O objetivo da Engenharia de Software é produzir software com alta qualidade e baixo custo

Qualidade de Software: Segundo Pressman a qualidade de software pode ser definida como adequação a:

- Requisitos funcionais e de desempenho explicitamente estabelecidos.
- Padrões de desenvolvimento documentados.
- Características implícitas que são esperadas de todo software desenvolvido profissionalmente.

Garantia de Qualidade de Software: atividade cujo objetivo é garantir que tanto o processo de desenvolvimento quanto o produto atinjam níveis de qualidade especificados.

Verificação: conjunto de atividades que garante que o produto está construído corretamente.

Validação: conjunto de atividades que garante que o produto correto está sendo construído.

2. Importância do Teste

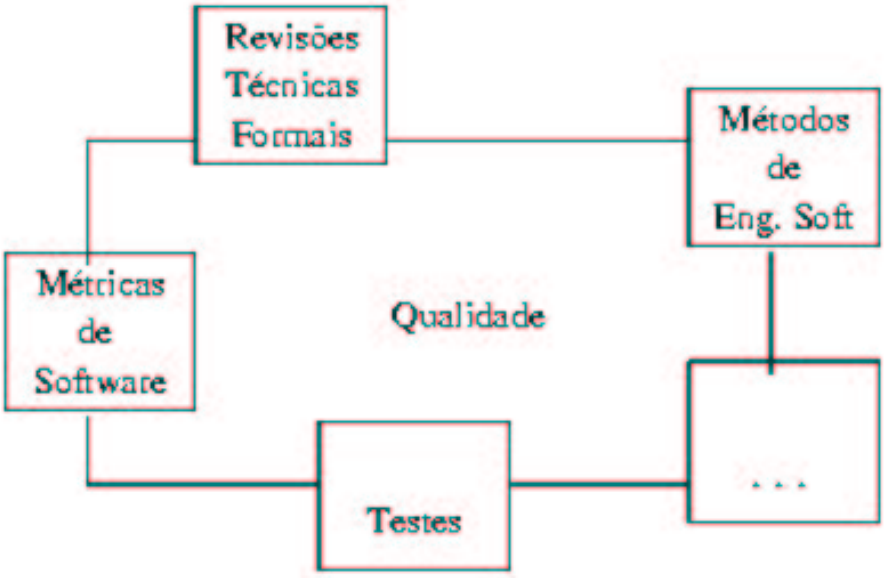
- Teste é uma atividade essencial para se garantir qualidade.
  É uma das últimas atividades que fará a revisão no produto.

- A atividade de desenvolvimento está sujeita a vários tipos de erros.
  Inabilidade para se garantir que as demais atividades foram realizadas corretamente.
  Falhas de comunicação e transformação nas diversas fases da E.S..

- Segundo Pressman as atividades de teste consomem cerca de 40% do tempo e custo do desenvolvimento.

Qualidade de Software :

Atividades de Verificação e Validação.



3. Teste: Definição e Objetivos

Segundo Myers teste é o processo de executar um programa com o objetivo de encontrar defeitos.

Um caso de teste é composto de um dado de entrada (dado de teste) e de uma saída esperada.

Um bom caso de teste é aquele que tem alta probabilidade de revelar um defito ainda não descoberto.

Um caso de teste é bem sucedido se revelou um defeito ainda não revelado.

Objetivos Secundários

- Demonstrar que um programa trabalha aparentemente como desejado.

- Fornecer indicações de confiabilidade e qualidade do software.

" Teste não nos pode mostrar a ausência de efeitos. O teste só pode nos mostrar que defeitos estão presentes ".

Erros, Falhas e Defeitos

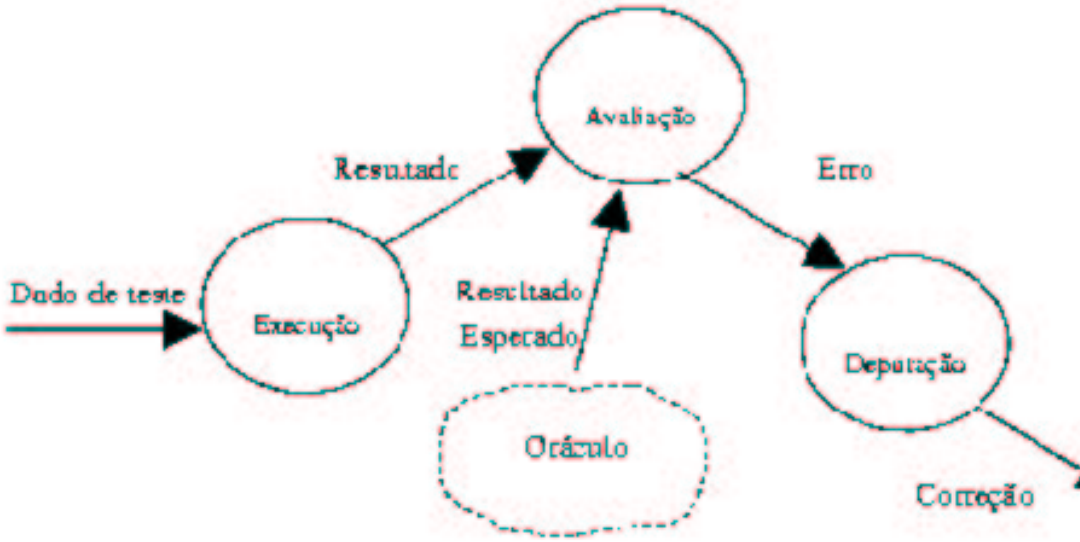
Engano ("mistake"): ação humana que produz um resultado incorreto – ex. uma ação incorreta tomada pelo programador.

Defeito ("fault"): um problema no programa – ex. uma instrução ou comando incorreto.

Falha ("failure"): produção de uma saída incorreta com relação a especificação.

Erro: diferença entre o valor correto e o valor esperado – qualquer estado intermediário incorreto ou resultado inesperado na execução do programa.

A atividade de teste segue o seguinte fluxo :



Caso de teste = Dado de teste : dado de teste + resultado esperado

TESTE diferente de DEPURAR.

IV - Técnicas de Projeto de Casos de Teste

Projeto de Casos de Teste - Exercício

Seja o seguinte programa:

```
programa:

- le um inteiro n

- escreve o inteiro n lido.
```

Idealmente: testar o programa para todas as entradas possíveis.

Impraticável devido à restrições de tempo e de custo.

Nas últimas décadas surgiram várias técnicas.

Objetivo: projetar bons casos de teste que possam revelar a maioria dos erros com um mínimo de tempo e esforço.

Técnicas de Teste

Técnica Baseada em Erros: utiliza certos tipos de erros para derivar requisitos de teste. Os casos de teste gerados são específicos para mostrar a presença ou ausência desses erros (ex. Análise de Mutantes)

Técnica Funcional: estabelece casos de teste baseados na especificação e na identificação dos requisitos funcionais (ex. Análise do Valor Limite, Grafos de Causa-Efeito, Teste de Partição)

Técnica Estrutural: utiliza o código fonte e a implementação para estabelecer os casos de teste (ex. Teste Baseado em Fluxo de Controle, Teste Baseado em Fluxo de Dados).

Critérios de Teste

Duas questões básicas:

1. Como selecionar os casos de teste?

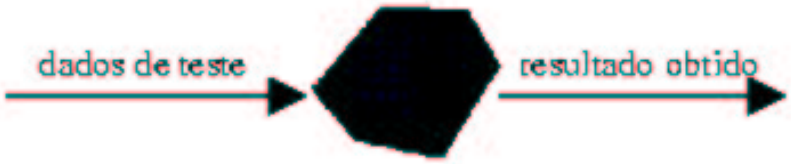
2. Como saber se o programa foi testado o suficiente?

Critérios de Seleção de Casos de Teste: estabelecem propriedades que dever ser satisfeitas para elaboração dos casos de teste.

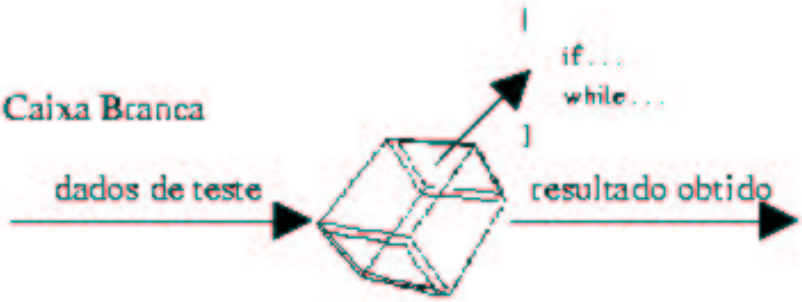
Critérios de Adequação de Casos de Teste: predicados a serem satisfeitos para avaliar um dado conjunto de casos de teste e para se considerar a atividade de teste encerrada.

Técnicas de projeto

2.1. Teste de Caixa Preta :



2.2. Teste de Caixa Branca



2.3. Teste Baseado em Erros

1. Técnica Funcional

Técnica Baseada na Especificação - Teste de Caixa Preta

- Utiliza a especificação para derivar os casos de teste

- detalhes de implementação são desconhecidos

- é mais aplicada durante as fases finas do teste.

1.1 Particionamento em Classes de Equivalência

- Classifica erros em uma mesma classe
- Particiona o domínio de entrada em classes de equivalência
- Considera que um dado de teste em uma mesma classe pode revelar o mesmo tipo de erro

As entradas de um programa podem ser: valores númericos, faixas de valores, um conjunto de valores relacionados, condições booleanas.

Regras para definir classes de equivalência

- Faixa de valores: classes: uma para um valor válido, duas para valores válidos.
  ex: $0 \leq n \leq 100$ — *n =50, n = 110; n = -40.*

- Valor específico: valores válidos e inválidos
  ex: numero de conta que deve ter 6 dígitos — um número com 6, um com 3, um com 8.

- Conjuntos: valores válidos e inválidos, combinado com valores válidos e inválidos de outros elementos do conjunto.
  ex: $n > 0, i > 5$ — *n = 40 i = 0; n = 40 i = 10; n = -10 i = 0; n -10 i = 10.*

- Booleanos: um válido e outro inválido.
  ex password.

1.2 Análise do Valor Limite

- Baseia-se na experiência de que um grande número de erros tende a ocorrer nos limites do domínio de entrada.

- É uma técnica mais completa que o particionamento, dados para executar os programas são tomados nos limites das classes de equivalência.

Regras

- Faixa de valores dadas por a e b, testar a, b, o valor imediatamente superior e inferiror a a e b.

  ex: $0 \leq n \leq 100$ — *n =0, n = -1;*

  n = 1, n = -100, n = 99, n = 101.

- Valor específico: testar máximo e mínimo

  ex: numero de conta que deve ter 6 dígitos — um número com 6, um com 5, um com 7.

- Se existir limites nas estruturas de dados:

  ex: se s é array[100], testar s[100], s[101], s[99].

2. Técnica Estrutural

Técnica baseada em programa, teste de caixa branca)

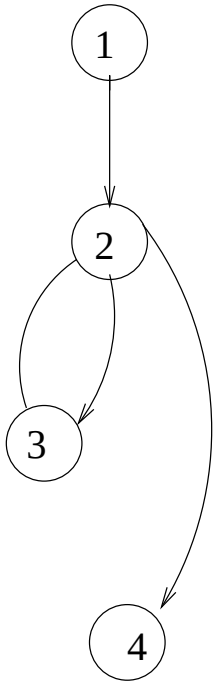
- Deve-se ter conhecimento sobre a estrutura interna do software e sobre o fluxo de controle do programa.

- É aplicada durante as fases iniciais do teste.

- Critérios Baseados em Fluxo de Controle
- Critérios Baseados em Complexidade
- Critérios Baseados em Fluxo de Dados.

Critérios de Teste Baseados em Fluxo de Controle

```
      void soma()
      {
          float soma;
          int i,n;
1         soma = 1;
1         scanf ("%d",&n);
1,2,3     for (i=2;i<=n;i++)
3           soma = soma + pow(i,2);
4         print ("%f", soma);
      }
```



Um caminho é uma sequência finita de nós (n1, n2, ..., nk), tal que existe um arco de ni para ni+1 em G.

ex: 1 2 4 é um caminho do grafo.

2.1 Critérios de Teste Baseados em Fluxo de Controle

Critério Todos-Ramos: exige que todos os ramos do grafo sejam executados pelo menos uma vez.

Critério Todos-Nós: exige que todos os nós do grafo sejam executados pelo menos uma vez.

Critérios Todos-Caminhos: exige que todos os caminhos sejam executados pelo menos uma vez.

Selecionar só os caminhos mais interessantes.

2.2 Critérios Baseados em Complexidade

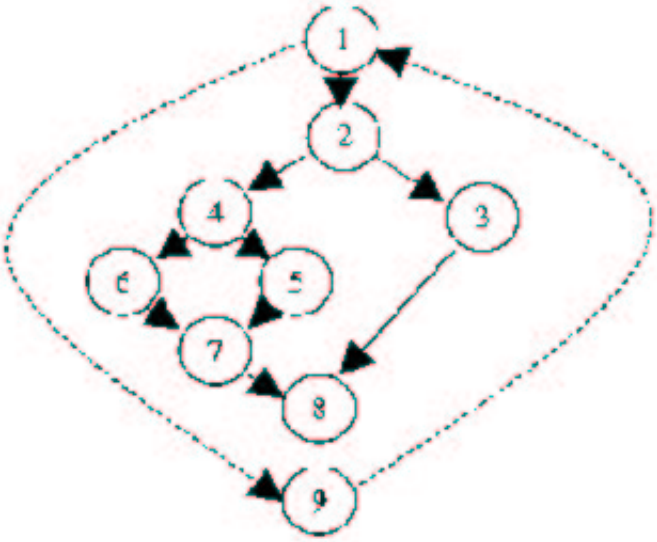
Critério Todos-Caminhos Linearmente Independentes de McCabe

- o número de caminhos li é dado por V(G).
- inclui o critério todos-ramos

Método dos Caminhos Básicos

– Testar apenas caminhos independentes (Caminhos que não são combinações de outros).

– O número de caminhos independentes é igual à complexidade ciclomática V(G).



Exercício :