# Introdução

Em geral o tempo e custo total de desenvolvimento de software gastos com teste são de 50%.

A maior parte dos defeitos é de origem humana.

Continuam presentes nos diversos produtos de software produzidos e liberados (10 defeitos a cada 1000 linhas de código).

A maioria encontra-se em partes do código raramente executadas.

**Garantia da Qualidade de Software**: é um conjunto de atividades técnicas aplicadas durante o processo de desenvolvimento. O Objetivo é garantir que tanto o *processo* de desenvolvimento quanto o *produto* de software atinjam níveis de qualidade especificados.

**V&V**: Verificação e Validação

Verificação: assegurar consistência, completude e corretude do produto em casa fase e entre fases consecutivas do ciclo de vida do software.

*“Estamos construindo corretamente o produto?”.*

Validação: assegurar que o produto final corresponda aos requisitos do software.

*“Estamos construindo o produto certo?”*

Principal atividade: teste de software.

**Processo de Teste**

Planejamento: análise de risco, definição de recursos e ambientes necessários, critérios de parada, cronograma, resultados esperados, equipe etc.

Projeto de casos de teste: definição dos cenários, identificação e priorização de casos de teste, criação de casos de teste, identificar dados de teste específicos etc.

Configuração de dados e do ambiente de teste: desenvolver e priorizar procedimentos de teste, implementar o ambiente de teste, realizar pré-teste etc.

Execução e avaliação do teste: executar casos de teste, relatar incidentes de teste, escrever log de teste, decidir sobre incidentes de teste, acompanhar o status dos incidentes etc.

Monitoramento e controle: conduzir revisões do progresso do teste, monitorar defeitos, conduzir revisões de qualidade do produto, analisar problemas, tomar ações corretivas etc.

TMMi – Test Maturity Model integration: modelo de maturidade especializado na melhoria do processo de teste.

5 níveis de maturidade:

1. Inicial; (2) Gerenciado; (3) Definido; (4) Gerenciado Quantitativamente; (5) Em otimização.

Quando o teste ocorre? Analise 🡪 Projeto 🡪 Codificação 🡪 Código 🡪 Teste:

Atividade dinâmica, implica a execução do artefato em teste, avalia o comportamento.

*Obs: Não requer código, desde que os artefatos disponíveis tenham essas características...*

Uma das principais causas de uma atividade de teste pobre é a falsa impressão de que: “*Teste é o processo de demonstrar que defeitos não estão presentes.” / “O objetivo do teste de software é mostrar que o programa se comporta corretamente.” / “Teste é o processo de estabelecer uma confiança que o programa faz o que é esperado que ele faça.”*

**Teste é o processo de executar um programa com o objetivo de revelar defeitos.**

Qual o objetivo da atividade de teste? É uma atividade para mostrar que o programa está correto?

**Não. Refutar a afirmação que o programa está correto, isto é, mostrar que o programa está incorreto.**

Depois de realizado o teste é possível afirmar que o programa está correto?

**Não. É possível aumentar a confiança de que isso seja verdade, mas não garantir! A não ser que se aplique o teste exaustivo.**

Quando se aplica teste exaustivo = testar com TODAS as possíveis entradas.

Em geral, não é possível aplicar o teste exaustivo: Não é possível afirmar que o programa está correto.

Quando para a atividade de teste? Terminou o tempo, terminou o recurso financeiro, o plano de teste foi cumprido.

**O Plano de Teste:** documento que contém todas as informações sobre a atividade de teste pretendida –

Equipe de teste, software de suporte, cronograma das atividades, técnicas e critérios a serem seguidos, forma de execução dos casos de teste, forma de avaliação dos resultados, etc.

***Teste de Software: Conceitos Básicos***

Defeito: deficiência algorítmica que, se ativada, pode levar a uma falha;

Erro: estado inconsistente, decorrente da execução de um defeito;

Falha: evento observável que mostra que o programa violou suas especificações.

Ocorre quando um erro extrapola as barreiras do sistema.

Defeito 🡪 Erro 🡪 Falha

Tupla [d, O(d)] onde:

*d:* representa um elemento de um domínio D que servirá

*O(d)*: saída esperada do software quando executado com a entrada *d*

Informalmente, um caso de teste é formado por um conjunto de entradas e um conjunto de saídas esperadas após a execução do software com tais entradas.

Oráculo: é quem decide se a saída obtida é de fato a saída esperada. Pode ser automatizado, porém é difícil em diversas situações.

***Técnicas de Teste***: formas de se testar o software baseadas em diferentes artefatos de software disponíveis.

Relacionadas aos diferentes “pontos de vista” adotados para se testar um software:

Teste Funcional: especificação funcional (ou não funcional) do software;

Os requisitos de teste são extraídos da especificação do software.

Teste Estrutural: estrutura interna;

Os requisitos de teste são extraídos da estrutura do software, em variados níveis de granularidade.

Teste baseado em defeitos: defeitos mais recorrentes.

Os requisitos de teste são extraídos dos defeitos típicos e comuns inseridos durante a construção do software.

São complementares, pois identificam tipos diferentes de defeitos.

Maneira sistemática e planejada de elaborar os casos de teste (CT).

Podem ser usados de duas formas:

Para seleção dos casos de teste: quando os CTs são criados para satisfazer os *requisitos* do critério de teste.

Para adequação dos casos de teste: quando os CTs são criados e.g. aleatoriamente, e então verifica se eles atendem (ou satisfazem) os *requisitos* do critério de teste.

***O objetivo do teste é revelar, detectar e se eles não forem detectados o teste não pode afirmar sua ausência.***

***Testar tudo é impossível.***

***As técnicas de teste são complementares, isto é, devem ser aplicadas conjuntamente.***

***A execução do teste é criativa e difícil, pois para testar com eficiência é preciso conhecer o software a fundo.***

# Teste Funcional

Caixa Preta: objetivo do teste – cobertura da especificação

Modelos de teste obtidos das especificações: requisitos, arquitetura.

Detecção de defeitos depende da especificação;

Especificações ambíguas, incompletas, incorretas e inconsistentes são perigosas para o teste.

Especificações devem ser verificadas (Ex.: inspeção)

Limitações: não garante cobertura de código (nem mesmo de partes essenciais / críticas);

Por outro lado: qualquer fase, qualquer paradigma – não leva em conta detalhes de implementação.

**Particionamento de Equivalência:** domínio de entrada (e algumas vezes de saída) do programa/função é dividido em números finito de partições (ou classes) de equivalência.

Supõe-se que dados pertencentes a uma partição tenham a capacidade de revelar os mesmos defeitos;

Partições válidas e inválidas são consideradas.

Gerações de testes: selecionar um dado de cada partição;

Critério de cobertura: cada partição deve ser considerada por ao menos um CT.

# Análise do Valor Limite

*Myers* – experiência 🡪 casos de teste que exploram condições limites tem maior probabilidade de encontrar defeitos.

Valores imediatamente acima ou abaixo dos limitantes das classes de equivalência.

Análise do Valor Limite: usado em conjunto com Particionamento de Equivalência – não escolher dados de teste aleatoriamente – selecionar de forma a explorar limites.

Não existem diretrizes bem definidas, somente recomendações.

Recomendações:

1 – se a CE especifica um intervalo de valores – dados de teste para os limites do intervalo e dados de teste imediatamente subsequentes 🡪explorando classes inválidas vizinhas

Exemplo: classe válida estiver no intervalo -1.0 e +1.0.

Dados de teste: -1.0; +1.0; -1.001 e +1.001;

2 – se a CE especifica uma quantidade de valores, por exemplo, de 1 a 255 valores.

Dados de teste: , somente um valor; 255 valores e 256 valores.

3 – usar a diretriz (i) para as condições de saída;

4 – usar a diretriz (ii) para as condições de saída;

5 – se a entrada ou saída for um conjunto ordenado, maior atenção ao primeiro e último elemento;

6 – usar a intuição para definir outras condições limite.

# Teste Funcional Sistemático

Combina Particionamento e Análise de Valor Limite.

Requer ao menos dois CTs de cada partição, para minimizar problema de defeitos coincidentes.

Exemplo: programa que calcula o quadrado de um número;

Dado de teste: -2; resultado 4 é correto, mas pode ter vindo de uma soma 2+2.

Várias diretrizes...

**Valores Numéricos**

Para domínio de entrada, selecionar valores de entrada da seguinte forma:

1 – Valores discretos: testar todos os valores;

2 – Intervalo de valores: testar os extremos e um valor no interior do intervalo.

Para domínio de saída, selecionar valores de entrada que resultem nos valores de saída (tipo da entrada pode ser != tipo de saída; Ex.: entrada numérica, saída boleana); Escolher como entrada valores que explorem os valores de saída da seguinte forma:

1 – Valores discretos: gerar cada um deles;

2 – Intervalo de valores: gerar cada um dos extremos e ao menos um valor no interior do intervalo.

**Valores Especiais**

Devem ser explorados na entrada e na saída; Ex.: espaço em branco que pode ser interpretado como zero em um campo numérico.

Casos especiais (por ex., valor zero): sempre selecionados individualmente, mesmo que esteja dentro de um intervalo.

O mesmo para valores no limite da representação binária dos dados; Ex.: para campos inteiros de 16 bits, selecionar valores -32768 e +32767.

**Valores Inválidos**

Valores que correspondem a entradas inválidas – incluir nos CTs – assegurar que o software rejeita;

Tentar gerar valores inválidos, os quais não devem ser bem-sucedidos.

Importante – limites dos intervalos numéricos (tanto inferior como superior); valores imediatamente fora dos limites desses intervalos e também os valores imediatamente subsequentes aos limites do intervalo pertences ao mesmo.

**Números Reais**

Checar o limite para números reais pode não ser exato – ainda assim, incluir verificação como caso de teste;

Definir margem de erro de tal forma que, se ultrapassada o valor pode ser considerado distinto;

Além disso, selecionar números reais pequenos e também zero.

**Arranjos**

Quando se usa arranjo, tanto como entrada como saída – considerar o fato do tamanho do arranjo ser variável, bem como dos dados serem variáveis.

Elementos dos arranjos devem ser testados como se fossem variáveis comuns;

Além disso – tamanho do arranjo deve ser testado com valores intermediários, mínimo e máximo;

Para simplificar o teste – considerar linhas e colunas de um arranjo como se fossem subestruturas a serem testadas separadamente. Assim – testar arranjo em isolamento; como uma coleção de subestruturas, e estar cada subestrutura independentemente.

**Texto ou String**

Explorar comprimentos variáveis e também validar caracteres que compõem; Ex.: apenas alfabéticos; alfanuméricos; com caracteres especiais.

# Teste Estrutural

Caixa Branca: requisitos de teste com base em uma implementação.

Requer execução de partes ou componentes elementares do programa.

Caminhos lógicos são testados 🡪 verificam conjuntos de condições, laços e pares definição-uso de variáveis.

Limitações do teste de software que afetam o teste estrutural:

Não existe um procedimento de teste de propósito geral para provar a correção de um programa;

Dados dois programas, é indecidível se eles computam a mesma função;

É indecidível, em geral, se dois caminhos de um programa, ou de programas diferentes, computam a mesma função;

É indecidível, em geral, se um dado caminho é executável 🡪 se existe um conjunto de dados de entrada que leve à execução do caminho;

Caminhos ausentes: quando o programa não implementa algumas condições (ou funcionalidades) – caminho não existe;

Correção coincidente: o programa pode apresentar um resultado correto para um dado em particular de entrada, satisfazendo um requisito de teste e não revelando a presença de um defeito.

**Teste Estrutural**: conhecimento da estrutura interna do programa 🡪 aspectos de implementação fundamentais para a geração/seleção de CTs.

Em geral, abordagens utilizam Grafo de Fluxo de Controle (GFC) ou “grafo de programa”:

Programa P pode ser decomposto em conjunto de blocos de comandos;

Execução do primeiro comando de um bloco acarreta a execução de todos os outros comandos desse bloco;

Todos os comandos de um bloco, possivelmente com exceção do primeiro, têm um único predecessor e exatamente um único sucessor, exceto possivelmente o último comando.

"caminho" - sequência finita de nós (n1, ..., nk), k>= 2, tal que existe um arco de ni para ni+1 para i=1, ..., k

“caminho simples” – se todos os nós são distintos, exceto possivelmente o primeiro e o último.

“caminho livre de laço” – se todos são distintos;

“caminho completo” – n1 = s e nk = o

“caminho não executável” – caminho para o qual não existe um dado de entrada que leve à sua execução.

IN(X) e OUT(X) – número de arcos que entram e que saem do nó x, respectivamente.

Se IN(X) = 0, x é um nó de entrada, e se OUT(x) = 0, x é um nó de saída.

Fluxo de Dados: ocorrência de variáveis em programas – “definição”, “indefinição” ou “uso”;

Modelo de fluxo de dados de Maldonado – definição:

Quando um valor é armazenado em uma posição de memória, em geral, definição de variável está:

- no lado esquerdo de um comando de atribuição;

- em um comando de entrada;

- em chamadas de procedimentos como parâmetro de saída.

*Uso*: quando referência não define valor à variável

*c-uso:* uso em computação de valores

*p-uso*: uso em predicado – afeta diretamente o fluxo de controle do programa;

Obs.: *c-usos* são associados aos nós; *p-usos* às arestas;

*Indefinição*: quando, ou não se tem acesso ao seu valor, ou sua localização deixa de estar definida na mem