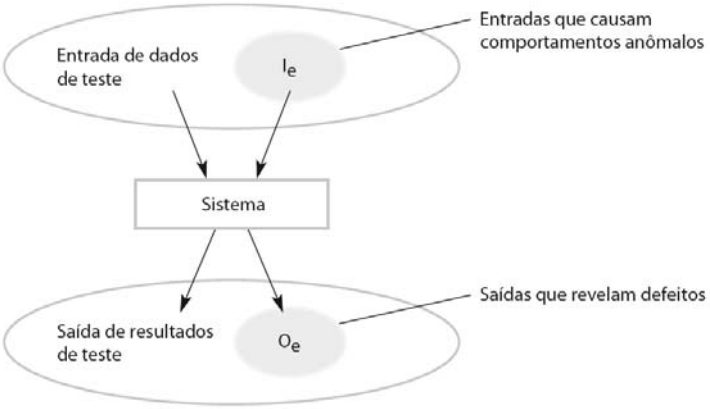
Objetivos do processo de testes

- **Demonstrar** para o desenvolvedor e o cliente que o **software atende** aos seus requisitos.
  - ✓ Para softwares customizados, isso significa que **deve haver pelo menos um teste para cada requisito no documento de requisitos**. Para produtos de software genéricos, isso significa que deve haver testes para todos as características do sistema, além de suas combinações, as quais serão incorporadas no release do produto.
- Para **descobrir situações** em que o comportamento do **software está incorreto**, indesejável ou em desacordo com sua especificação.
  - ✓ Testes de defeitos estão interessados em eliminar comportamentos indesejáveis do sistema, tais como falhas no sistema, interações indesejáveis com outros sistemas, cálculos incorretos e corrupção de dados.

Objetivos do processo de testes

- **Testes de validação**
  - ✓ Para demonstrar para o desenvolvedor e o cliente que o sistema de software corresponde às suas exigências.
  - ✓ **Um teste bem sucedido mostra que o sistema opera como planejado**.
- **Testes de defeitos**
  - ✓ Para descobrir falhas ou defeitos no software, em que seu comportamento é incorreto ou não está em conformidade com a especificação.
  - ✓ **Um teste bem sucedido** é um teste que faz o sistema funcionar incorretamente e, dessa maneira **expõe um defeito no sistema.**

Um modelo entrada-saída de teste de programa



Verificação vs. validação

- **Verificação:**

  "Estamos construindo o produto da maneira correta?".

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

- **Validação:**

  "Estamos construindo o produto certo?".

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

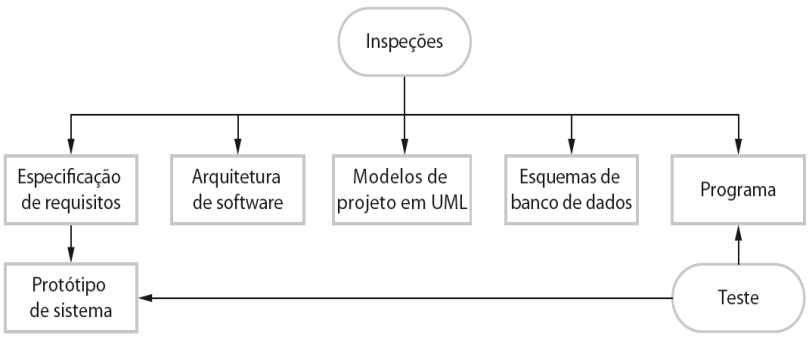
Confiança e V & V

- O objetivo de V &V é estabelecer a confiança na ”adequação do sistema ao seu intuito".
- Depende da finalidade do software, das expectativas do usuário e do ambiente de marketing
  - ✓ Finalidade 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.
  - ✓ Ambiente de marketing
    - Colocar um produto em um mercado rapidamente pode ser mais importante do que encontrar defeitos no programa.

Inspeções e testes

- **Inspeções de software** interessadas na **análise da representação estática** do sistema para descobrir problemas (**verificação estática**)
  - ✓ Pode ser suplementado por ferramentas baseadas em documentos e análise de códigos.
- **Teste de software** interessados no **exercício e observando o 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

- Essas envolvem **pessoas examinando** principalmente a representação do código-fonte com o objetivo de descobrir anomalias e defeitos.
- As inspeções **não exigem a execução** de um sistema, assim, podem ser usadas antes da implementação.
- Elas podem ser **aplicadas a qualquer representação** do sistema (requisitos, os dados de projeto, configuração, dados de teste, etc.)
- Têm se mostrado uma técnica **eficaz** para descobrir defeitos de programa.

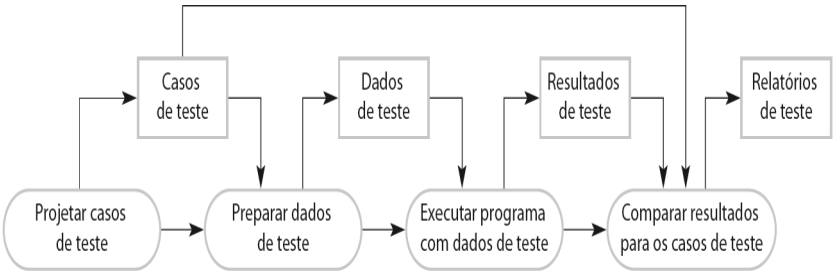
Vantagens das inspeções

- **Durante os testes,** os erros podem **mascarar (esconder**) outros erros. Pois a inspeção é um processo estático, no qual não é necessário se preocupar com as interações entre os erros.
- **Versões incompletas de um sistema podem ser inspecionadas** sem custos adicionais.
- Se um programa está incompleto, então é necessário desenvolver testes especializados para testar as partes que estão disponíveis.
- Bem como procurar por defeitos no programa, uma **inspeção** também pode considerar **atributos mais amplos de qualidade** de um programa, como o cumprimento de normas, portabilidade e manutenibilidade.

Inspeções e testes

- Inspeções e testes são **complementares** e não técnicas opostas de verificação.
- **Ambos** devem ser usadas durante o processo de V &V.
- As inspeções podem verificar a conformidade com uma especificação, mas não a conformidade com os requisitos reais do cliente.
- As inspeções **não podem verificar** características não-funcionais, como **desempenho**, **usabilidade**, etc.

Um modelo do processo de teste de software



Estágios de teste

- **Testes de desenvolvimento**, no qual o sistema é testado durante seu desenvolvimento para descobrir bugs e defeitos.
- **Testes de release**, em que uma **equipe de testes** separada testa uma **versão completa do sistema** antes que ele seja liberado para os usuários.
- **Testes de usuário**, em que os **usuários ou potenciais usuários** de um sistema testam o sistema em seu **próprio ambiente**.

Testes de desenvolvimento

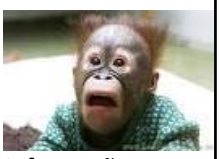
- Testes de desenvolvimento incluem todas as atividades de testes que são realizas pela equipe de desenvolvimento do sistema.
  - ✓ **Teste de unidade**, em que são testadas as unidades de programa individual ou classes de objetos. Os teste de unidade devem se concentrar em testar a **funcionalidade dos objetos ou métodos**.
  - ✓ **Testes de componentes**, em que várias unidades individuais são **integradas** para criar componentes compostos. Testes de componentes devem se concentrar em testar as interfaces dos componentes.
  - ✓ **Teste de sistema**, em que alguns ou todos os componentes de um sistema são integrados e o **sistema é testado como um todo**. Esses devem se concentrar em testar interações entre os componentes.

Testes de unidade

- Teste de unidade é o processo de teste de componentes individuais isoladamente.
- **É um processo de teste de defeitos**.
- As unidades podem ser:
  - ✓ **Funções** individuais ou métodos dentro de um objeto
  - ✓ **Classes** de objetos com vários atributos e métodos

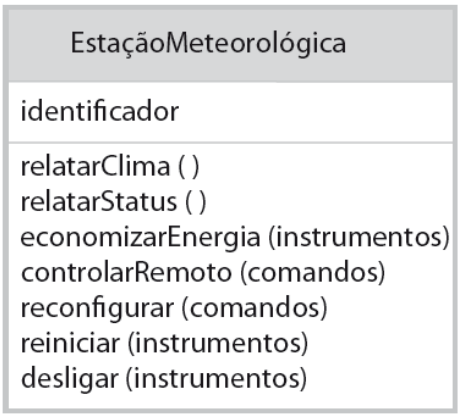
Testes de classe

- A **cobertura completa** dos testes de uma classe envolve
  - ✓ Testes de **todas as operações** associadas com um objeto
  - ✓ **Definição e verificação de todos os atributos** do objeto
  - ✓ **Exercitar o objeto em todos os estados possíveis**.



- A **herança torna mais difícil projetar testes de classe** pois a informação a ser testada não é localizada.

A interface do objeto Estação Meteorológica



Testes na estação meteorológica

- Necessidade de definir casos de testes para relatar o clima, calibrar, reiniciar e desligar.
- Usando um modelo de estado, identificar as sequências de transições de estado a serem testadas e as sequências de eventos para causar essas transições.
- Por exemplo:
  - ✓ Desligar-> Executar-> Desligar
  - ✓ Configurar-> Executar-> Testar -> Transmitir -> Executar
  - ✓ Executar->Coletar->Executar->Resumir->Transmitir->Executar

Testes automatizados

- **Sempre que possível, os testes de unidade deve ser automatizados** para que essas sejam executados e verificados sem intervenção manual.
- Em testes de unidade automatizados, você faz uso de um **framework de automação de teste (como JUnit)** para escrever e executar os testes do seu programa.
- **Frameworks de testes de unidade fornecem classes de testes genéricos** que você estende para criar casos de teste específicos.
- Eles podem executar todos os testes implementados e **informar**, muitas vezes por meio de alguma interface gráfica, **sobre o sucesso ou fracasso dos testes**.

Componentes de testes automatizados

- **Configuração**: você inicia o sistema como caso de teste, ou seja, com as entradas e saídas esperadas.
- **Chamada**: você chama o objeto ou o método a ser testado.
- **Afirmação**: você compara os resultados da chamada com o resultado esperado.
- Se a afirmação é avaliada como verdadeira, o teste foi bem sucedido, se for falsa, então ele falhou.

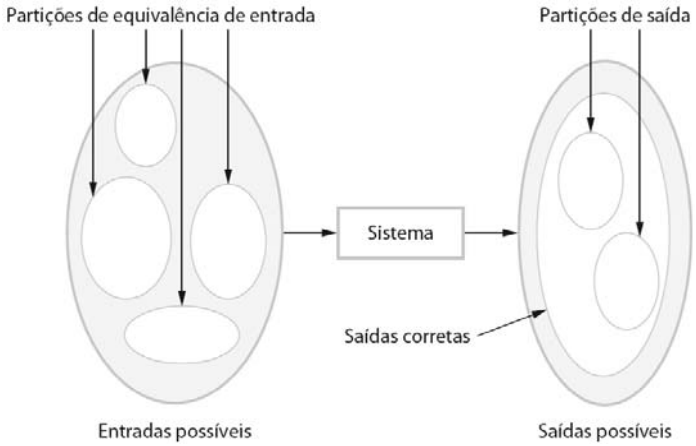
Eficácia dos testes de unidade

- Os casos de teste devem mostrar que, **quando usado como esperado**, o componente sendo testado **faz o que supostamente ele deve fazer**.
- Se **houver defeitos no componente**, esses devem ser **revelados** por casos de testes.
- O que nos leva a **dois tipos** de casos de teste de unidade:
  - ✓ O primeiro deve refletir a **operação normal** de um programa e deve mostrar que o componente funciona como esperado.
  - ✓ O outro tipo de casos de teste deve ser baseado em testes de experiências, de onde surgem os **problemas comuns**. Ele deve **usar entradas anormais** para verificar se esses são devidamente processados e que o componente não falhe.

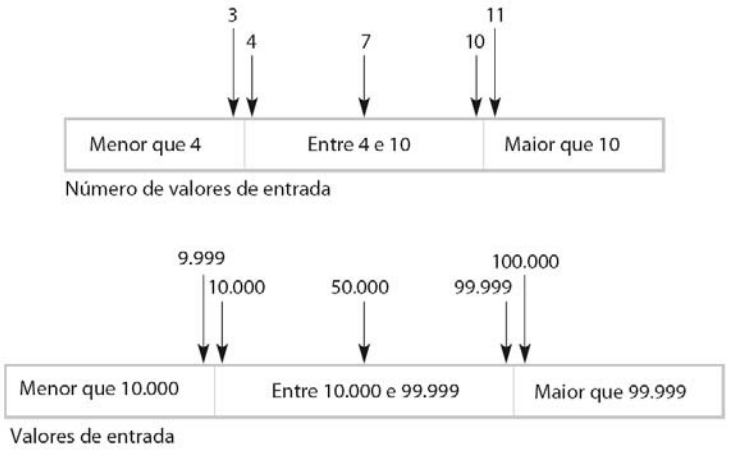
Estratégias de teste

- **Testes de partição**, nos quais se identificam **grupos de entradas** que têm características comuns e devem ser processados da mesma maneira.
  - ✓ Você deve **escolher** os testes de **dentro de cada um desses grupos**.
- **Testes baseados em diretrizes**, em que você usa diretrizes de testes para escolher casos de teste.
  - ✓ Essas diretrizes refletem a **experiência anterior dos tipos de erros** que os programadores cometem frequentemente no desenvolvimento de componentes.

Particionamento de equivalência



Partições de equivalência



Diretrizes de teste (sequências)

- Testar o software com as **sequências que possuem apenas um único valor**.
- Usar **sequências de tamanhos diferentes** em diferentes testes.
- Derivar testes para que o **primeiro elemento** da sequencia, os do **meio** e o **último sejam acessados**.
- Testar com **sequências de comprimento zero**.

Diretrizes gerais de testes

- Escolher entradas que forcem o sistema a **gerar todas as mensagens de erro**.
- Projetar entradas que causem o **transbordamento dos buffers de inputs**.
- **Repetir** a mesma entrada ou uma série de entradas inúmeras vezes.
- Forçar a geração de **saídas inválidas**.
- Forçar os **resultados** de cálculos serem **muito grandes ou muito pequenos**.

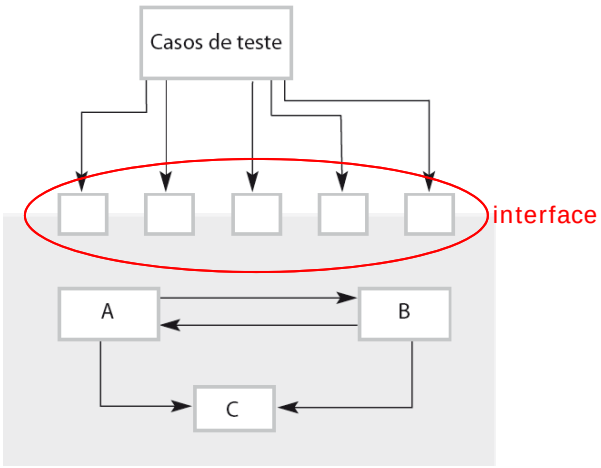
Pontos Importantes

- Os testes podem apenas mostrar a presença de erros em um programa. **Testes não podem demonstrar que não existem defeitos** remanescentes.
- Testes de **desenvolvimento** são de responsabilidade da **equipe** de desenvolvimento de software.
- Uma **equipe separada** deve ser responsável por **testar** um sistema **antes** que ele seja liberado para os clientes.
- Testes de desenvolvimento incluem **testes de unidade**, em que você testa objetos individuais e métodos, **testes de componentes**, em que você testa grupos de objetos relacionados e **testes do sistema**, em que você testa sistemas parciais ou completos.

Teste de componente

- Geralmente, os **componentes de software** são componentes compostos de **vários classes que interagem**.
  - ✓ Por exemplo, no sistema da estação meteorológica, o componente reconfiguração inclui classes que lidam com cada aspecto da reconfiguração.
- Você acessa a funcionalidade desses classes através da **interface do componente definido**.
- Portanto, os **testes de componentes** compostos devem focar em mostrar que a **interface do componente** se comporta de **acordo com sua especificação**.
  - ✓ Você pode **supor** que já foram concluídos os **testes de unidade** sobre classes individuais dentro do componente.

Teste de interface



Teste de interface

- Os objetivos são detectar defeitos devidos **a erros de interface** ou **suposições inválidas** sobre as interfaces.
- Tipos de interfaces
  - ✓ Interfaces de **parâmetro** - Dados passados de um método ou procedimento para o outro.
  - ✓ Interfaces de **memória compartilhada** - Blocos de memória são compartilhados entre os procedimentos ou funções.
  - ✓ Interfaces de **procedimento** - Subsistemas sintetizam um conjunto de procedimentos para serem chamados por outros subsistemas.
  - ✓ Interfaces de **passagem de mensagem** - Subsistemas solicitam serviços de outros subsistemas.

Erros de interface

- **Mau uso de interface**
  - ✓ Um componente chamado chama outro componente e comete um erro no uso da sua interface, por exemplo, **de parâmetros na ordem errada**.
- **Mau entendimento de interface**
  - ✓ Um componente chamador, incorpora **suposições incorretas** sobre o comportamento dos componentes chamados.
- **Erros de timing**
  - ✓ O componente chamador e o componente chamado **operam em velocidades diferentes** e são acessadas informações desatualizadas.

Diretrizes de testes de interface

- Projete os testes de modo que os valores dos **parâmetros** para um procedimento chamado estejam **nos extremos** de suas escalas.
- Sempre teste **parâmetros** de ponteiros com **ponteiros nulos**.
- Projete os testes que **busquem causar a falha** do componente.
- Use **testes de estresse** em sistemas de **passagem de mensagens**.
- Em sistemas de **memória compartilhada**, **varie a ordem** em que os componentes são **ativados**.

Testes de sistema

- 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.
- Os testes de sistema **testam o comportamento emergente** de um sistema.

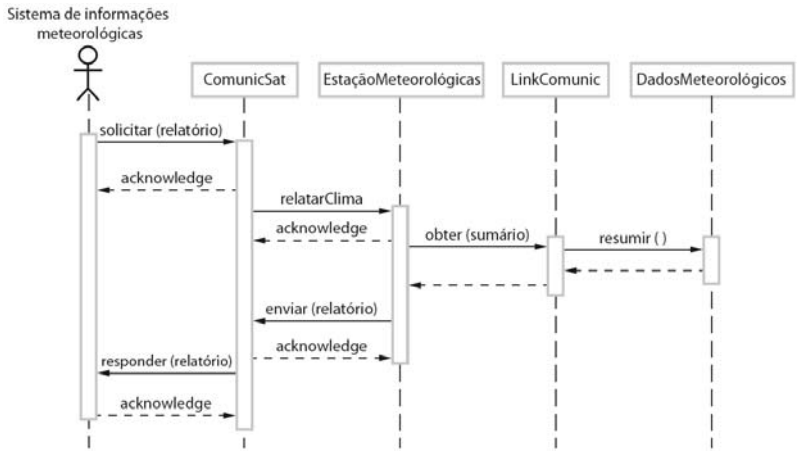
Testes de sistema e de componentes

- Durante os testes de sistema, os **componentes reusáveis** que tenham sido desenvolvidos separadamente e os sistemas de prateleira **podem ser integrados** com os componentes recém-desenvolvidos. Em seguida, o sistema completo é testado.
- 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 casos de uso

- Os **casos de uso** desenvolvidos para identificar as interações do sistema **podem ser usados como uma base** para testes de sistema.
- Geralmente, **cada caso de uso** envolve **vários componentes** do sistema, assim que, os testes de caso de uso forçam a ocorrência dessas interações.
- Os **diagramas de sequência** associados com os documentos de casos de uso, **mostram** como os componentes e as interações **estão sendo testadas**.

Diagrama de sequência de coletar dados meteorológicos



Engenharia de Software
Prof. Flávio de Oliveira Silva, Ph.D.

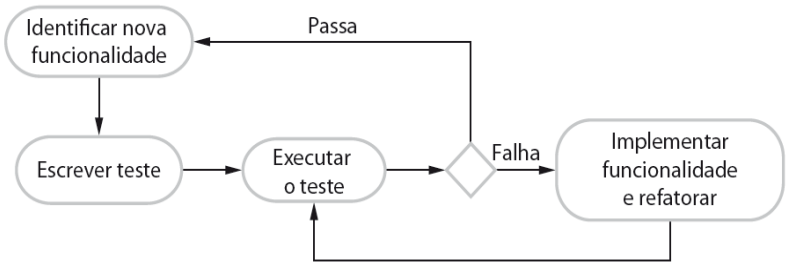
Políticas de testes

- **Testes de sistema exaustivos são impossíveis**,
  - assim **políticas de teste** que definem a **cobertura necessária** dos testes do sistema devem ser desenvolvidas.
- **Exemplos** de políticas de testes:
  - ✓ **Todas as funções** do sistema que são **acessadas** através de menus devem ser testadas.
  - ✓ Devem ser testadas **todas as combinações de funções** (por exemplo, formatação de texto) acessadas por meio do mesmo menu.
  - ✓ Onde a **entrada do usuário é fornecida**, todas as funções devem ser testadas com **entradas corretas e incorretas**.

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.
- Você desenvolve **o código de forma incremental**, juntamente com um t**este para esse incremento**. Você não passa para o próximo incremento até que o código que você desenvolveu passe no seu 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.

Processo de atividades de TDD



- Iniciar identificando o incremento de funcionalidade.
  - Normalmente pequeno e implementável em poucas linhas de código.
- Escrever um teste para esta funcionalidade e implementar isso como um teste automatizado.
- Executar o teste, junto com todos os outros testes que foram implementados.
  - Inicialmente, você não implementou a funcionalidade de modo que o novo teste irá falhar.
- Implementar a funcionalidade e reexecutar o teste.
- Uma vez que todos os testes forem executados com sucesso, você vai para próxima parte de funcionalidade.

Benefícios do TDD

- **Cobertura de código**
  - ✓ **Cada segmento** de código que você escreve deve **ter pelo menos um teste** associado, para todo o código escrito tem pelo menos um teste.
- **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 estar fazendo.

Testes de regressão

- **Testes de regressão** testam o sistema para verificar se as **mudanças não "quebram"** o código previamente trabalhado.
- Em um **processo** de teste **manual**, os teste de regressão são **caros**, mas, com testes **automatizados**, são **simples** e **diretos**.
- **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.

Testes de release

- **Teste de release** é o processo de testes de uma **versão** particular de um sistema que se destina para **uso fora da equipe** de desenvolvimento.
- O principal objetivo do processo de teste de release é **convencer o cliente** de que o sistema é bom o suficiente para o uso.
  - ✓ 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 centrar 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 requisitos

Envolvem o exame de cada requisito e o desenvolvimento de um ou mais testes para esses

- Requisitos do MHC-PMS:
  - ✓ Se é sabido que um paciente é alérgico a algum medicamento em particular, a prescrição dessa medicação deve resultar na emissão de uma **mensagem de aviso** para o usuário do sistema.
  - ✓ Se um médico opta por **ignorar um aviso de alergia**, ele deve fornecer o motivo pelo qual o aviso foi ignorado.

TESTES

- Criar o registro de um paciente com alergia. Prescrever a medicação para a alergia do paciente, e verificar se o **aviso é emitido pelo siste**ma.
- Estabelecer um registro do paciente em que são registradas alergias a duas ou mais medicações. Prescrever ambas as medicações separadamente e verificar se é **emitido o aviso correto para cada droga**.
- Receitar dois medicamentos às quais o paciente é alérgico. Verificar se são **emitidos corretamente dois avisos**.
- Prescrever um medicamento que emite um aviso e **ignorar esse aviso**. Verificar se o sistema exige informações explicando por que o aviso foi ignorado.

Definindo testes a partir de cenários

Um cenário de uso para o MHC-PMS

Kate é uma enfermeira especialista em saúde mental. Uma de suas responsabilidades é visitar os pacientes em casa para verificar a eficácia do tratamento e se os pacientes estão sofrendo com os efeitos colaterais da medicação.

Em um dia de visitas, Kate faz o login no MHC-PMS e usa-o para imprimir sua agenda de visitas domiciliares para aquele dia, juntamente com o resumo das informações sobre os pacientes a serem visitados. Ela pede que os registros desses pacientes sejam transferidos para seu notebook. É solicitada a palavra-chave para criptografar os registros no notebook.

Um dos pacientes que Kate visita é Jim, que está sendo tratado com medicação para depressão. Jim acha que a medicação está ajudando, mas acredita que tem o efeito colateral de mantê-lo acordado durante a noite.

Um cenário de uso para o MHC-PMS

Kate vai consultar o registro de Jim; sua frase-chave é solicitada para decifrar o registro. Ela verifica o medicamento prescrito e consulta seus efeitos colaterais. Insônia é um efeito colateral conhecido.

Ela faz uma observação sobre o problema no registro de Jim e sugere que ele visite a clínica para ter sua medicação alterada. Ele concorda. Assim, Kate faz um prompt de entrar em contato com ele quando ela voltar à clínica, para que faça uma consulta com um médico. Ela termina a consulta e o sistema criptografa novamente o registro de Jim.

Depois, terminadas suas consultas, Kate retorna à clínica e transfere os registros dos pacientes visitados para o banco de dados. O sistema gera para Kate uma lista de pacientes que ela precisa contatar para obter informações de acompanhamento e fazer agendamentos de consultas.

Rastreabilidade das características testadas pelo cenário

- Autenticação por logon no sistema.
- Download e upload dos registros de um paciente específico para um computador portátil.
- Programação de visitas domiciliares.
- Criptografia e descriptografia do registros de um paciente em um dispositivo móvel.
- Recuperação e modificação de registros.
- Links com o banco de dados dos medicamentos, o qual mantém informações sobre os efeitos colaterais.
- O sistema de aviso de chamada.

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 devem refletir **o perfil de uso** do sistema.
- Geralmente, 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.

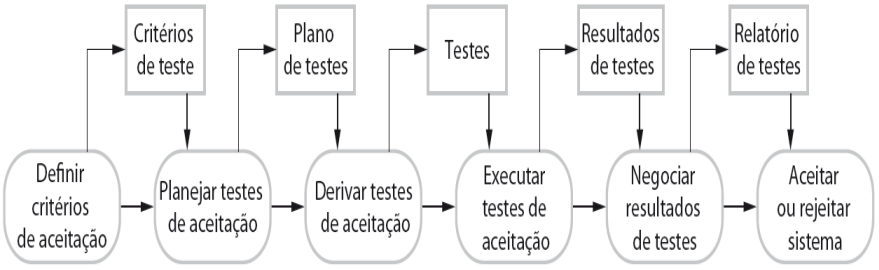
Testes de usuário

- **Testes de usuário** 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 quando já foram realizados os testes de sistema abrangentes e testes de release.
  - ✓ A razão para tanto, 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 se esse está pronto** para ser aceito, e **implantado** no ambiente do cliente. Principalmente para sistemas **sob encomenda**.

O processo de testes de aceitação



Métodos ágeis e testes de aceitação

- Em métodos ágeis, o cliente/usuário faz parte da equipe de desenvolvimento e é responsável pela tomada de decisões sobre a aceitabilidade do sistema.
- Os **testes são definidos pelo usuário/cliente** e são **integrados com outros testes** executados automaticamente quando mudanças são feitas.
- **Não existem** processo de **testes de aceitação separados**.
- O principal problema aqui é se o usuário incorporado **é ou não um usuário "típico"** e se pode representar os interesses de todos os *stakeholders* do sistema.

Pontos Importantes

- Ao testar o software, você deve **tentar "quebrar"** o software usando a experiência e as diretrizes para escolher tipos de casos de teste que têm sido eficazes na descoberta de defeitos em outros sistemas.
- Sempre que possível, você deve escrever **testes automatizados**.
- Cada vez que uma **mudança** é feita em um sistema, os **testes são incorporados** em um programa que possa ser executado. **Teste de regressão** são executados após mudança.
- O desenvolvimento ***test-first*** é uma abordagem para o desenvolvimento em que os testes são escritos antes do código ser testado.
- Os **testes de cenário envolvem a invenção de um cenário típico** de uso e para derivar casos de testes.
- Os **testes de aceitação** são um processo de teste de usuário, em que o objetivo é decidir se o software é **bom o suficiente para ser implantado** e usado em seu ambiente operacional.