Testes e Qualidade de Software

Fernando Pedrosa – fpedrosa@gmail.com

Bibliografia

- **Sommerville,** Ian. Software Engineering. **Editora**: Addison Wesley.
- **Pressman**, Roger S. Software Engineering: A Practiotioner's Approach. **Editora**: McGraw-Hill.
- **RUP** – www.wthreex.com/rup
- **IEEE TC FTD/IFIP WG 10.4**

Contexto e Objetivo

- Uma vez que o código fonte tenha sido gerado é necessário testar para descobrir erros antes de entregar o software para o cliente
- O objetivo é projetar uma série de testes com máxima probabilidade de encontrar erros
- São utilizadas técnicas para:
  - Testar a lógica interna dos componentes
  - Testar as entradas e saídas das funções

Quem está envolvido?

- Durante os estágios iniciais do desenvolvimento, um Engenheiro de Software executa todos os testes
- Entretanto, à medida que o processo evolui, especialistas em testes podem ser envolvidos

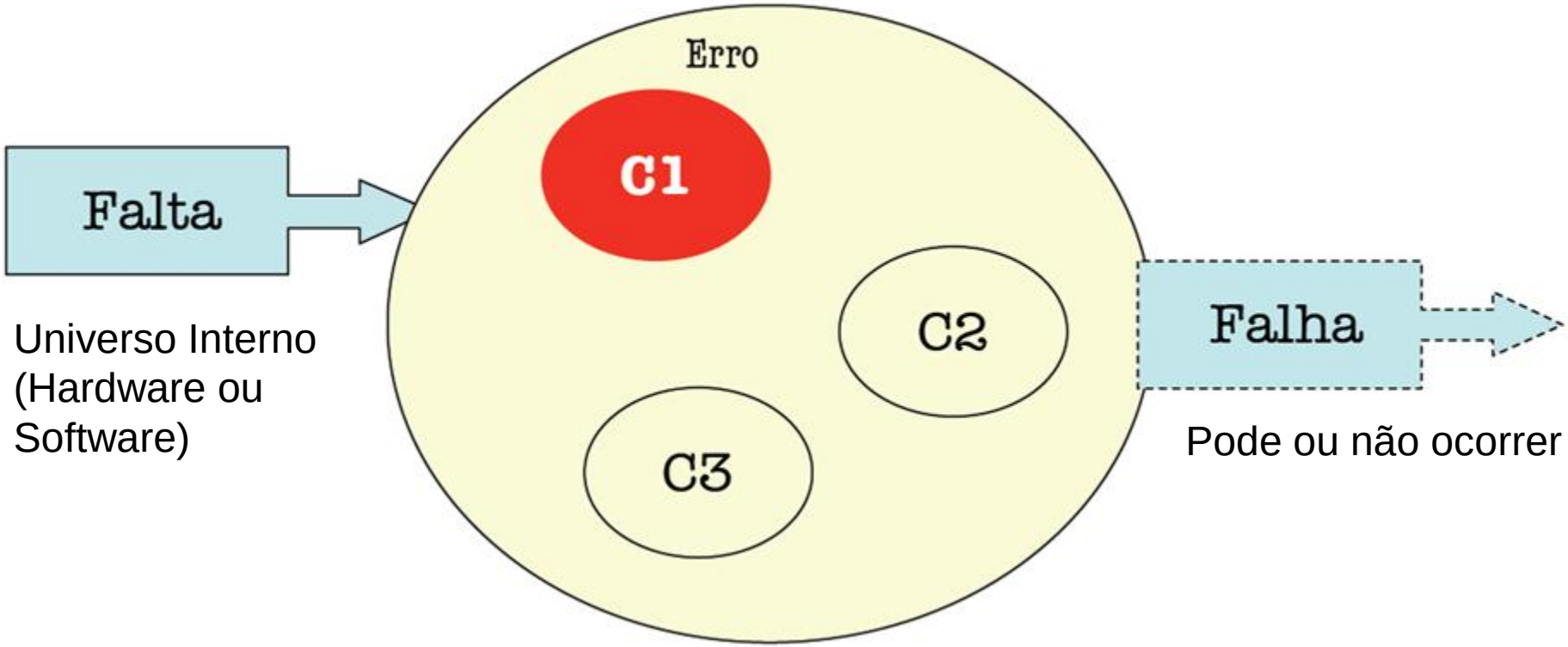
Tipos de Manutenção de Software

- Corretiva
  - Correção de erros encontrados na verificação ou na validação
- Adaptativa
  - Adaptação a mudanças externas
- Melhoria (perfectiva)
  - Melhorias requeridas pelos usuários
- Preventiva ou de reengenharia
  - Abordagem pró-ativa com foco na melhoria da manutibilidade

Falta, Erro e Falha (IEEE)

- **Falta (Fault)**
  - Causa de uma falha (aspecto físico)
  - Exemplo: código incorreto ou faltando, defeito de hardware
- **Erro (Error)**
  - Estado intermediário, de instabilidade (aspecto de informação)
  - Pode resultar em falha, se propagado até a saída
- **Falha (Failure)**
  - Incapacidade do software de realizar a função requisitada (aspecto externo). Manifestação observável.
  - Exemplo: terminação anormal, restrição temporal violada

Falta, Erro e Falha (IEEE)



Universo Interno (Hardware ou Software)

Pode ou não ocorrer

Estado de Instabilidade

Falta, Erro e Falha (IEEE)

Como tornar os sistemas mais confiáveis?

- Prevenção de faltas
  - Especificação rigorosa
  - Proteção de Hardware
  - Ambientes e linguagens apropriados
- Tolerância a falhas
  - Replicação/Redundância
  - Isolamento do componente faltoso
  - *Hot swapping*

Exercícios [1]

(Min. Comunicações - CESPE 2008)

[109] Ao longo do desenvolvimento, artefatos produzidos podem ser revisados, objetivando garantir que os mesmos apresentem, pelo menos, a qualidade mínima especificada. Não apenas o código, mas também outros artefatos podem ser revisados. Os defeitos encontrados pelas revisões referem-se à faltas (fault), enquanto os defeitos encontrados por testes são falhas do software, pois testes avaliam a qualidade comparando o comportamento esperado com o observado.

Exercícios [1]

(TRE/BA – CESPE 2010)

[61] Segundo o IEEE, defeito é um ato inconsistente cometido por um indivíduo ao tentar entender determinada informação, resolver um problema ou utilizar um método ou uma ferramenta; erro é o comportamento operacional do software diferente do esperado pelo usuário, e que pode ter sido causado por diversas falhas; e falha é uma manifestação concreta de um defeito em um artefato de software, ou seja, é qualquer estado intermediário incorreto ou resultado inesperado na execução de um programa.

Verificação e Validação

- É todo um processo de ciclo de vida que deve ser aplicado em cada estágio do desenvolvimento do software
- Tem dois objetivos principais:
  - O descobrimento de defeitos no sistema
  - A avaliação sobre se o sistema é útil e adequado em uma situação operacional
- V&V não garante que o software está livre de defeitos, mas apenas garante um certo nível de confiabilidade

Verificação

- Refere-se ao conjunto de atividades que garantem que o software implementa corretamente as funções especificadas

  “Estamos construindo o produto de forma correta?”

  “Are we building the Product Right?”

- Principais atividades
  - Inspeções (verificação estática)
  - Testes (verificação dinâmica)

Validação

- Refere-se ao conjunto de atividades que garantem que o software construído implementa o que o cliente realmente desejava

  “Estamos construindo o produto certo?”

  “Are we building the Right Product?”

- Principais atividades
  - Homologação, Testes de Aceitação (beta)
  - Revisões, etc.

Exercícios [2-A]

(MPOG – ESAF 2008)

[13-B] Demonstrar ao desenvolvedor e ao cliente que o software atende aos requisitos é uma meta de validação do software.

(IPEA – CESPE 2008)

[85] A verificação assegura que o produto, como fornecido, irá atender o seu uso pretendido, ou seja, que se está construindo o produto certo. E a validação confirma que os produtos de trabalho refletem de forma apropriada os requisitos que foram especificados, ou seja, que se está construindo o produto corretamente.

(TRT5 – CESPE 2009)

[75] A diferença entre verificação e validação reside no fato de que a primeira se refere ao conjunto de atividades que garante que o software realiza corretamente uma função específica, enquanto a segunda refere-se a um conjunto diferente de atividades que garante que o software que foi construído é rastreável às exigências do cliente.

Exercícios [2-A]

(MPU – FCC 2007)

[56] Considere as informações abaixo em relação ao desenvolvimento de sistemas:

I. executar um software com o objetivo de revelar falhas, mas que não prova a exatidão do software.

II. correta construção do produto.

III. construção do produto certo.

Correspondem corretamente a I, II e III, respectivamente,

a) validação, verificação e teste.

b) verificação, teste e validação.

c) teste, verificação e validação.

d) validação, teste e verificação.

e) teste, validação e verificação

Qualidade: Garantia x Controle (Padrão de mercado)

Garantia da Qualidade	Controle da Qualidade
Focada no processo	Focada no produto
Orientada a prevenção	Orientada a detecção
Exemplos: metodologias e padrões de desenvolvimento	Exemplos: checagem de requisitos, testes de software, etc.
Garante que você está fazendo as coisas da maneira correta	Garante que os resultados do seu trabalho estão de acordo com o esperado

Qualidade: Garantia x Controle (Sommerville & Pressman)

- Alguns autores não diferenciam estas atividades
- Para Sommerville inclui Validação e Verificação de Produtos, além dos Processos adequados
- Para Pressman inclui um espectro amplo de atividades, tais como padronização, revisões, testes, etc.

Exercícios [2-B]

(SERPRO - CESPE 2010)

[98] A garantia de qualidade tem como objetivo testar os produtos de software de modo a identificar, relatar e remover os defeitos encontrados, enquanto o controle da qualidade provê a gerência sênior da organização com a visibilidade apropriada sobre o processo de desenvolvimento.

(STJ – CESPE 2008)

[97] Um processo de gerenciamento da qualidade do projeto tipicamente visa garantir e controlar a qualidade. No controle da qualidade, são executadas atividades planejadas e sistemáticas visando garantir que o projeto empregará os processos necessários para atender aos requisitos. Por sua vez, a garantia da qualidade, diferentemente do controle de qualidade, monitora resultados do projeto a fim de determinar se eles estão de acordo com os padrões relevantes de qualidade e procura identificar meios para eliminar as causas de resultados que sejam insatisfatórios.

Inspeções de Software (verificação estática)

Inspeção de Software

- Se preocupa com a análise estática dos artefatos do sistema
- Envolve pessoas examinando os produtos de trabalho com o objetivo de encontrar anomalias e defeitos
- Pode ser suplementada por ferramentas CASE de análise estática

Inspeção x Teste

- Inspeções e Testes são técnicas complementares e não opostas
- Muitos defeitos podem ser encontrados durante uma inspeção – testes podem mascarar erros e necessitar de várias execuções
- Inspeções não podem checar requisitos não funcionais (desempenho, usabilidade, etc.)
- Inspeções checam a conformidade com a especificação e não com as reais necessidades do cliente

Vantagens

- Inspeções não necessitam da execução do sistema, portanto podem ser aplicadas antes da implementação
- Podem ser aplicadas a qualquer representação do sistema
  - Requisitos
  - Projeto,
  - Código, etc.
- Têm se mostrado uma técnica efetiva para se descobrir erros no programa

Dificuldades

- Inspeções aumentam o custo do processo de desenvolvimento, no início
- As equipes devem ser bem informadas e ter acesso a especificações precisas
- Padrões organizacionais devem ser bem definidos
- A gerência pode utilizar os achados de inspeção para avaliar (culpar) indivíduos

Inspeções do programa

- São abordagens formais para documentar as revisões do código
- A intenção é de **detecção de defeitos, apenas** (e não correção)
- Defeitos podem ser
  - Lógicos (caminhos incorretos, cálculos errados, etc.)
  - Anomalias no código (variáveis não inicializadas, laços incompletos, etc.)
  - Não conformidade com padrões

Inspeções manuais do programa

Procedimento:

- Um resumo do sistema é apresentado à equipe de inspeção
- Código e documentos associados são distribuídos à equipe com antecedência
- A inspeção acontece e os erros descobertos são anotados, de acordo com um *checklist*
- Uma nova inspeção pode ser necessária

Inspeções automatizadas

- Inspeções de código automatizadas utilizam **Analisadores Estáticos**
  - São ferramentas CASE que analisam o código fonte do sistema
  - Buscam e relatam erros em potencial
  - São bastante efetivos como um auxílio às inspeções, mas não a substituem
- Têm um melhor custo-benefício para linguagens fracamente tipadas, onde o compilador detecta poucos erros

Inspeções automatizadas

- Exemplos de possíveis análises:
  - Análise de controle de fluxo
    - Checa a codificação de loops, código inalcançável, etc.
  - Análise de dados
    - Detecta variáveis não inicializadas, variáveis que nunca são utilizadas, etc.
  - Análise de interfaces
    - Checa a consistência de declarações de rotinas e procedimentos e como são utilizados

Exercícios [3]

(STJ – CESPE 2008)

[73] Inspeções e walkthroughs podem fazer parte de um processo de verificação e validação, sendo realizadas por equipes cujos membros têm papéis definidos. Quando da inspeção de um código, uma lista de verificação de erros (checklist) é usada. O conteúdo da lista tipicamente independe da linguagem de programação usada.

(TSE – CESPE 2006)

[63-A] Inspeções e walkthroughs podem ser usadas para revisar artefatos. Uma walkthrough requer mais tempo de preparação dos revisores do que uma inspeção, também exige que seja feito o acompanhamento das soluções dos problemas identificados e a coleta de métricas associadas à revisão.

Exercícios [3]

(TSE – CESPE 2006)

[63-B] Em uma inspeção, os participantes têm papéis definidos. O moderador conduz reuniões e os inspetores devem, durante as reuniões, descrever os problemas identificados e soluções para os mesmos.

(DETRAN – CESPE 2009)

[97] O processo de validação tem por objetivo estabelecer com os clientes confiança quanto ao funcionamento adequado de um software. Enquanto inspeções de software ou revisões por pares são consideradas validação estática, o teste consiste em uma técnica dinâmica de validação de software. Os termos estático ou dinâmico são relativos à necessidade ou não do software ser executado.

Testes
(verificação dinâmica)

Teste

- Processo de executar um programa com o **objetivo de encontrar erros**.
- Um bom caso de teste é aquele que tem alta probabilidade de achar um erro ainda não descoberto.
- Um teste de sucesso é aquele que descobre um erro ainda não descoberto.

O processo de testes **não** pode garantir que o software está livre de erros, ele pode apenas mostrar que erros e defeitos de software estão presentes.

Princípios

- Testes devem ser rastreáveis aos requisitos do cliente
- Testes devem ser planejados muito antes de serem executados
- O princípio de Pareto se aplica a Testes
  - 80% dos erros vão acontecer em 20% dos componentes
- Testes devem começar “pequenos” e progredir para pedaços maiores

Princípios

- Testes exaustivos não são possíveis
  - É impossível executar todos os caminhos existentes em um programa
- Para terem o máximo de efetividade, testes devem ser, preferencialmente, conduzidos por terceiros

O que constitui um “bom” teste?

- Um bom teste deve ter alta probabilidade de encontrar erros
- Um bom teste não deve ser redundante
  - Todo teste deve ter um propósito diferente
- Um bom teste deve ser “representativo” na sua classe
- Um bom teste não deve ser nem muito simples nem muito complexo [Pressman]

Abordagens de Testes

Abordagens de Testes

- Abordagem Funcional (“caixa preta”)
  - Focada nas entradas e saídas especificadas nos requisitos funcionais
- Abordagem Estrutural (“caixa branca”)
  - Focada nas estruturas internas dos procedimentos do sistema
- Abordagem Mista (“caixa cinza”)
  - É um meio termo entre caixa preta e branca
  - Algum conhecimento sobre as estruturas internas é utilizado para executar testes mais “bem informados”

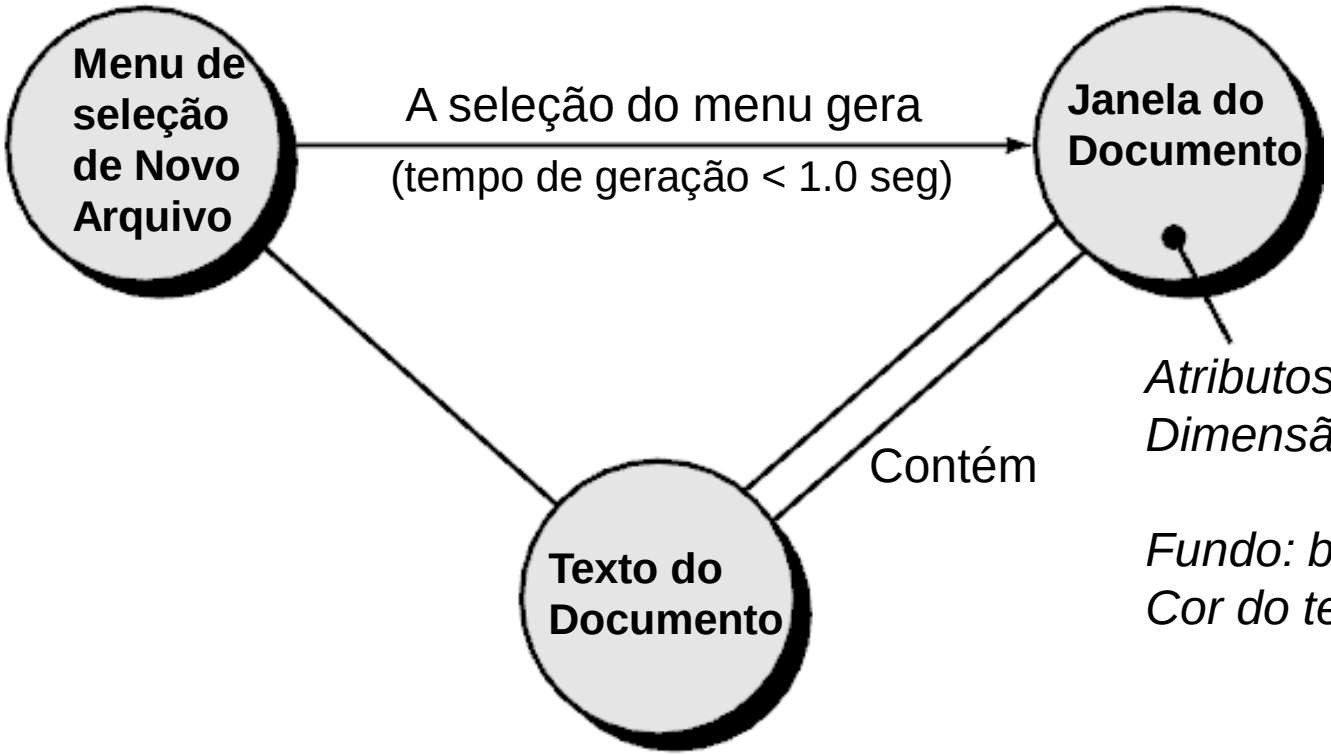
Abordagem Caixa Preta

- Baseada em prés e pós condições
- Geralmente utilizada nas etapas posteriores da disciplina de testes
- Busca erros nas seguintes categorias (dentre outras):
  - Funções incorretas ou inexistentes
  - Erros de comportamento ou desempenho
  - Erros de inicialização e término, erros de interface
- Principais técnicas: testes baseados em grafos, matriz ortogonal, análise de valores limítrofes, particionamento de equivalências

Testes baseados em grafos

- *Graph-based testing methods*
- Toda aplicação é construída por "objetos"
- A técnica identifica todos estes objetos e gera gráficos para representá-los
- Os objetos e relacionamentos são testados para descobrir erros e comportamentos inesperados

Testes baseados em grafos (exemplo: editor gráfico)



Particionamento de Equivalências

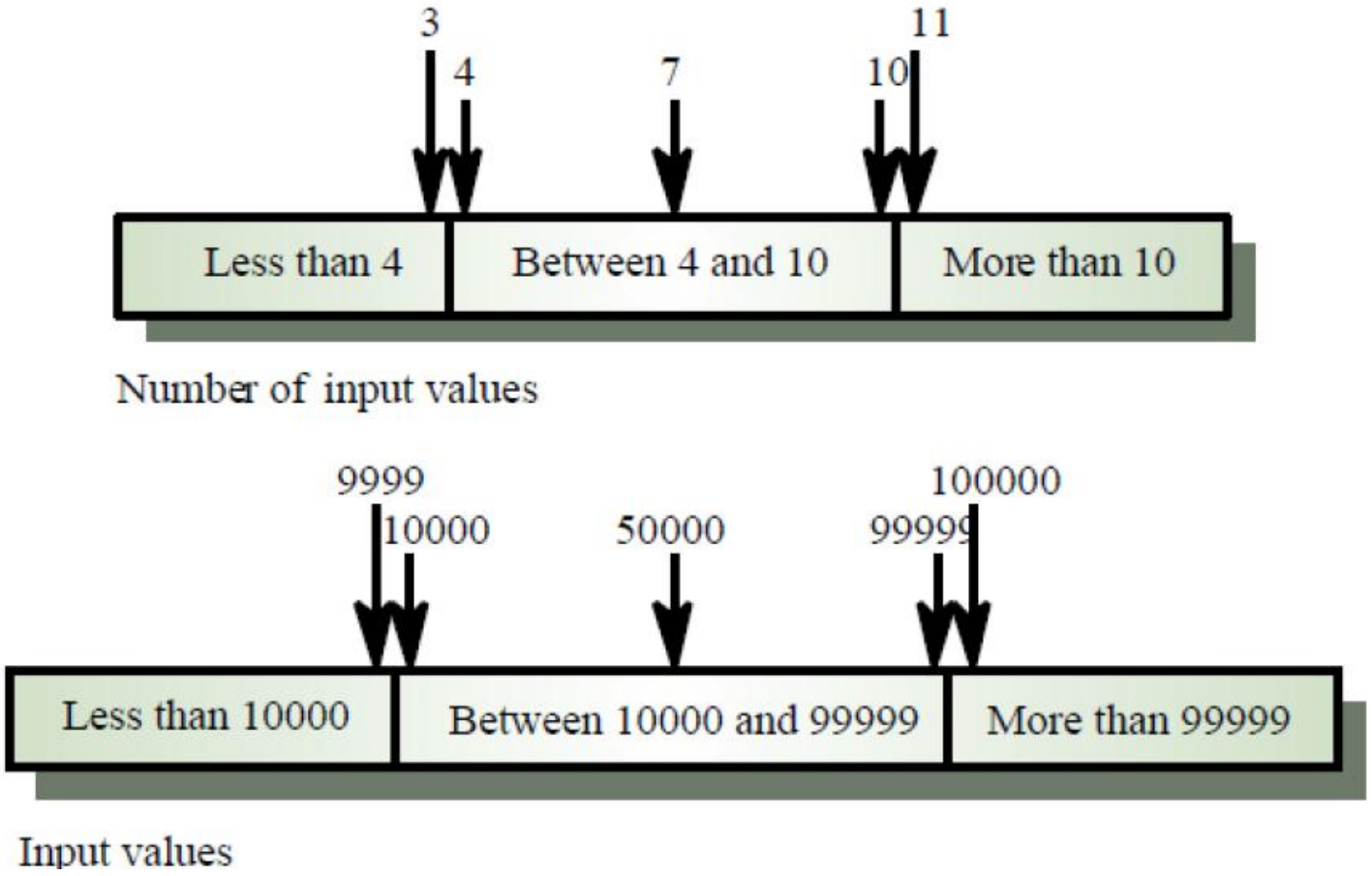
- Técnica baseada em dividir o domínio de entradas de um programa em classes de dados
- Cada classe de dados é chamada de "partição de equivalência"
- Casos de Teste são gerados considerando cada partição de equivalência

Particionamento de Equivalências

- Como guia, deve-se considerar entradas do tipo:
  - Faixa de Valores
    - 1 partição válida, 2 inválidas
  - Valor específico
    - 1 partição válida, 2 inválidas
  - Conjunto específico de valores
    - 1 partição válida, 1 inválida
  - Entrada *booleana*
    - 1 partição válida, 1 inválida

Particionamento de Equivalências

Exemplos (faixas de valores)



Análise de valores limítrofes

- É sabido que a maioria dos erros acontece nos limites do domínio de entrada, e não no “centro”
- Os testes devem ser gerados considerando esses valores “limítrofes”
  - Números máximos e mínimos
  - “Um a mais do que o máximo”
  - “Um a menos do que o mínimo”
- É utilizada em conjunto com a técnica de Particionamento de Equivalência

Exercícios [4]

(TRE/BA – CESPE 2010)

[79] Teste funcional é uma técnica para se projetar casos de teste na qual o programa ou sistema é considerado uma caixa-preta e, para testá-lo, são fornecidas entradas e avaliadas as saídas geradas.

(CEF – CESPE 2010)

[57-C] Relativamente ao modelo de caixa-cinza, o teste de software caixa-preta apresenta maior dependência no trabalho entre implementador e testador.

(CEF – CESPE 2010)

[57-D] O teste de software caixa-preta, relativamente ao modelo de caixa-cinza, apresenta maior dependência de conhecimento a respeito dos programas-fontes a serem testados.

Exercícios [4]

(TRT5 – CESPE 2008)

[74] Entre os tipos de testes de caixa preta, encontram-se o teste baseado em grafos; o particionamento de equivalência; a análise de valor-limite; e o teste de matriz ortogonal.

(MPOG – ESAF 2008)

[13-C] o particionamento de equivalência é uma maneira estratégica de aplicar testes de software.

Abordagem Caixa Branca

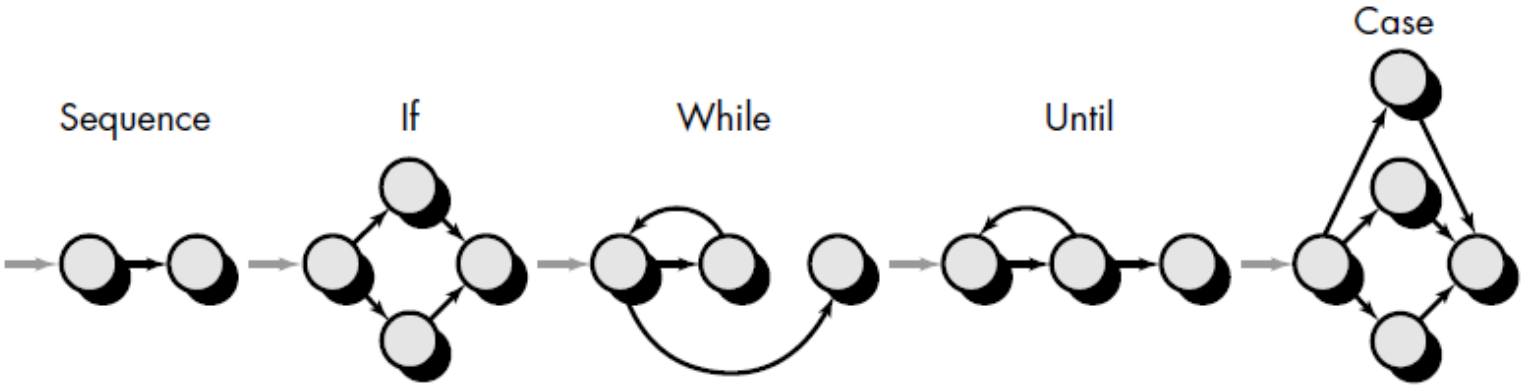
- Os testes são gerados a partir de uma análise dos caminhos lógicos possíveis de serem executados
- É necessário conhecimento do funcionamento interno dos componentes do software
- Objetivos
  - Garantir que todos os caminhos independentes de um módulo sejam executados pelo menos uma vez

Abordagem Caixa Branca

- Objetivos (continuação)
  - Realizar todas as decisões lógicas para valores falsos e verdadeiros
  - Executar laços dentro dos valores limites
  - Avaliar as estruturas de dados internas
- Principais técnicas
  - Testes de caminhos
  - Testes de estruturas de controle (laços, estruturas condicionais, etc.)
  - Complexidade ciclomática (métrica)

Complexidade Ciclomática

- Métrica que fornece uma medida quantitativa da complexidade lógica de um programa
- Quando usada no contexto de testes caixa branca, denota o número de caminhos possíveis dentro de um módulo
  - Nos dá uma idéia do limite superior necessário!



Exercícios [5]

(SERPRO – CESPE 2010)

[85] Com relação ao emprego de técnicas para a realização de testes de software, é correto afirmar que haverá maior diminuição da dependência de acesso às especificações arquiteturais de um sistema se o testador empregar a técnica de caixa-branca (white-box), em vez das técnicas de caixa-cinza (gray-box) e de caixa-preta (black-box).

(CEF – CESPE 2010)

[57-B] O aumento na medida de complexidade ciclomática de um programa introduz mudanças significativas no refinamento de uma abordagem do tipo caixa-preta.

Exercícios [5]

(CEF – CESPE 2010)
[57-E] À medida que avança o nível de integração dos módulos de um software, mais viável se torna a adoção do método de caixa-branca para desenho do teste de software.

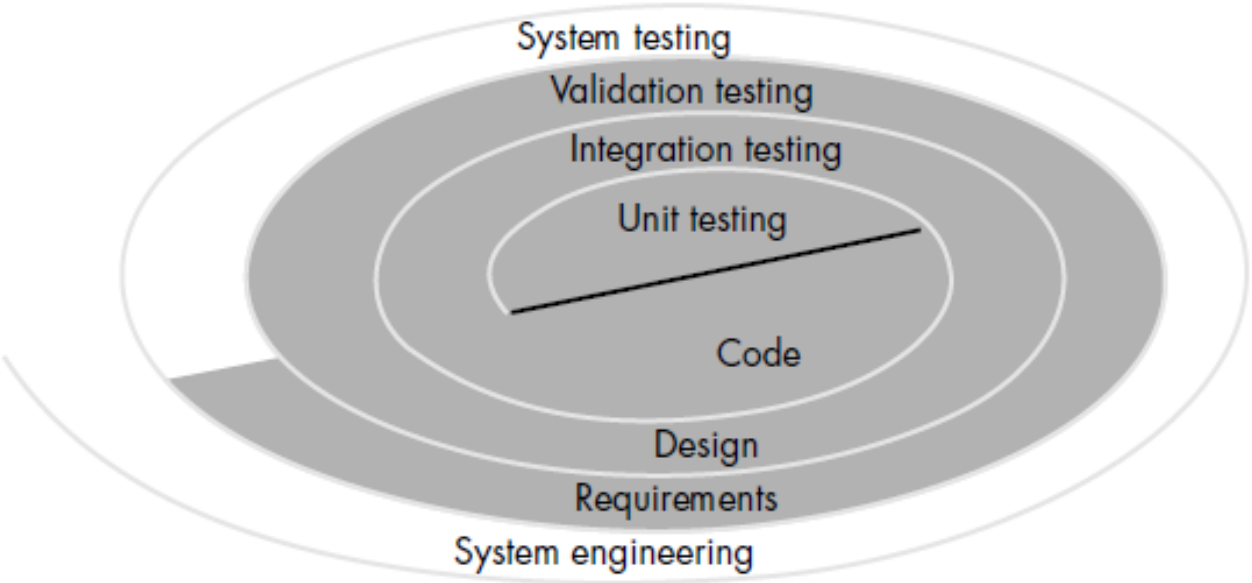
(CEF – CESPE 2010)
[57-A] Para aderência à abordagem de software caixa-branca, podem ser empregados testes de fluxo de controle e de dados, que não são apoiadores diretos em testes de caixa-preta.

(MPOG – ESAF 2008)
[13-D] O teste estrutural é uma estratégia que se baseia na análise da especificação de um programa para ajudar na seleção de casos de teste.

Estágios (estratégias) de Testes

Estágios (estratégias) de Teste

- Os testes geralmente são agrupados de acordo com o momento no qual eles são executados ou pelo nível de especificidade do teste



Teste de Unidade

- Primeiro nível de testes, onde componentes individuais são testados para assegurar que os mesmos operam de forma correta
- Componentes podem ser:
  - Métodos individuais dentro de um objeto
  - Classes com vários atributos e métodos
  - Componentes que tenham sua estrutura interna bem conhecida
- Ferramenta mais utilizada (Java): JUnit

JUnit

- Framework de código aberto, que provê o ambiente necessário para testar códigos em Java
- A maioria das IDEs (Eclipse, Netbeans, JDeveloper, etc.) incorpora-o dentro do seu ambiente, tornando o uso mais fácil

Componentes de um teste JUnit

- Classe a ser testada
  - `Calculadora.java`
- Classe contendo os casos de teste
  - É quem realmente executa os testes na classe a ser testada
  - `CalculadoraTest.java`
- Chamador (driver) dos testes
  - Chama as classes contendo os testes
  - Normalmente é utilizado automaticamente através das IDE's

Exemplo

Classe a ser testada

```
public class Calculadora {

    public static int somar (int a, int b) {
        return a + b;
    }

    public static int multiplicar (int a, int b) {
        return a * b;
    }
    ...
}
```

Classe contendo os casos de teste

```
import junit.framework.*;

public class CalculadoraTest extends TestCase {

    public void testSomar() {
        int a = 1;
        int b = 2;
        int total = 3
        int resultado = 0;
        resultado = Calculadora.somar(a,b);
        assertEquals(resultado,total);
    }
```

Assertiva

Exercícios [6]

(PETROBRAS – CESGRANRIO 2010)

[13-A] A utilização de testes unitários faz com que seja desnecessária a fase de testes de aceitação, pois é garantido que o software passou por todos os testes de funcionamento necessários.

(PETROBRAS – CESGRANRIO 2010)

[13-B] A utilização de testes unitários ajuda a verificar se alterações introduzidas por um dos membros de uma equipe de tamanho médio ou grande não causaram efeitos colaterais em módulos de outros membros da equipe.

Exercícios [6]

(SERPRO – CESPE 2010)

[83] A atividade de teste unitário de software é, conforme os modelos de ciclo de vida de software vigentes, realizada de forma mais eficaz no escopo de implementação e da construção de software – nas quais a codificação de uma unidade executável de software é feita –, quando comparada à situação em que o teste unitário é realizado simultaneamente ao teste de integração.

(TRE/MT – CESPE 2010)

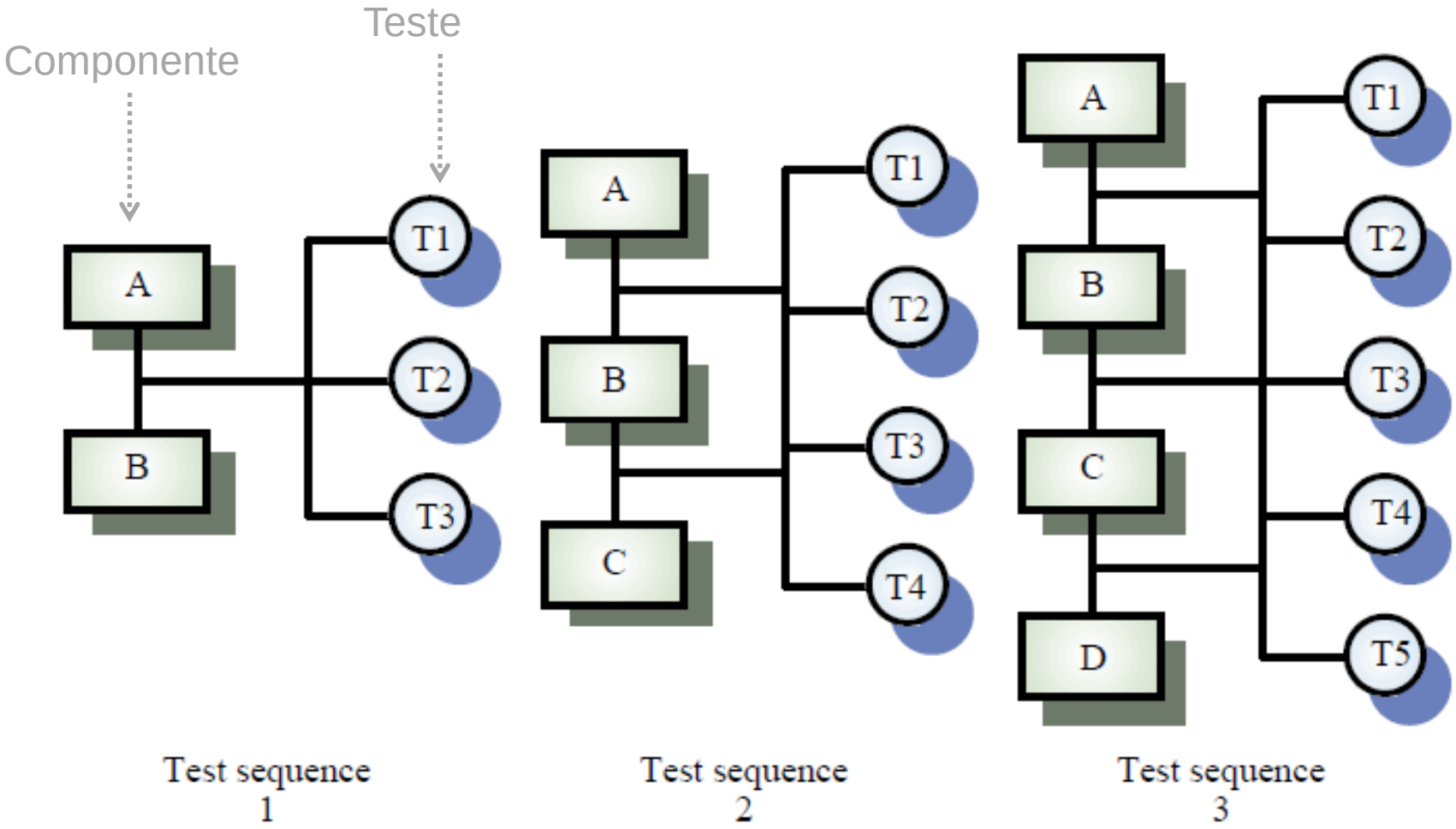
[40-A] JUnit é um framework open-source (arcabouço livre) para escrever e executar testes automaticamente, sem necessidade de escrever código adicional.

[40-C] A classe AssertEquals(a,b) compara dois valores. O teste é executado com sucesso se a.equals(b).

Teste de Integração

- Integra e testa os componentes de um sistema com o objetivo de encontrar problemas durante suas interações
- Integração Top-Down
  - Desenvolve o esqueleto do sistema e o preenche com os componentes do sistema
- Integração Bottom-Up
  - Integra os componentes de infraestrutura e depois adiciona componentes funcionais
- Para facilitar a localização de erros, sistemas devem ser integrados gradualmente

Integração gradual



Exercícios [7]

(IPEA – CESPE 2008)

[58] Os testes de integração verificam se os componentes do sistema funcionam em conjunto, se os componentes são chamados corretamente e se os componentes transferem dados corretos via suas interfaces. Nesses testes, os componentes são testados interligados; podem ser necessários drivers e stubs para simular componentes ainda não implementados; e, em sistemas de software orientados a objeto, os stubs podem ser classes.

(TRE/PR – CESPE 2009)

[83] Nos testes de integração, realizados antes dos testes unitários, os componentes são construídos e testados separadamente.

Teste de Aceitação (Validação)

- Ao final dos testes de integração, o software está consolidado e iniciam-se os testes de aceitação
- A finalidade é demonstrar a conformidade com os requisitos de software
- O ambiente utilizado deve ser o mais próximo possível do ambiente real
- Os mais comuns são Testes Alfa ou Beta

Teste Alfa

- É virtualmente impossível para o desenvolvedor prever como o software será utilizado pelo usuário
- São necessários testes conduzidos pelo cliente nas instalações do desenvolvedor
- O desenvolvedor anota os erros e problemas que possam ocorrer
- Acontece em ambiente **controlado**

Teste Beta

- É conduzido em um ou mais locais do cliente, pelo usuário final do produto
- O desenvolvedor geralmente não está presente
- O usuário anota todos os problemas que possa encontrar e os envia para o desenvolvedor frequentemente
- Acontece em ambiente **real**

Teste de Sistema

- Software é apenas um elemento de todo um sistema computacional maior
- É necessário testá-lo no ambiente operacional, onde são considerados
  - Hardware e Pessoas
  - Processos e informações
  - Outros sistemas, etc.
- São conduzidos em um ambiente completo e integrado, por várias pessoas (não só os desenvolvedores)

Exercícios [8]

(TRE/MT – CESPE 2010)
[39-D] O teste alfa é conduzido pelo cliente em seu ambiente de uso final.

(TSE – CESPE 2006)
[55-C] Os testes são realizados em várias fases de um desenvolvimento. Testes de unidade são de baixo nível, testes de sistema são executados após os de integração, testes beta empregam apenas desenvolvedores.

(EPE – CESGRANRIO 2010)
[43] Um novo sistema de informação interno de uma empresa está sendo testado por um grupo restrito de usuários, fora do ambiente dos desenvolvedores. Isso caracteriza o teste
(A) de unidade. (B) de usabilidade.
(C) alfa. (D) beta.
(E) de stress.

Tipos de Testes

Teste de Regressão

- Cada vez que um módulo é adicionado ao sistema, o software muda
  - Novas entradas e saídas surgem
  - Podem ser executados novos caminhos
  - A lógica de controle muda
- Teste de regressão visa a executar um subconjunto de testes que já foram executados com o intuito de garantir que as mudanças não propagaram efeitos indesejados

Smoke Testing (Teste de Fumaça)

- O termo é usado em várias áreas e refere-se ao primeiro teste realizado depois de integrar os componentes
- Em Software, é aplicado após cada montagem (build) do produto para verificar sua funcionalidade básica
- O intuito deve ser o de encontrar erros que têm a maior probabilidade de atrasar o projeto (*show stopper errors*)

Teste de Recuperação

- Muitos sistemas devem se recuperar de falhas e voltar ao processamento dentro de um tempo pré-estabelecido
- Teste de recuperação força o software a falhar de várias formas e verifica que a recuperação foi feita de forma adequada
- A recuperação pode ser automática ou manual

Teste de Segurança

- Tem o objetivo de verificar que mecanismos de proteção implementados no sistema irão, de fato, protegê-lo de ataques
- O testador tenta penetrar no sistema
- Dado tempo suficiente, um teste de segurança irá penetrar o sistema
  - É papel do desenvolvedor do sistema assegurar que isto custe mais caro que os ganhos obtidos

Teste de Carga (estresse)

- Inicialmente, testes caixa branca e caixa preta tem o intuito de testar o sistema sob condições normais
- Testes de carga visam a confrontar programas com situações anormais
  - Têm caráter destrutivo
  - Até onde ele aguenta?
- Podem ser estressados
  - Memória (vários objetos criados)
  - I/O (muitas interrupções por segundo)
  - Disco (busca por dados), etc.

Teste de Desempenho

- Pode ocorrer durante todos os estágios de testes
- Visa a garantir que o sistema atende aos níveis de desempenho e tempo de resposta acordados com o usuário e definidos nos requisitos

Teste de Usabilidade

- Avalia o sistema do ponto de vista do usuário final
- Enfatiza o seguinte:
  - Fatores humanos
  - Estética
  - Consistência na interface do usuário
  - Ajuda online e contextual
  - Assistentes e agentes (wizards)
  - Documentação do usuário
  - Material de treinamento

Exercícios [9]

(PRODEST – CESPE 2006)

[101] O XP é um processo que visa a um desenvolvimento ágil e portanto não recomenda os testes de unidade, pois eles consomem muitos recursos. Durante o desenvolvimento, o primeiro teste recomendado é o smoke test que foca os detalhes de funcionamento. O smoke test é realizado após as unidades serem integradas. Após o smoke test, é realizado o teste de sistema.

(TRT16 – FCC 2009)

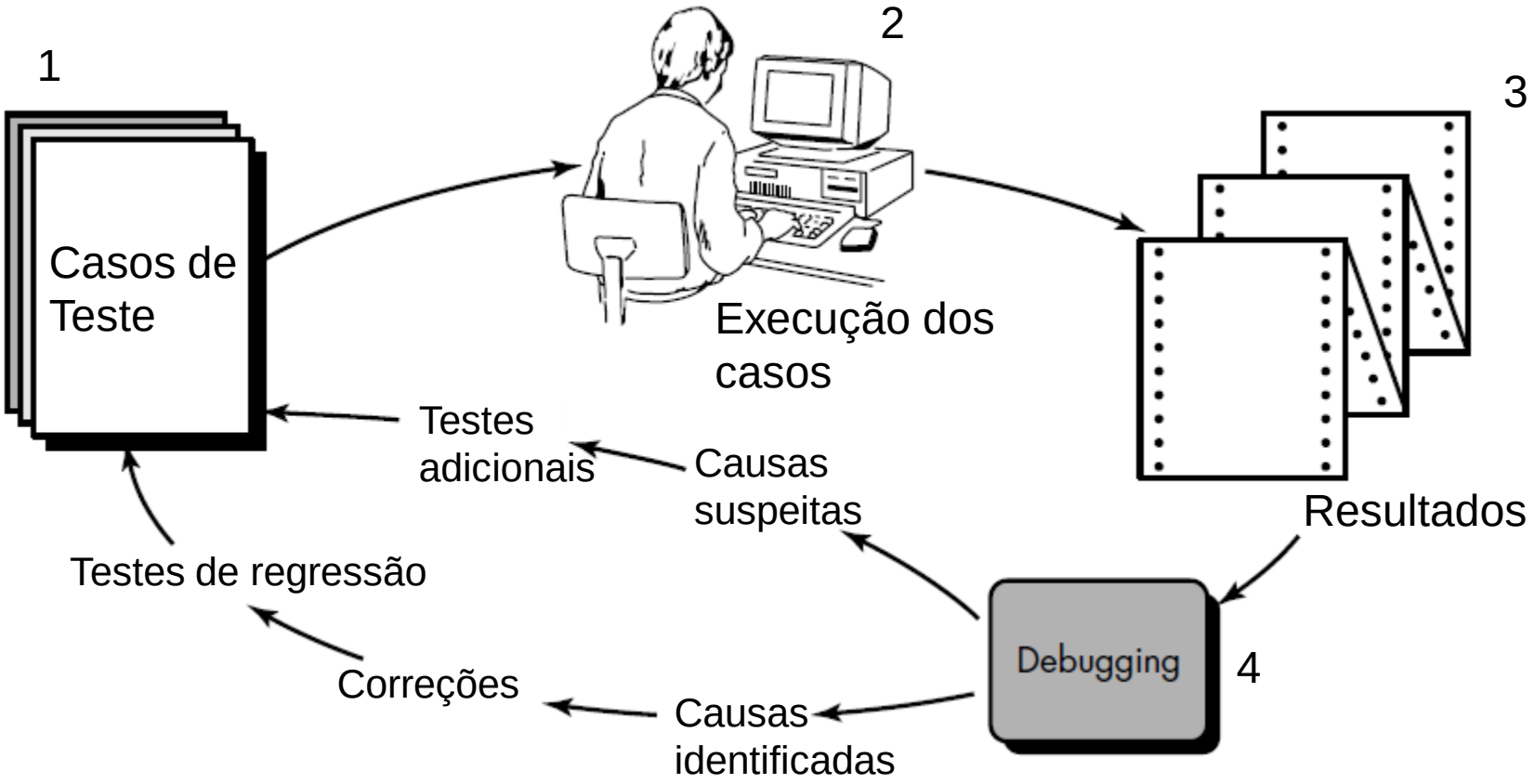
[48] Há um tipo de teste que vislumbra a "destruição do programa" por meio de sua submissão a quantidades, frequências ou volumes anormais que é o teste

(A) de recuperação. (B) de configuração. (C) beta. (D) de desempenho. (E) de estresse.

Debugging

- É o processo que resulta na remoção de um erro encontrado
- Ocorre como uma consequência de um teste de sucesso (um teste que encontrou erros)
- Envolve formular uma hipótese sobre o comportamento do sistema e testar essa hipótese para achar os erros

O processo de debugar



Abordagens de debugging

- Força Bruta
  - Popula-se o sistema com escritas ao console para tentar encontrar algum traço de erro durante a execução
  - É o método menos eficiente de todos
- *Backtracking*
  - Começando a partir de onde o erro ocorreu, deve-se rastrear o código manualmente até a fonte do erro
- Eliminação de causa
  - Uma "hipótese de causa" é elaborada e os dados relacionados ao erro são utilizados para prová-la

Exercícios [10]

(TRE/MT - CESPE 2010)

[43] Existem várias maneiras de se depurar (debug) programas. Algumas delas envolvem conhecimento, prática e bom senso do programador. Acerca de pontos que são importantes para depurar programas, julgue os itens a seguir.

I É possível encontrar falhas nos programas por meio da reprodução do erro em testes.

II Quanto maior a entrada de dados nos testes, mais simples é encontrar o problema e mais fácil é encontrar a solução da falha.

III Em um programa modular, o processo de encontrar falhas requer uma menor variação de informações de entrada, de modo que o programador possa encontrar o módulo com erros.

IV A passagem de parâmetros para variáveis auxiliares evita o uso de Break points.

Exercícios [10]

V A análise estruturada é a melhor maneira de encontrar erros em programação orientada a objetos.

Estão certos apenas os itens

A) I e II.
B) I e III.
C) II e V.
D) III e IV.
E) IV e V.

Testes segundo o RUP

Plano de Testes

- Define metas e objetivos dos testes no escopo da iteração (ou projeto)
- Explicita a abordagem adotada, os recursos necessários e os produtos liberados
- Determina a estrutura (framework) na qual os papéis de testes funcionarão
- Assim como em outros documentos, é necessário ganhar a aprovação e aceitação dos envolvidos

Caso de Teste

- Tem a finalidade de identificar e comunicar as condições específicas nas quais as funcionalidades serão testadas
- Define um conjunto de:
  - Entradas de teste
  - Condições de execução
  - Resultados esperados

Estrutura do Caso de Teste

- Descrição do Caso de Teste
  - Descreve o objetivo e o escopo do teste

Condições de execução:

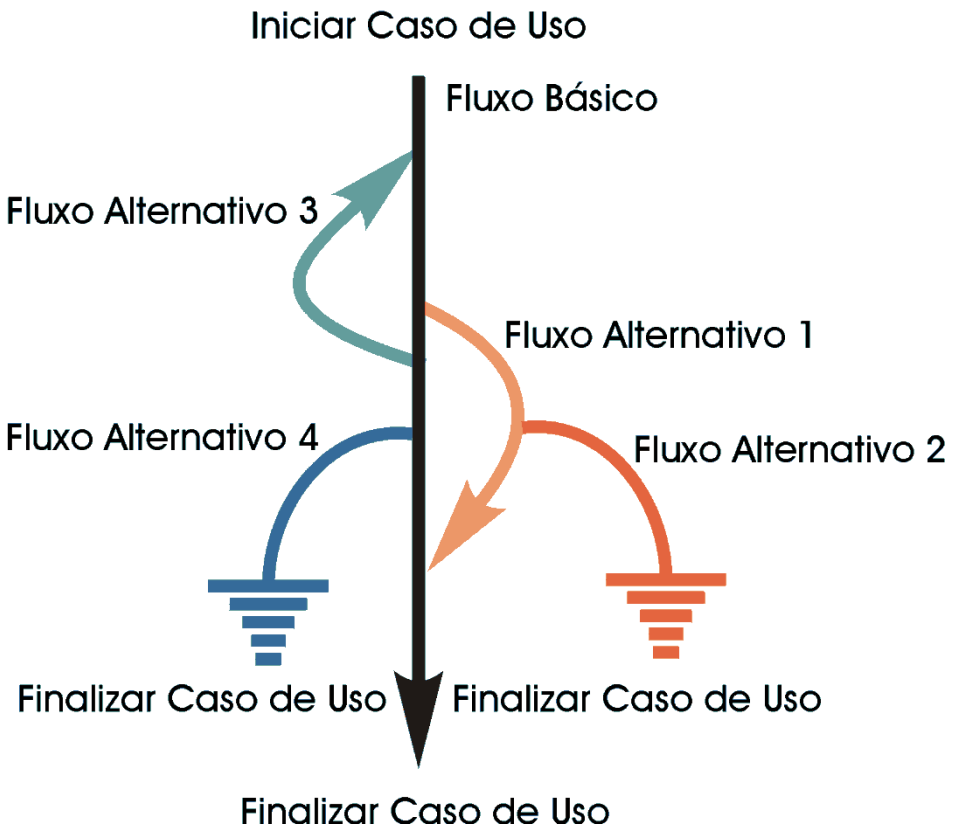
- Pré-Condições
  - É o estado obrigatório do sistema antes do início do teste
- Entradas
  - Estímulos específicos aplicados durante o teste
- Pontos de observação
  - Observações específicas a serem feitas

Estrutura do Caso de Teste

- Pontos de Controle
  - Identifica os pontos em que pode ocorrer mudança do fluxo de controle
- Resultados esperados
  - Condições observáveis esperadas após a execução do teste. Pode incluir resposta positivas e negativas (erros, falhas, etc.)
- Pós-condições
  - Estado ao qual o sistema deve retornar para permitir a execução dos testes subsequentes

Caso de Teste a partir de Caso de Uso

- É necessário desenvolver casos de teste para cada cenário do caso de uso
- Cenário (instâncias do caso de uso): caminhos que percorrem o fluxo básico, alternativo, de exceção, etc.



Caso de Teste a partir de Caso de Uso

- Após cada caminho possível do caso de uso mostrado, é possível identificar cenários do caso de uso através da combinação do fluxo básico com fluxos alternativos

Cenário 1	Fluxo Básico			
Cenário 2	Fluxo Básico	Fluxo Alternativo 1		
Cenário 3	Fluxo Básico	Fluxo Alternativo 1	Fluxo Alternativo 2	
Cenário 4	Fluxo Básico	Fluxo Alternativo 3		
Cenário 5	Fluxo Básico	Fluxo Alternativo 3	Fluxo Alternativo 1	
Cenário 6	Fluxo Básico	Fluxo Alternativo 3	Fluxo Alternativo 1	Fluxo Alternativo 2
Cenário 7	Fluxo Básico	Fluxo Alternativo 4		
Cenário 8	Fluxo Básico	Fluxo Alternativo 3	Fluxo Alternativo 4	

Caso de Teste a partir de Caso de Uso

- É possível obter casos de teste para cada cenário através da identificação da condição específica que causará a execução deste cenário

Exemplo (fluxo alternativo 3):

"Ocorrerá esse fluxo de eventos se o valor digitado no Passo 2 acima, "Digitar o Valor da Retirada" for maior que o saldo atual da conta. O sistema exibirá uma mensagem de aviso e, depois, retornará ao Passo 2 do fluxo básico "Digitar o Valor da Retirada" acima, onde o cliente do banco poderá digitar um novo valor de retirada"

Caso de Teste a partir de Caso de Uso

- Com estas informações, podemos especificar alguns casos de teste para o fluxo alternativo número 3

ID de Caso de Teste	Cenário	Condição	Resultado Esperado
TC x	Cenário 4	Passo 2 - Valor da Retirada > Saldo da Conta	Retorna ao Passo 2 do fluxo básico
TC y	Cenário 4	Passo 2 - Valor da Retirada < Saldo da Conta	Não executa o Fluxo Alternativo 3, segue o fluxo básico
TC z	Cenário 4	Passo 2 - Valor da Retirada = Saldo da Conta	Não executa o Fluxo Alternativo 3, segue o fluxo básico

Na prática, teríamos uma lista de TC's parecida com isso...

ID do TC	Cenário/Condição	Senha	No da Conta	Valor Digitado (ou escolhido)	Valor na Conta	Valor no Caixa Eletrônico	Resultado Esperado
CW1.	Cenário 1 - Retirada em Dinheiro Bem-sucedida	4987	809 - 498	50.00	500.00	2,000	Retirada em dinheiro bem-sucedida. Saldo da conta atualizado para 450,00
CW2.	Cenário 2 - Caixa Eletrônico sem Dinheiro	4987	809 - 498	100,00	500,00	0,00	Opção Retirada em Dinheiro indisponível, fim do caso de uso
CW3.	Cenário 3 - Fundos insuficientes no caixa eletrônico	4987	809 - 498	100,00	500,00	70,00	Mensagem de aviso, retorno ao Passo 6 do Fluxo Básico - Digitar o Valor
CW4.	Cenário 4 - Senha Incorreta (> 1 nova tentativa)	4978	809 - 498	n/a	500,00	2.000	Mensagem de aviso, retorno ao Passo 4 do Fluxo Básico, Digitar a Senha
CW5.	Cenário 4 - Senha Incorreta (= 1 nova tentativa)	4978	809 - 498	n/a	500,00	2.000	Mensagem de aviso, retorno ao Passo 4 do Fluxo Básico, Digitar a Senha
CW6.	Cenário 4 - Senha Incorreta (= sem novas tentativas)	4978	809 - 498	n/a	500,00	2.000	Mensagem de aviso, cartão retido, fim do caso de uso

Exercícios [11]

SAD/PE (CESPE 2010)

[56] A respeito do plano de teste, um registro do processo de planejamento de testes de software, assinale a opção correta.

A) O processo de planejamento de testes é usualmente descrito em um plano de testes.

B) Um plano de teste de software é um registro da execução de um caso de teste de software.

C) A automação de um teste de integração é mais facilmente empreendida que a de um teste de módulo.

D) A produção de scripts de teste deve preceder a eventual construção de casos de teste.

E) Ao se inspecionar o conteúdo de um plano de testes, devem-se encontrar, entre outras, as seguintes descrições: escopo de testes, abordagens de teste, recursos para realização dos testes e cronograma das atividades de teste a serem realizadas.

Gabarito dos Exercícios

[1] - [109] C, [61] E
[2-A] - [13-B] E, [85] E, [75] C, [56] C
[2-B] - [98] E, [97] E
[3] - [73] E, [63-A] E, [63-B] E, [97] C
[4] - [79] C, [57-C] E, [57-D] E, [74] C, [13-C] E
[5] - [85] E, [57-B] E, [57-E] E, [57-A] C, [13-D] E
[6] - [13-A] E, [13-B] E, [83] C, [40-A] E, [40-C] E
[7] - [58] C, [83] E
[8] - [39-D] E, [55-C] E, [43] D
[9] - [101] E, [48] E
[10] - [43] B
[11] - [56] E

FIM