Verificação e Validação

Centro de Informática - Universidade Federal de Pernambuco
Engenharia da Computação
Kiev Gama
kiev@cin.ufpe.br

Slides elaborados pelo professor Marcio Cornélio

UNIVERSIDADE FEDERAL DE PERNAMBUCO

Verificação vs Validação

- **Verificação**:
  "Estamos construindo o produto corretamente?"
- **O software deve estar de acordo com sua especificação.**
- **Validação**:
  "Estamos construindo o produto certo?".
- **O software deve fazer o que o usuário realmente deseja.**

O processo V & V

- **Deve ser aplicado a cada estágio do desenvolvimento de software**
  - Vale tanto para verificação quanto validação
- **Tem dois objetivos principais:**
  - Descobrir problemas em um sistema;
    - Problema = sistema que não satisfaz sua **especificação**
  - Avaliar se o sistema é útil e usável ou não em uma situação operacional.

Objetivos de V& V

- Verificação e validação devem estabelecer confiança de que o software é adequado ao seu propósito.
- Isto NÃO significa completamente livre de defeitos.
- Ao invés disso, deve ser bom o suficiente para seu uso pretendido
  - Tipo de uso determinará o grau de confiança necessário.

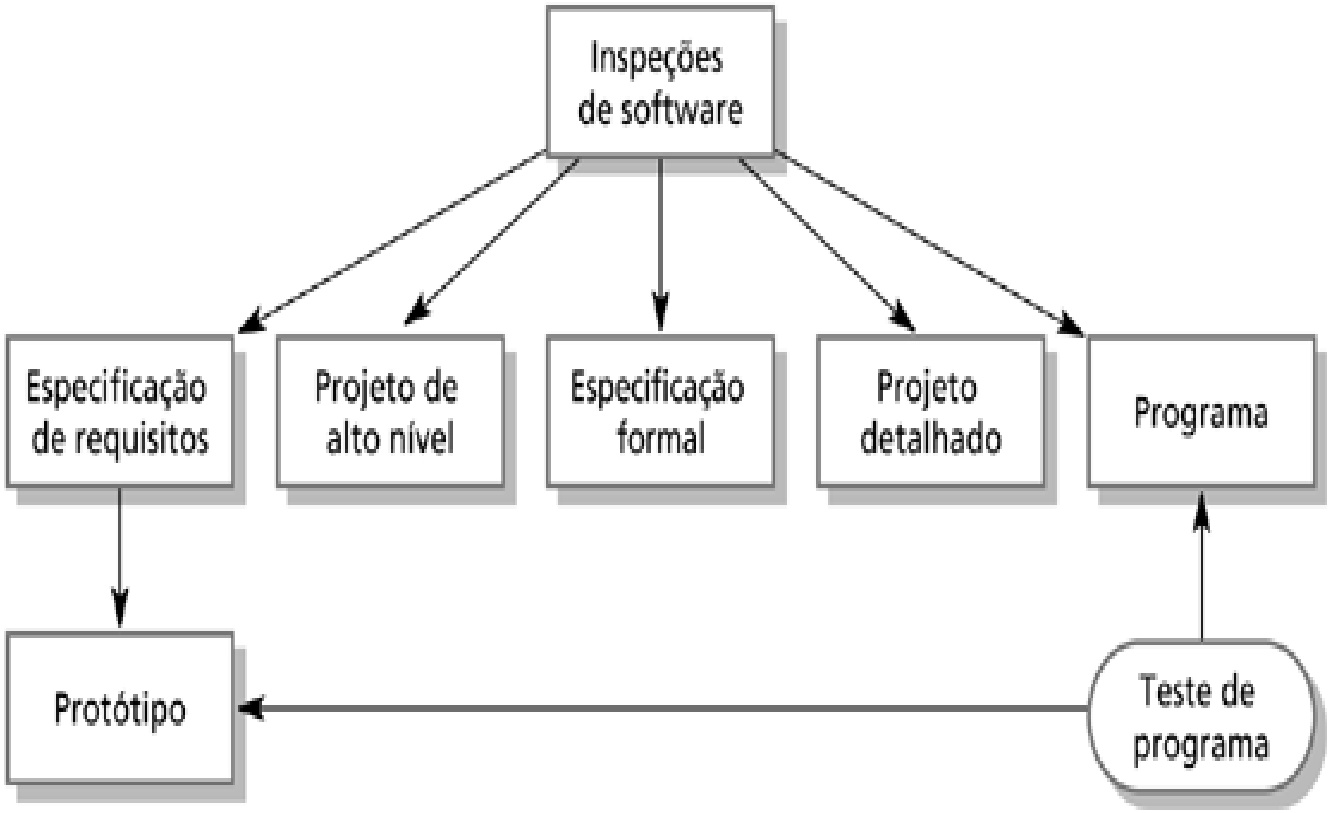
V & V estática e dinâmica

- **Inspeções de software. Análise de representações estáticas do sistema com o objetivo de descobrir problemas (verificação estática)**
  - Pode ser suplementado por um documento baseado em ferramenta e análise de código.
- **Teste de software. Relacionado ao exercício e à observação do comportamento do produto (verificação dinâmica)**
  - O sistema é executado com dados de teste e seu comportamento operacional é observado.
- **Outras técnicas: análise dinâmica, prototipação, entrevistas, cenários**

V & V estática e dinâmica

Figura 22.1

Verificação e validação dinâmica e estática.



Testes de Programas

- *"Testes podem apenas revelar a presença de defeitos, não a ausência deles"* (E. Djikstra)
- Principal técnica de validação para requisitos não-funcionais
  - O software é executado para ver como se comporta.
- Devem ser usados em conjunto com a verificação estática para fornecer uma cobertura mais completa de V&V.

Tipos de teste

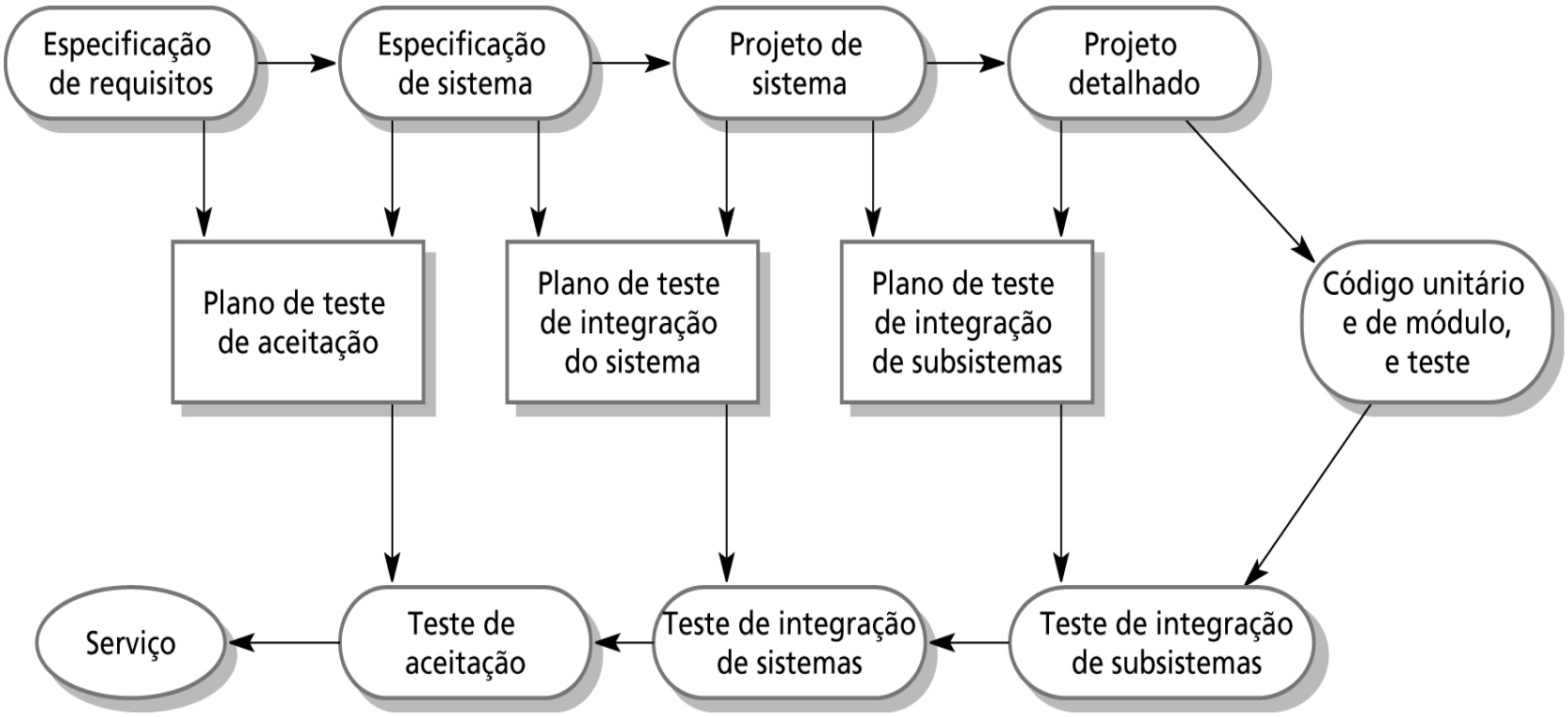
- Teste de validação
  - Pretende mostrar que o software atende as necessidades dos usuários;
  - Um teste bem sucedido é aquele que mostra que um requisito foi adequadamente implementado.
- Teste de defeitos
  - Testes projetados para descobrir defeitos de sistema;
  - Um teste de defeitos bem sucedido é aquele que revela a presença de falha em um sistema;

Teste e Depuração

- Testes e depuração de defeitos são processos distintos
- Verificação e validação estão relacionados ao estabelecimento da existência de falhas em um programa
- Depuração está relacionado à localização e repararação dessas falhas

O Modelo V de Desenvolvimento

Figura 22.3 Plano de teste como ligação entre o desenvolvimento e os testes.



Inspeções de software

- Exame de um artefato de desenvolvimento para descobrir anomalias e defeitos
  - Técnica de verificação
  - Feitas por uma equipe, em uma reunião formal
- Não requerem a execução de um sistema,
  - Podem e devem ser usadas antes da implementação
- Podem ser aplicadas a qualquer artefato
- Têm se mostrado uma técnica efetiva para descobrir erros de programa
  - *"...according to statistics it will find up to 90% of the contained errors, if done properly."*
    - (http://www.the-software-experts.de/e_dta-sw-test-inspection.htm)

Sucesso das inspeções

- Muitos defeitos diferentes podem ser descobertos em uma única inspeção.
  - Em teste, um defeito pode mascarar um outro, por isso, várias execuções são necessárias.
- Conhecimento sobre o domínio e sobre programação aumentam a eficácia
  - Revisores têm alta probabilidade de já ter visto os tipos de erros que normalmente surgem

Inspeções e testes

- Inspeções e testes são complementares
  - Inspeções => verificação
  - Testes => verificação e validação
- Ambos devem ser usados durante o processo de V & V.
- As inspeções podem verificar a conformidade com uma especificação
  - Não verificam a conformidade com os requisitos reais do cliente!
- As inspeções não podem verificar características de qualidade, tais como desempenho, usabilidade, etc.

Inspeções de Programas

- Abordagem formalizada para revisões de artefatos
- Voltadas explicitamente para detecção de falhas (não correção).
- Falhas podem ser erros lógicos (por exemplo, uma variável não iniciada) ou não-conformidade com padrões.

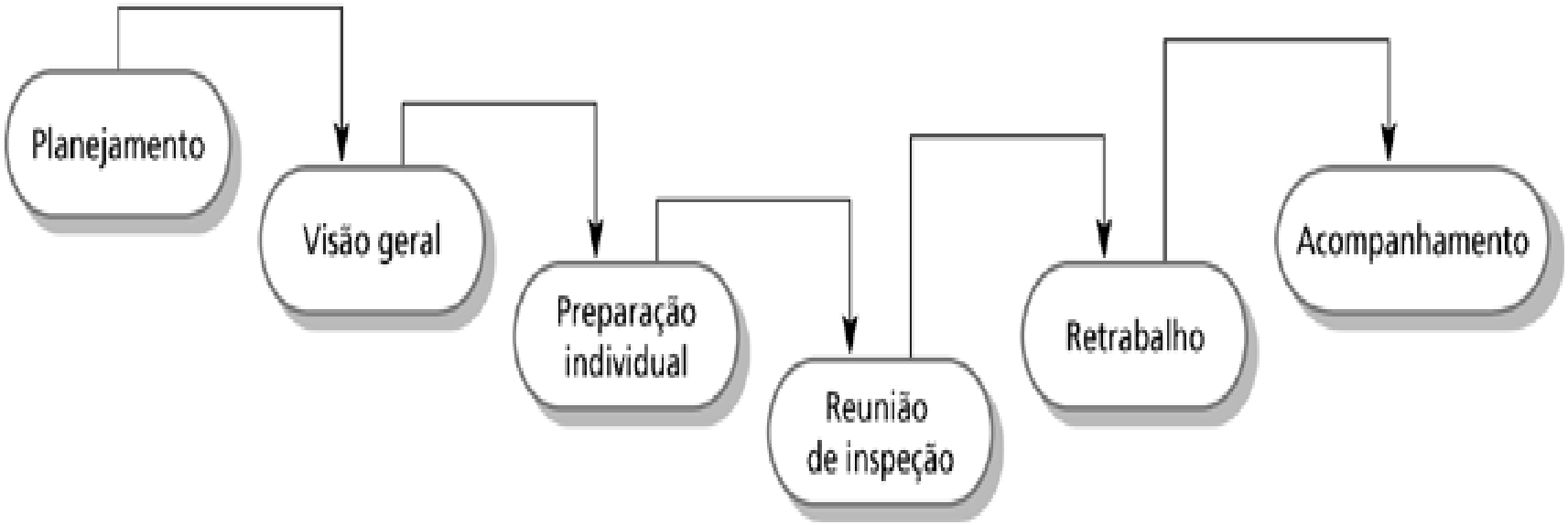
Pré-condições para inspeção

- Uma especificação precisa deve estar disponível
- Os membros da equipe devem estar familiarizados com os padrões organizacionais
- O código sintaticamente correto ou outras representações do sistema devem estar disponíveis.
- Um *checklist* de erros deve ser preparado

- A gerência deve aceitar que a inspeção aumentará os custos no início do processo de software
- A gerência não deve usar inspeções para avaliar pessoal

O processo de inspeção

Figura 22.4 Processo de inspeção.



Procedimento de inspeção

- Visão geral do sistema apresentado para a equipe de inspeção
- Código e documentos associados são previamente distribuídos para a equipe de inspeção
- A inspeção ocorre e os erros descobertos são anotados
  - Alguns podem ser descobertos na análise individual
- Modificações são feitas para reparar os erros descobertos
- Uma nova inspeção pode ou não ser necessária

Papéis da inspeção

Tabela 22.1 Papéis no processo de inspeção

Papel	Descrição
Autor e proprietário	O programador ou projetista responsável por produzir o programa ou o documento. Ele é responsável pela correção de defeitos descobertos durante o processo de inspeção.
Inspetor	Encontra erros, omissões e inconsistências nos programas e documentos. Pode também identificar questões mais amplas fora do escopo da equipe de inspeção.
Leitor	Apresenta o código ou documento em uma reunião de inspeção.
Relator	Registra os resultados da reunião de inspeção.
Presidente ou moderador	Gerencia o processo e facilita a inspeção. Relata os resultados do processo ao moderador-chefe.
Moderador-chefe	Responsável pelos aprimoramentos do processo de inspeção, pela atualização da lista de verificação, pelo desenvolvimento de padrões etc.

Checklists de Inspeção

- Um *checklist* de erros comuns deve ser usado para direcionar a inspeção.
- *Checklists* de erros são dependentes de linguagem de programação
  - Refletem os erros característicos com maior probabilidade de surgimento na linguagem
- Exemplos de itens da *checklist*: inicialização de variáveis, terminação de laços, etc.
- Inspeções também podem "executar" o sistema, através da análise passo-a-passo de seu código

Tabela 22.2 Verificações de inspeção

Classe de defeitos	Verificação de inspeção
Defeitos de dados	Todas as variáveis de programa são iniciadas antes que seus valores sejam usados? Todas as constantes foram denominadas? O limite superior de vetores deve ser igual ao tamanho do vetor ou Tamanho –1? Se são usados strings de caracteres, um delimitador é explicitamente atribuído? Existe alguma possibilidade de overflow de buffer?
Defeitos de controle	Para cada declaração condicional, a condição está correta? Cada loop está terminando corretamente? As declarações compostas estão corretamente delimitadas entre parênteses? Em declarações 'case', todos os casos possíveis são levados em conta? Se um comando 'break' é necessário após cada caso nas declarações 'case', ele foi incluído?
Defeitos de entrada/saída	Todas as variáveis de entrada são usadas? Todas as variáveis de saída têm valor atribuído antes de sua saída? Entradas inesperadas podem fazer com que os dados sejam corrompidos?
Defeitos de interface	Todas as chamadas de funções e de métodos têm o número correto de parâmetros? Tipos de parâmetros reais e formais se combinam? Os parâmetros estão na ordem correta? Se os componentes acessam memória compartilhada, eles têm o mesmo modelo de estrutura de memória compartilhada?
Defeitos de gerenciamento de armazenamento	Se uma estrutura ligada é modificada, todas as ligações foram corretamente reatribuídas? Se o armazenamento dinâmico foi usado, o espaço foi corretamente alocado? O espaço de memória é liberado depois de não ser mais necessário?
Defeitos de gerenciamento de exceções	Todas as condições possíveis de erro foram consideradas?

Taxa de Inspeção

- 500 declarações de código-fonte por hora durante a visão geral.
- 125 declarações de código fonte por hora durante a preparação individual.
- De 90 a 125 declarações por hora podem ser inspecionados durante a reunião de inspeção.
- A inspeção é, portanto, um processo dispendioso.
- A inspeção de 500 linhas custa aproximadamente 40 homem-hora de esforço – £2800 em valores da Grã-Bretanha (UK).

Análise Estática Automatizada

- Processamento de código fonte (ou *bytecode*)
- Varre o texto do programa e tenta descobrir condições potencialmente errôneas
  - Técnica de verificação
- São um suplemento, mas não um substituto, para as inspeções
- Podem ser usadas para aumentar a compreensão sobre um programa

Verificações de Análise Estática

Tabela 22.3 Verificação de análise estática automatizada

Classe de defeitos	Verificação de análise estática
Defeitos de dados	Variáveis usadas antes da inicialização Variáveis declaradas, mas nunca usadas Variáveis atribuídas duas vezes, mas nunca usadas entre atribuições Possíveis violações de limites de vetor Variáveis não declaradas
Defeitos de controle	Código inacessível Ramificações incondicionais em loops
Defeitos de entrada/saída	Variáveis geradas duas vezes sem tarefa de impedimento
Defeitos de interface	Tipo de parâmetro que não combina Número de parâmetro que não combina Resultados de funções não usadas Funções e procedimentos não chamados
Defeitos de gerenciamento de armazenamento	Ponteiros não atribuídos Aritmética de ponteiros

Tipos de Análise Estática

- Análise de fluxo de controle. Verifica laços com múltiplos pontos de saídas ou de entrada, encontra código inacessível, etc
- Análise de uso de dados. Detecta variáveis não iniciadas, variáveis que são declaradas mas nunca usadas, etc
- Análise de interface. Verifica a consistência das declarações de rotina e procedimentos e seus usos

Tipos de Análise Estática

- Análise de caminho. Identifica caminhos através do programa e estabelece as declarações executadas naquele caminho.

- Muitos outros tipos de análises são possíveis!
- Limitações: escalabilidade, completude, precisão, excesso de informações

Análise Estática com o Lint

Figura 22.5

Análise estática com o Lint.

```
138% more lint_ex.c
#include <stdio.h>
printarray (Anarray)
int Anarray;
{
printf("%d",Anarray);
}
main ()
{
int Anarray[5]; int i; char c;
printarray (Anarray, i, c);
printarray (Anarray) ;
}

139% cc lint_ex.c
140% lint lint_ex.c

lint_ex.c(10): warning: c may be used before set
int_ex.c(10): warning: i may be used before set
printarray: variable # of args. lint_ex.c(4) :: lint_ex.c(10)
printarray, arg. 1 used inconsistently lint_ex.c(4) :: lint_ex.c(10)
printarray, arg. 1 used inconsistently lint_ex.c(4) :: lint_ex.c(11)
printf returns value which is always ignored
```

Uso de Análise Estática

- Particularmente valiosa quando uma linguagem tal como C é usada
  - Muitos erros não são detectados pelo compilador.
- Em linguagens como Java, que têm verificação tipo forte, muitos erros são detectados durante a compilação.
  - Análises mais sofisticadas ainda podem ser úteis, porém!

Referências

- SOMMERVILLE, I. Engenharia de Software. 9ª. Ed. São Paulo: Pearson Education, 2011
  - Capítulo 22

Testes de Software

Centro de Informática - Universidade Federal de Pernambuco
Engenharia da Computação
Kiev Gama
kiev@cin.ufpe.br

Slides elaborados pelo professor Marcio Cornélio

UNIVERSIDADE FEDERAL DE PERNAMBUCO

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

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

Falha, Falta e Erro

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

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
- 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

Informações de um caso de teste

- ID do caso de teste
- Prioridade do teste (Baixa/alta/média)
- Nome do módulo
- Título do caso de teste
- Resumo/descrição do teste
- Pré-condição
- Dependências
- Passo a passo
- Dados de teste
- Resultado esperado
- Pós-condição
- Resultado real
  (única informação preenchida após execução do caso teste)

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

Modelos de falta (fault model)

- Combinações de testes possíveis é muito grande
  - Dificuldade de prever faltas/erros/falhas
- Necessidade de testes baseados em modelos de falta
  - Funciona como hipóteses de falhas explicitando potenciais fontes de erros
- Riscos de faltas identificados com base em error-guessing e suspeitas

O processo de teste

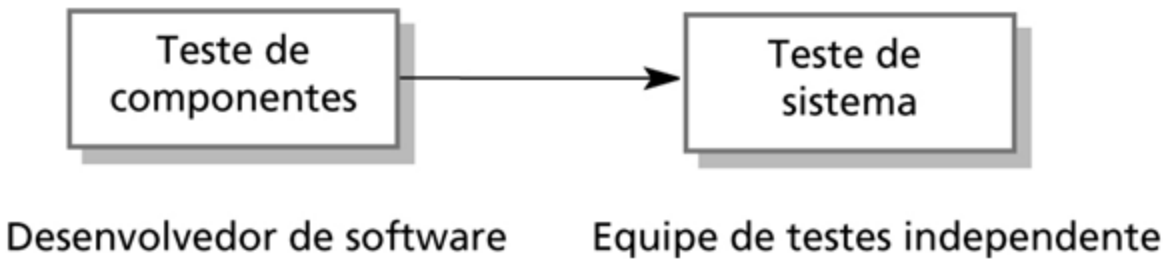
- Teste de componentes
  - Teste de componentes individuais de programa;
  - Geralmente é de responsabilidade do desenvolvedor do componente (exceto algumas para sistemas críticos);
  - Os testes são derivados da experiência do desenvolvedor.

- Teste de sistema
  - Teste de grupos de componentes integrados para criar um sistema ou um subsistema;
  - A responsabilidade 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.

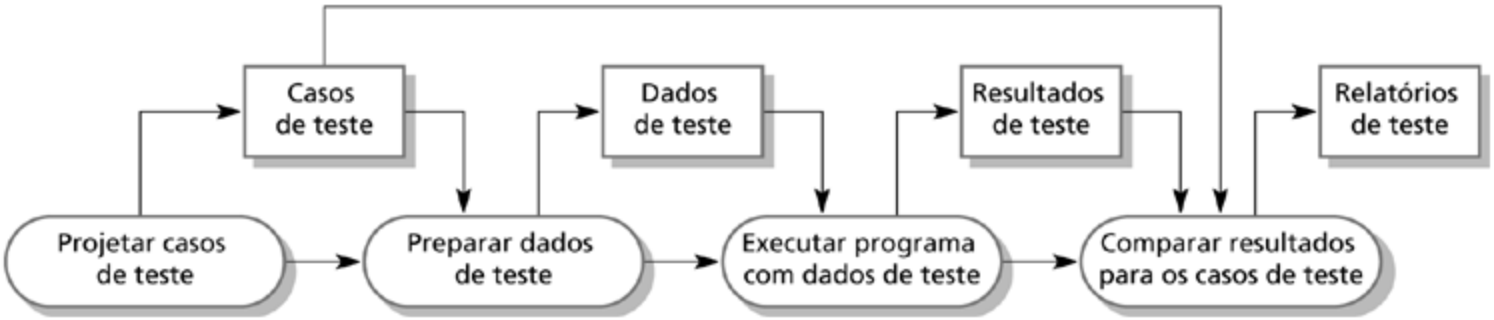


Metas do processo de teste

- 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.
  - Os testes mostram a presença e não a ausência de defeitos

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 virtualmente 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.

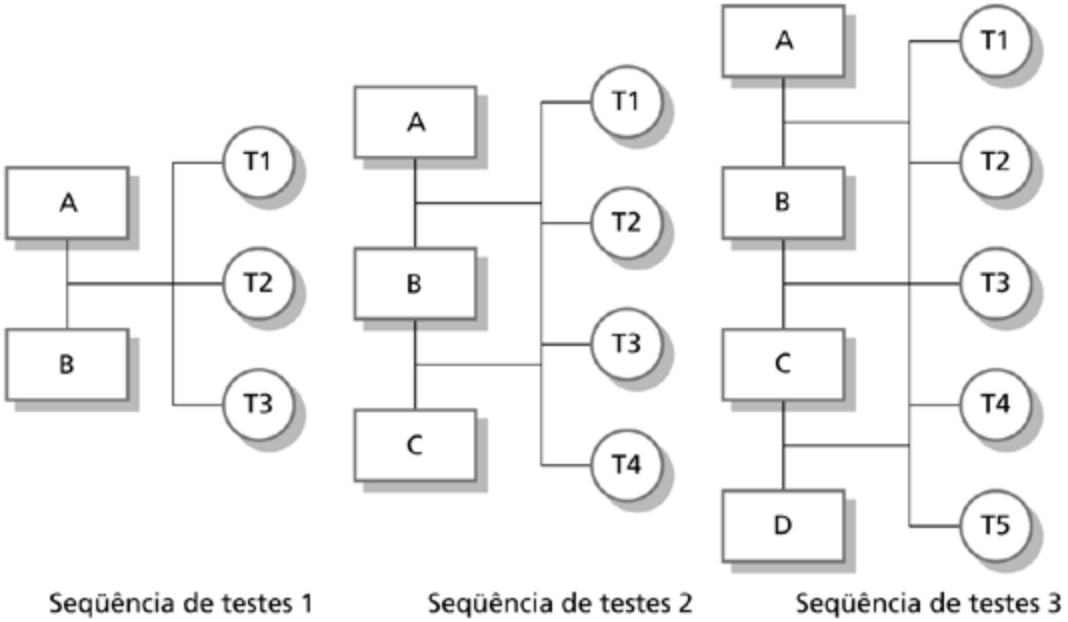
Teste de integração

- 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.

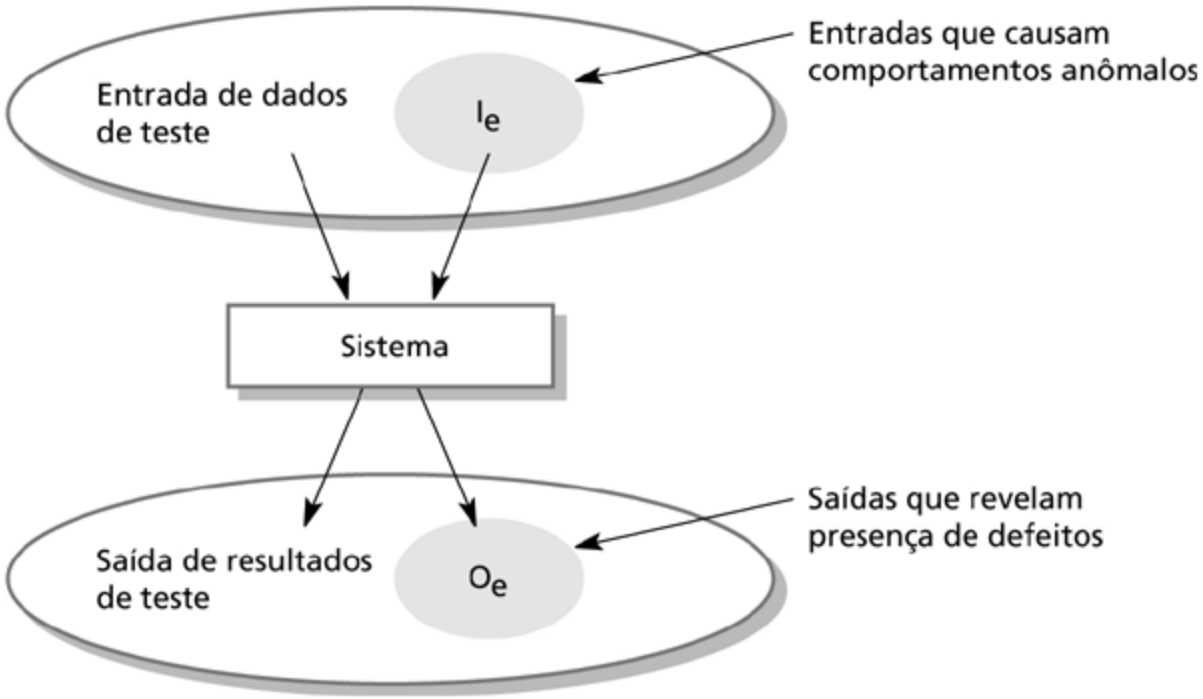
Teste de releases

- 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

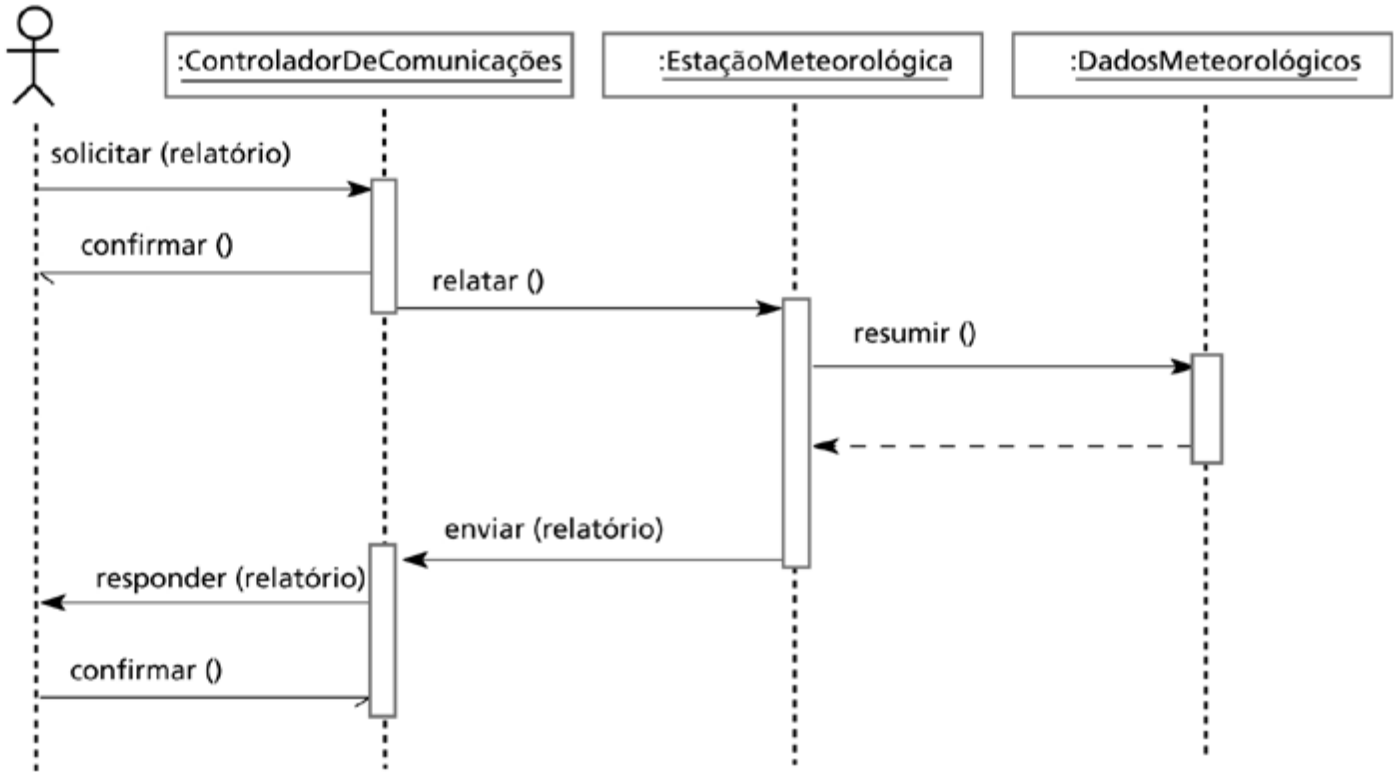
1. Testar o mecanismo de *login* usando *logins* corretos e incorretos para verificar se os usuários válidos são aceitos e os usuários inválidos são rejeitados.
2. Testar o recurso de busca, usando consultas contra fontes conhecidas para verificar se o mecanismo de busca está realmente encontrando documentos.
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 envolvem, geralmente, o planejamento de uma série de testes onde a carga é constantemente aumentada até que o desempenho do sistema se torne inaceitável.
  - Transações 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 proteção 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

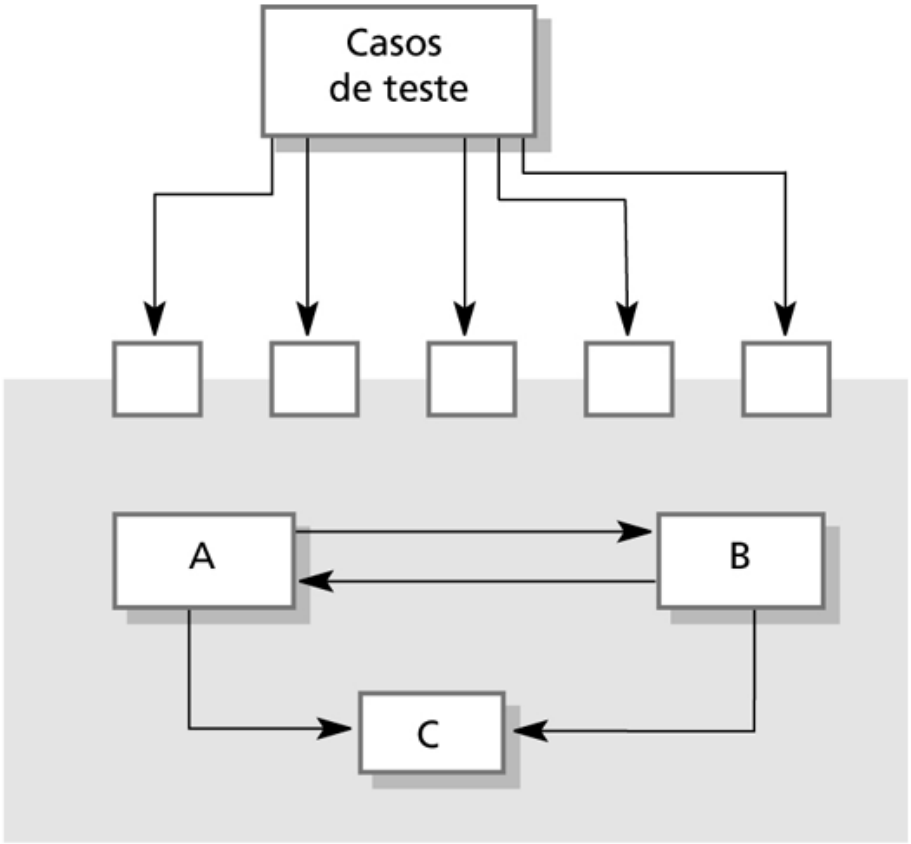
Teste de interfaces

- 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.

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.

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.

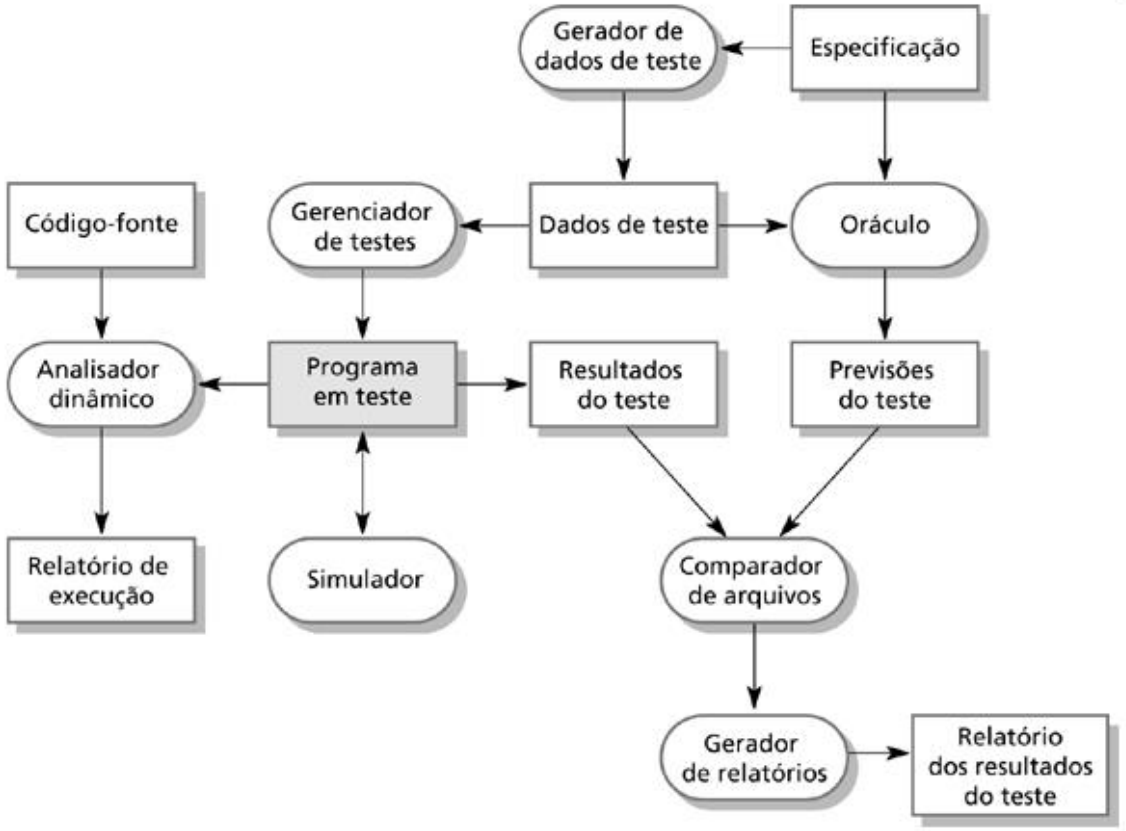
Automação de testes

- 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.
  - Randoop
- Maioria dos workbenches são sistemas abertos a fim de permitir adaptação às necessidades específicas de organizações. Por outro lado, há dificuldade para integrar com ferramentas de análise e projeto que sejam fechadas

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.

Referências

- SOMMERVILLE, I. Engenharia de Software. 9ª. Ed. São Paulo: Pearson Education, 2011
  - Capítulo 23