1. Teste de software

Definição: Teste consiste na verificação dinâmica do funcionamento de um programa em um conjunto finito de casos de teste, cuidadosamente selecionado dentro de um domínio infinito de entradas, contra seu funcionamento esperado. São algumas as causas verificadas:

- **Fault (falta):** Causa de um mal funcionamento.
- **Failure (falha):** Efeito observável não desejável.
- **Erro**: Qualquer diferença entre um valor obtido, independente da sua forma de obtenção, e o valor teoricamente correto.
- **Exceção:** Evento que causa a suspensão da operação normal de um programa.
- **Anomalia:** Qualquer coisa observada na operação ou documentação de um programa que não está de acordo com as expectativas, baseado em documentos ou em outros programas.
- **Verificação:** Estamos construindo certo o produto?
- **Validação:** Estamos construindo o produto certo?
- **Oráculo:** Entidade responsável pela determinação dos resultados esperados em um teste.

Prováveis Defeitos no processo de desenvolvimento de software

- A maior parte é de origem humana – humanos não são 100% perfeitos
- Podem começar a aparecer desde a especificação de requisitos:
    - Especificação errada – não corresponde a necessidade do cliente
    - Especificação errada – requisitos impossíveis de implementar
- São gerados na comunicação e na transformação de informações
- 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
    - Principal causa: tradução incorreta de informações
    - Quanto antes a presença do defeito for revelada, menor o custo de correção do efeito e maior a probabilidade de corrigi-lo corretamente

    - Solução: introduzir atividades de VV&T ao longo de todo o ciclo de desenvolvimento

Tipos de falhas

- Algoritmo: algoritmo não produz a saída esperada
    - Esquecer de inicializar variáveis
    - Teste de condições erradas – desvios de código errado
- Sintaxe: linguagens de programação usadas indevidamente
    - Esquecer de colocar ponto-e-vírgula no final da linha em Pascal
- Computação e precisão
    - Erros em fórmulas matemáticas
    - Utilização errada de precisão – casas decimais e etc.
- Documentação
    - Código e documentação inconsistentes
- Overload: sobrecarga
    - Estouro de estruturas de dados – como vetores e matrizes
- Falha de Limite e Capacidade
    - Capacidade de tratamento = 32 dispositivos; Mais de 32 = erro
- Coordenação e Timing
    - Sistemas Distribuídos e de Tempo Real
    - Sistemas bancárias – controle de transações
- Desempenho
    - Velocidade do sistema não está de acordo com os requisitos
- Recuperação
    - Falhas em decorrência de eventos externos
    - Eventos como falta de luz
    - Tolerância a falhas
- Hardware
    - Normalmente relacionadas a DRIVERS de dispositivos
    - Falhas de Padronização e procedimentos

    - Padronização de codificação – entendimento do código

Princípios de testes

- Não planeje o teste assumindo que o programa está correto
- Um bom caso de teste é aquele que tem alta probabilidade de encontrar erro ainda não descoberto
- Caso de teste bem-sucedido é aquele que detecta erro ainda não descoberto
- A probabilidade de existência de mais erros numa parte do programa é proporcional ao número de erros já descoberto nela
- Teste deve ser feito por outra pessoa que não o autor do programa
- Dados de teste devem ser definidos para dados inválidos e não esperados
- Determinar sempre os resultados esperados
- Verificar cuidadosamente os resultados de cada teste
- Nunca jogue fora casos de teste, a não ser que esteja jogando fora também seu programa

Testes – Classificação

- Análise Estática
    - Análise do código-fonte sem executá-lo
    - Inspeções – entender o programa para descobrir erros
    - Walkthroughs – execução simulada do programa
- Análise Dinâmica
- Teste Simbólico
    - Executar o programa com dados simbólicos
    - Predicados de caminhos – utilização de grafos
- Verificação Formal
    - Especificação formal do programa
    - Provador de teoremas
    - Difíceis
    - Caros
    - Demorado

Teste de Unidade: Também conhecida como teste unitário ou teste de módulo, é a fase em que se testam as menores unidades de software desenvolvidas (pequenas partes ou unidades do sistema).

Teste de integração: Tenta identificar erros associados as interfaces entre os módulos de software. Necessário para verificar erros ocorridos quando da junção dos módulos.

Teste de validação: Tenta verificar se as funções estão de acordo com a especificação. Demonstrar a conformidade com a especificação e verificar se a documentação está correta. Duas abordagens:

- Teste Alfa – realizado pelo usuário no ambiente do desenvolvedor
- Teste Beta – realizado pelo usuário em seu próprio ambiente

Teste de sistema: O objetivo é executar o sistema sob ponto de vista de seu usuário final, varrendo as funcionalidades em busca de falhas em relação aos objetivos originais.

Ferramentas de Teste: ferramentas automatizadas de teste que apoiam o processo de teste e reduzem as falhas introduzidas pela intervenção humana, trazendo aumento de produtividade e facilitam estudos comparativos entre critérios.

2. Estimativa de Custo de Software

2.1. Estimativa LOC (Estimativa de Linha de Código)

A estimativa LOC contempla três passos:

- **(1) Estimativa do número de linhas de código no projeto final:** Uma abordagem padrão é, para cada parte, estimar o tamanho mínimo possível (Tamanho Mín.), o melhor tamanho (Ideal) e o tamanho máximo de linhas possível (Tamanho Máx.). A estimativa do número de linhas do projeto final é dada pela soma das estimativas de cada parte, em que a estimativa de uma parte é dada por 1/6 da soma da estimativa mínima, 4 vezes a estimativa ideal e a estimativa máxima de cada parte do projeto. Exemplo:

Parte	Tamanho Min.	Ideal	Tamanho Max.
1	20	30	50
2	10	15	25
3	25	30	45
4	30	35	40
5	15	20	25
6	10	12	14
7	20	22	25

As estimativas para cada parte são calculadas a seguir:

P1...P7 = (Tamanho Mínimo + (4 * Ideal) + Tamanho Máximo) / 6

P1 = (20 + 4*30 + 50)/6 = 190/6 = 31,6

P2 = (10 + 4*15 + 25)/6 = 95/6 = 15,8

P3 = (25 + 4*30 + 45)/6 = 190/6 = 31,6

P4 = (30 + 4*35 + 40)/6 = 220/6 = 36,7

P5 = (15 + 4*20 + 25)/6 = 120/6 = 20

P6 = (10 + 4*12 + 14)/6 = 72/6 = 12

P7 = (20 + 4*22 + 25)/6 = 133/6 = 22,17

Total = 31,6 + 15,8 + 31,6 + 36,7 + 20 + 12 + 22,17 = 170,07 LOC.

- (2) Estimativa de custo baseado por LOC baseado em dados históricos: a abordagem básica de LOC é a fórmula que combina dados históricos do projeto. A fórmula básica tem três parâmetros: Custo = $\alpha * KLOC^{\beta} + \gamma$

  α (Alfa) = custo marginal por KLOC (1000 linhas de código), que representa o custo adicional para 1000 linhas de código;

  β (Beta) = expoente que reflete a não-linearidade do relacionamento. Valores de beta maiores do que 1 significam que o custo por KLOC aumenta à medida que o tamanho do projeto aumenta. Se o valor de beta é menor do que 1, isso reflete uma economia de escala.

  γ (Gama) = reflete o custo fixo de qualquer projeto. Estudos encontraram valores positivos e o valor zero para gama.

  O exemplo a seguir tem como objetivo calcular o esforço necessário para um projeto novo de 30 KLOC.

PROJETO	TAMANHO (KLOC)	ESFORÇO
1	50	120
2	80	192
3	40	96
4	10	24
5	20	48

  -> Coeficiente = Esforço / Tamanho (KLOC) => Coeficiente = 120/50 = 2,4

  -> α = coeficiente = 2,4

  -> $\beta = 1$ (os coeficientes dos projetos são iguais)

  -> $\gamma = 0$ (valor padrão)

  Então => Custo = $\alpha * KLOC^{\beta} + \gamma$ => Custo = $2,4 * 30^{1} + 0$ => Custo = 72 PM

- (3) Estimativa de custo por LOC baseado no tipo de projeto – Constructive Cost Model (COCOMO): Sua unidade de esforço é dada em programador/mês (PM). Boehm dividiu dados históricos do projeto em três tipos de projeto:
    - Programas aplicativos (Ex: processadores de texto) => $PM = 2,4 * (KDSI)^{1,05}$;
    - Programas utilitários (Ex: compiladores) => $PM = 3,0 * (KDSI)^{1,12}$;
    - Programas de sistemas (embarcados) => $PM = 3,6 * (KDSI)^{1,20}$;

Exemplo: Calcule o esforço do programador para projetos de 5 a 50 KDSI.

KDSI	Aplicativos $PM = 2,4 * (KDSI)^{1,05}$	Utilitários $PM = 3,0 * (KDSI)^{1,12}$	Sistemas $PM = 3,6 * (KDSI)^{1,20}$
5K	13,0	18,19	24,83
10K	26,9	39,5	57,1
15K	41,2	62,2	92,8
20K	55,8	86,0	131,1
25K	70,5	110,4	171,3
30K	85,3	135,3	213,2
35K	100,3	160,8	256,6
40K	115,4	186,8	301,1
45K	130,6	213,2	346,9
50K	145,9	239,9	393,6

2.2. Análise por ponto de função

A análise por Pontos de Função (APF) é a técnica para determinar-se o tamanho de uma aplicação bem como os recursos de mão-de-obra e o prazo necessário para o desenvolvimento dessa aplicação. Pontos por Função (Contagem Total):

Complexidade de entrada:

Itens de Dados	1 a 4	5 a 15	16 ou mais
Arquivos			
0 - 1	Simples	Simples	Média
2	Simples	Média	Complexa
3 ou mais	Média	Complexa	Complexa

Complexidade de saída:

Itens de Dados	1 a 5	6 a 19	20 ou mais
Arquivos			
0 - 1	Simples	Simples	Média
2	Simples	Média	Complexa
3 ou mais	Média	Complexa	Complexa

Complexidade de arquivos:

Itens de Dados	1 a 19	20 a 50	51 ou mais
Tipo registro			
1	Simples	Simples	Média
2 a 5	Simples	Média	Complexa
6 ou mais	Média	Complexa	Complexa

Complexidade de interface:

Itens de Dados	1 a 19	20 a 50	51 ou mais
Tipo registro			
1	Simples	Simples	Média
2 a 5	Simples	Média	Complexa
6 ou mais	Média	Complexa	Complexa

Complexidade de consulta:

Entrada	1 a 4 itens	5 a 15 itens	16 ou mais
0 ou 1 Tipo de Arquivo	Simples	Simples	Média
2 Tipos de Arquivos	Simples	Média	Complexa
3 ou mais Tipos de Arquivos	Média	Complexa	Complexa
Saída	**1 a 5 itens**	**6 a 19 itens**	**20 ou mais**
0 ou 1 Tipo de Arquivo	Simples	Simples	Média
2 ou 3 Tipos de Arquivos	Simples	Média	Complexa
4 ou mais Tipos de Arquivos	Média	Complexa	Complexa

Peso por nível de complexidade:

Função	Entradas	Saídas	Arquivos	Interfaces	Consultas
Complexidade					
Simples	3	4	7	5	3
Média	4	5	10	7	4
Complexa	6	7	15	10	6

Ajuste de complexidade: 0,65 e 1,35 (+ - 35%).

Características da aplicação:

1- Comunicação de Dados
2- Processamento Distribuído
3- Performance
4- Utilização de Máquina
5- Volume de Transações
6- Entradas de Dados OnLine
7- Eficiência do Usuário Final
8- Atualização OnLine
9- Complexidade do Processamento
10- Reusabilidade
11- Facilidade de Instalação
12- Facilidade de Operação
13- Múltiplos Locais
14- Facilidade de Alteração

Você deve atribuir um peso de 0 a 5 a cada uma das características acima, de acordo com o nível de dificuldade que oferece ao projeto, construção, implantação e manutenção da aplicação.

Nível de influência:

Nível de Influência	Descrição
0	Não existe ou nenhuma influência
1	Pouca influência
2	Influência Moderada
3	Influência Média
4	Influência Significativa
5	Grande influência

Fórmulas úteis:

PF (pontos de função) = PF bruto x (0,65 + 0,01 x Grau de influência total

Exemplo = 87 x (0,65 + 0,01 x 38)

PF's ajustados = 90

Pontos por Função (Contagem Total):

			Fator de Ponderação			Total	
Parâmetro de Medida	Contagem		Simples	Médio	Complexo		
Número de Entradas		x	3	4	6	=	
Número de Saídas		x	4	5	7	=	
Número de Consultas		x	3	4	6	=	
Número de Arquivos		x	7	10	15	=	
Número de Interfaces Externas		x	5	7	10	=	
Contagem Total							