

Aula 6: Estratégias do processo de Teste de Software

Apresentação

Sem estratégia, até a nossa vida pode não ser como esperávamos. Assim também acontece com os softwares.

Por isso, aqui vamos conhecer as estratégias de teste e de desenvolvimento de software. Sabemos como definir as estratégias entre as atividades de desenvolvimento e teste.

Como um dos pontos importantes, vamos identificar onde e quando utilizar, diferenciar e reconhecer testes caixa branca e caixa preta, além das abordagens de teste.

Objetivos

- Definir estratégias de teste de software;
- Demonstrar as estratégias de teste;
- Contrastar caixa branca de caixa preta.

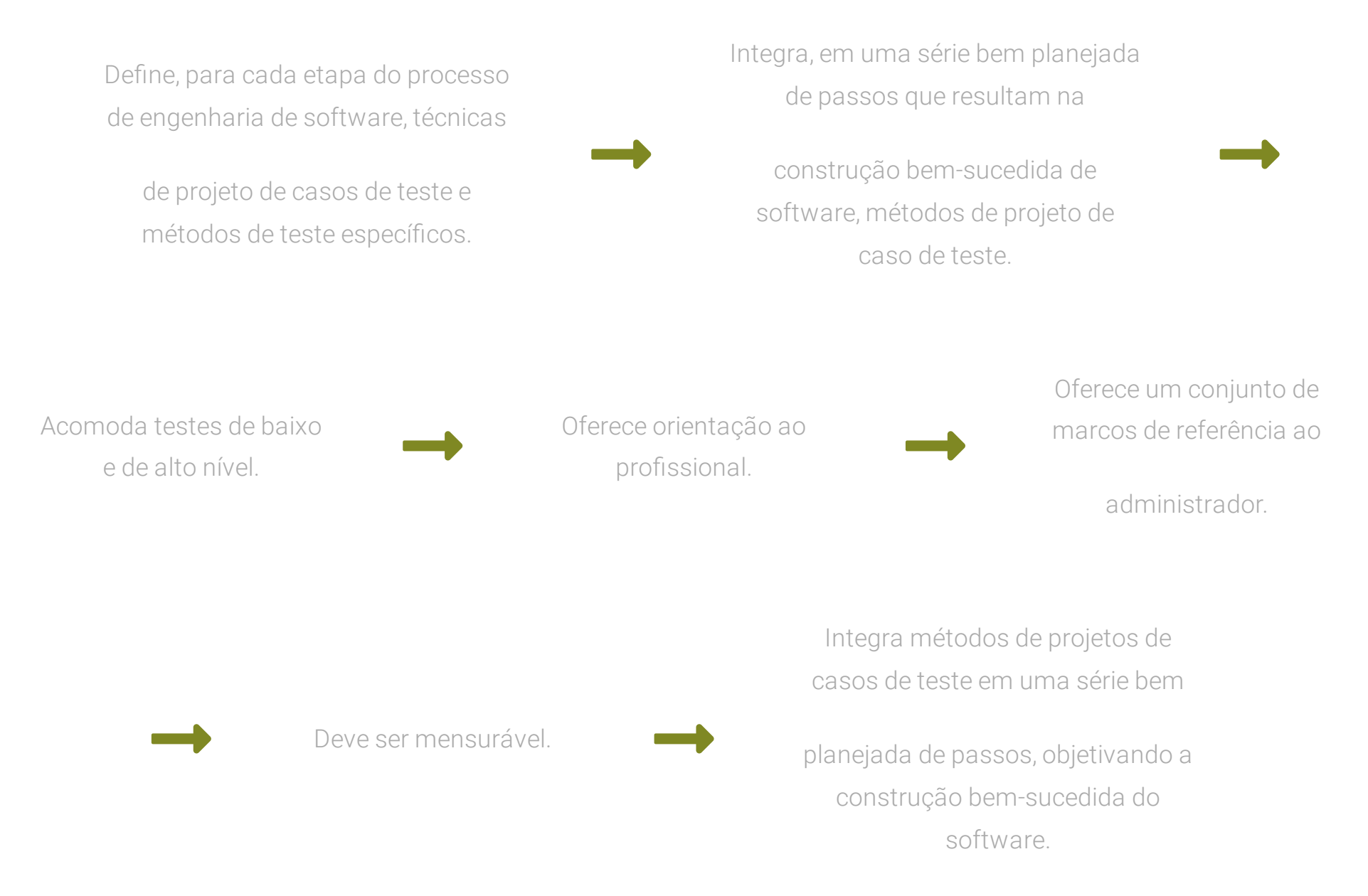


(Fonte: Photon photo / Shutterstock).

Estratégias de teste de software

O **planejamento** e a **realização** das atividades de teste definem a estratégia de testes de software.

Uma estratégia de teste de software:



Atenção

Em um primeiro momento, você pode não entender, mas o software é testado para promover erros que foram feitos inadvertidamente no momento em que foi construído.

Leia o texto para conhecer quem faz a estratégia e a atividade de teste.

Estratégia e atividade de teste

Quem faz a estratégia e a atividade de teste?

Estratégia de teste de software: desenvolvida pelo gerente de projeto, o engenheiro de software ou o analista de teste.

Atividade de teste: a equipe de desenvolvimento do software e, para grandes projetos, um grupo de teste independente.

Atenção: Atividades de teste e de depuração são **atividades diferentes**, mas a **depuração** deve acontecer em qualquer estratégia de teste.

Por que é importante?

O teste responde (e é responsável) por mais esforço que qualquer outra atividade de engenharia de software. Caso seja feito ao acaso, teremos tempo e esforço desperdiçados.

Quais são os passos?

Vejamos por uma visão bem simplista: o teste começa no “varejo” e acaba no “atacado”. Mas o que seria isso?

O *varejo* são os testes unitários, e o *atacado* são os testes de integração, depuração na satisfação dos requisitos do cliente.

Características genéricas de estratégias de teste:

- **Teste efetivo** – A equipe deve conduzir revisões técnicas formais (erros serão eliminados antes do teste);
- **Início do teste** – O teste começa no nível de componente e segue “para fora”, em direção à integração de todo o sistema;
- **Atividade de teste** - Inicia-se no nível de módulos e prossegue para fora, na direção da integração de todo o sistema;
- **Diferentes técnicas de teste** são apropriadas a diferentes pontos de tempo;
- **Fornecem roteiro** - Descrevem os passos a serem conduzidos como parte do teste, como:
 - Quando os passos são planejados e executados?
 - Quanto de esforço, tempo e recursos são necessários?

O que deve incorporar uma estratégia?

- O **planejamento** de teste;
- O **projeto** de casos de teste;
- A **execução** e o teste;
- A **coleta** e a avaliação de dados (resultado).

Importante: A estratégia deve ser flexível, para promover um planejamento razoável e acompanhamento gerencial ao projeto.

Veja, a seguir, um exemplo de uma estratégia (workflow) para sistemas bancários multicanais.



Testador



Planilha Excel com
casos de teste do
canal website



Website



Testador



Planilha Excel com
casos de teste do
canal dispositivo
móvel



Dispositivos móveis



Serviço



Testador



Planilha Excel com
casos de teste do
canal caixa eletrônico



Caixa eletrônico



Base de dados



Testador



Planilha Excel com
casos de teste do URA



URA



Testador



Planilha Excel com
casos de teste do
canal central de
atendimento




Central de
atendimento



Vejamos o que acontece nessa estratégia de teste: como não foi realizada por outros canais, ficou evidente que não foram realizados testes multicanais, e nesse caso, todos os participantes (em separado) têm a percepção de que os testes realizados pelo canal foram satisfatórios.

Algumas estratégias de teste

 Clique nos botões para ver as informações.

[Teste de unidade](#)



É caracterizado por concentrar-se em cada unidade do software, de acordo com o que é implementado no código fonte.

Cada módulo é testado individualmente, garantindo que ele funcione adequadamente.

Utiliza as técnicas de teste de caixa branca.

Teste de integração



É caracterizado por concentrar-se no projeto e na construção da arquitetura de software.

Os módulos são montados ou integrados para formar um pacote de software.

Utiliza principalmente as técnicas de teste de caixa preta.

Teste de validação



Os requisitos estabelecidos como parte da análise de requisitos de software são validados em relação ao software que foi construído.

Ao término da atividade de teste de integração, o software está completamente montado como um pacote.

Os erros de interface foram descobertos e corrigidos, e uma série final de testes de software — os **testes de validação** — pode iniciar-se.

Teste de sistema



Caracteriza-se por testar, como um todo, o software e outros elementos do sistema.

Envolve uma série de diferentes testes, cujo propósito primordial é pôr completamente à prova o sistema baseado em computador.

Alinhamento estratégico

Como fazer para que a busca pela melhoria do desenvolvimento de software seja realmente efetiva?

Podemos **alinhar** a engenharia de software aos objetivos de negócio da empresa.

Atenção

As **decisões** e **rumos** tomados pelas áreas que utilizam a Engenharia de Software devem sempre estar alinhados à estratégia da empresa, fazendo com que a área de desenvolvimento se torne uma ferramenta para aumentar o valor do negócio da organização.

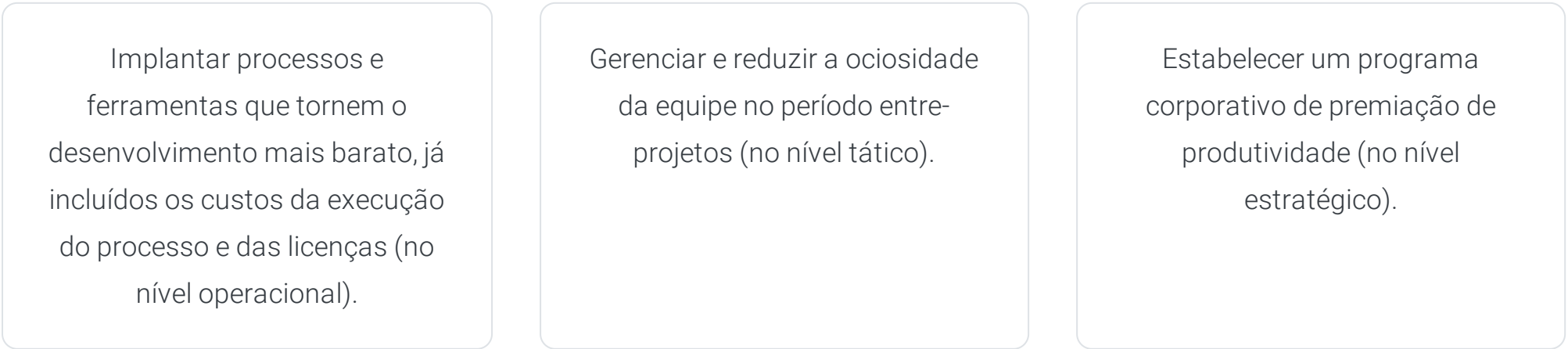
Nesse caso, é necessário que as empresas associem a Engenharia de Software às melhores técnicas a serem adotadas pelos profissionais relacionados com a produção de software. Será que todas as empresas têm essa visão ou apenas uma visão limitada?

Temos basicamente três níveis na empresa. A Engenharia de Software não deve apenas ficar no nível operacional, pois este cobre apenas parte das dificuldades, sendo que os problemas de maior impacto na organização aparecem nos níveis tático e estratégico.

Exemplo

- As empresas frequentemente apontam que sua área de desenvolvimento de software não conhece ou entende seu próprio negócio;
- Que utiliza indiscriminadamente tecnologias e ferramentas em seus produtos sem se preocupar se haverá ganho ou se o cliente está preparado para absorvê-las;
- Que se preocupa tanto com as próprias atividades que cria obstáculos, burocracia ou dificuldades para ser acionada por outras áreas;
- Empresas que, por exemplo, atuam desenvolvendo produtos de software customizados para outras organizações têm por estratégia minimizar seus custos.

Assim, as práticas adotadas devem favorecer essa estratégia, tais como:



Agora que entendemos que a Engenharia de Software deve atuar nos três níveis (Estratégico, Tático e Operacional), seus princípios de melhoria cíclica (processo de melhoria contínua) podem ser aplicados com sucesso.

Exemplo

Exemplo de um ciclo de melhoria:

- Primeiro é feito um diagnóstico sobre como o desenvolvimento de software na empresa precisa melhorar, em todos os níveis;
- Depois, os itens diagnosticados são priorizados com base no impacto no negócio, e os mais críticos são escolhidos;
- Ações para resolver os itens selecionados são então definidas e implementadas, observando o que precisa ser feito para que a ação gere resultados efetivos em todos os níveis;
- Finalmente, avalia-se o resultado da solução e o impacto positivo nos negócios.

As soluções de Engenharia de Software sempre contribuirão para agregar valor ao negócio com esses ciclos de melhoria adequadamente aplicados, e o software será efetivamente uma peça fundamental na estratégia da empresa.

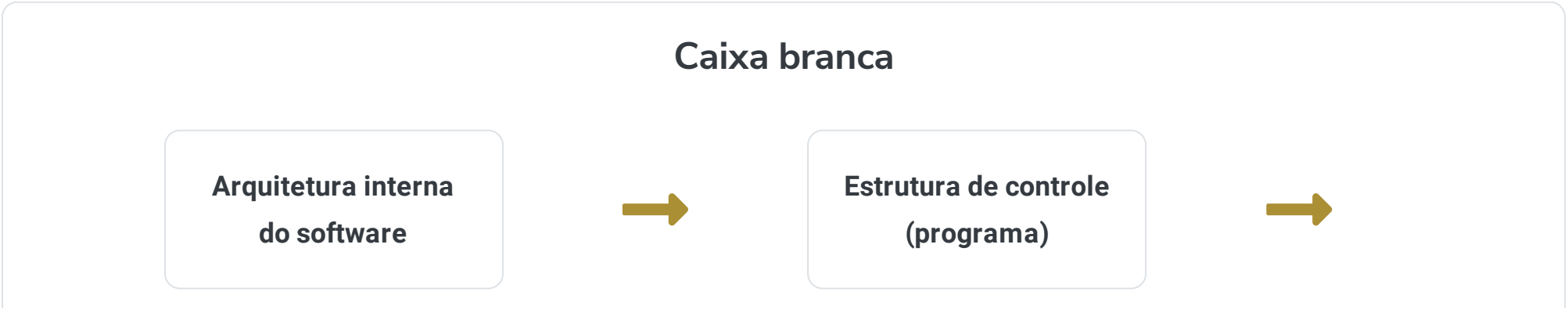
Teste caixa branca e teste caixa preta

Existem duas estratégias utilizadas para conduzir o **processo de validação** de software.



Estas estratégias não são excludentes; muito pelo contrário, elas são complementares.

- Conhecendo os detalhes internos de um produto, os testes são executados para assegurar que “**todas as engrenagens estejam articuladas**”, assim:
 - Operação interna de acordo com a especificação;
 - Todos os componentes internos foram adequadamente exercitados (teste **caixa branca** [aberta], white-box testing, teste estrutural).
- Conhecendo-se a função que um produto deve realizar, os testes são executados para demonstrar que cada função está completamente operacional (teste **caixa preta** [fechada], black-box testing, teste funcional).



Técnicas empregadas no teste caixa branca


Avaliação das estruturas de sequência, decisão e repetição, através da simulação de situações que exercitem todas as estruturas utilizadas na codificação adequadamente.

Vamos entender o que é um caso de teste, utilizado em teste caixa branca e preta.

É o documento que registra todo o planejamento dos testes e o que será testado. Deve identificar o maior número de cenários e variações possíveis, assim como os resultados esperados. Normalmente o desenho de um cenário de teste é mental.

Leia o texto para entender melhor sobre o **Teste caixa branca** e **Teste caixa preta**.

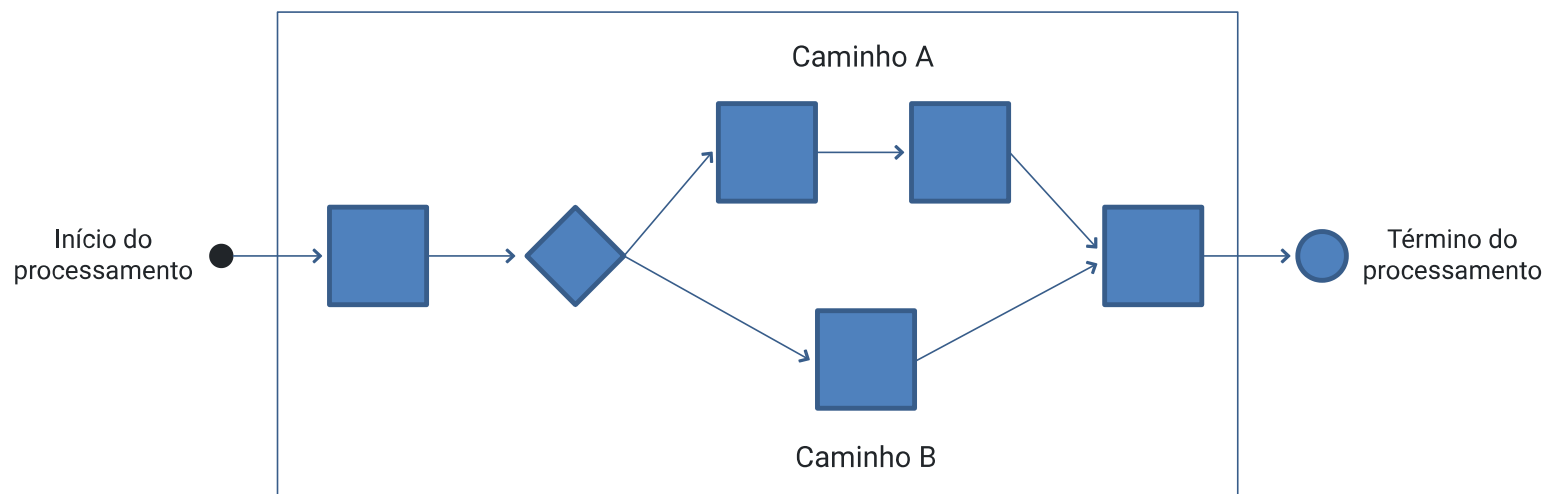
 Teste caixa branca e Teste caixa preta

 Clique no botão acima.

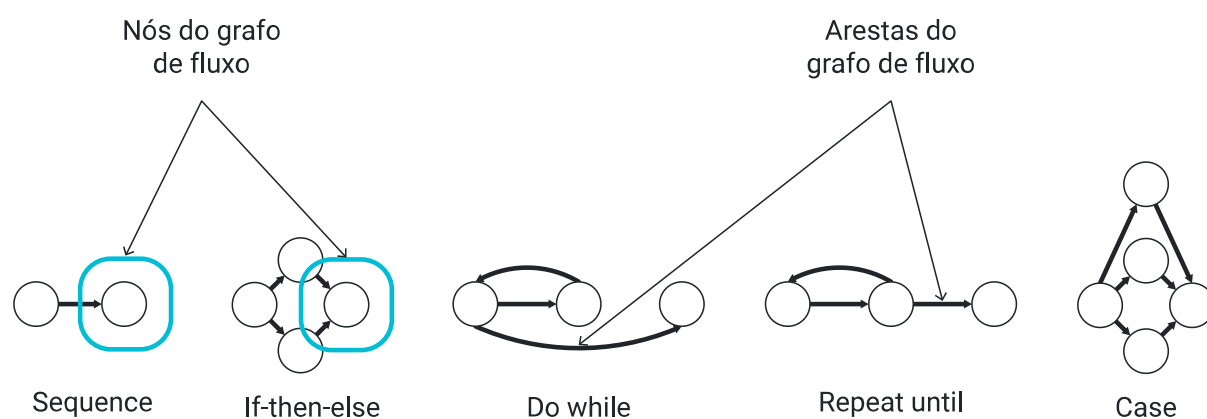
Teste caixa branca

O teste **caixa branca** é feito a partir das **estruturas de controle** do programa, através do maior número possível de cenários de testes que atendam ao maior número possível de situações.

Exemplo:



 Notação para grafo de fluxo (de programa) (Fonte: AUTOR / Shutterstock).



Nessa notação, cada **círculo** representa um ou mais comandos **nó de desvio** do programa fonte.

Exemplo:

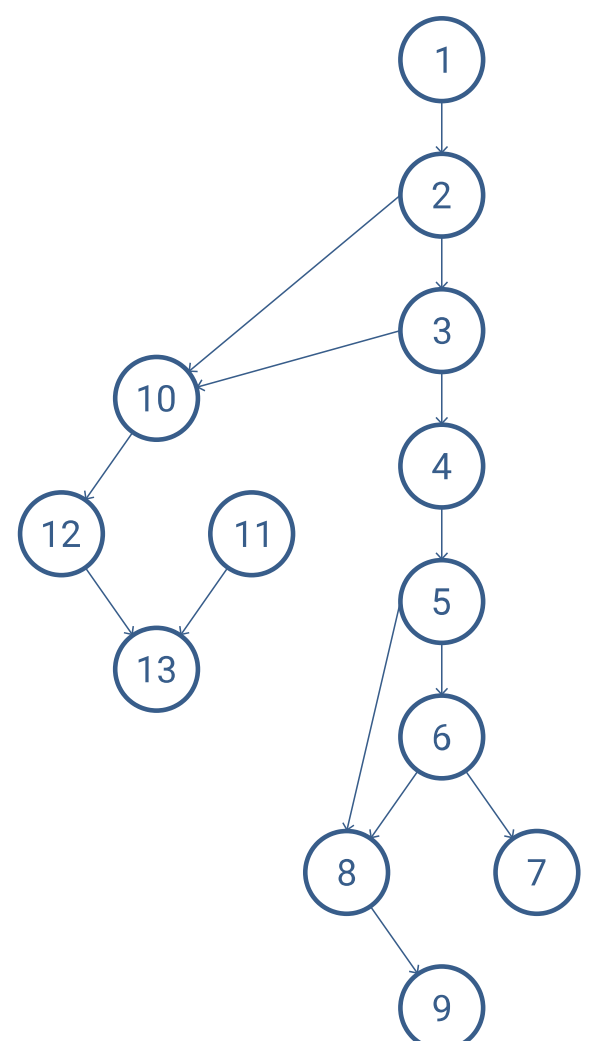
```

Procedimento média(valor[])

i=1;
soma=0;
total.entrada=0;
total.válidas=0

Faça-Enquanto (valor[i]_999 E total.entrada<100)
    incremente total.entrada de 1;
Se (valor[i]_min E valor[i]_max)
    Então
        incremente total.válidas de 1;
        soma = soma + valor[i]
Fim-Se
    incremente i de 1;
Fim-Enquanto
Se total.válidas>0
    Então média = soma/total.válidas;
Senão média = -999;
Fim-Se
Fim média

```



O teste **caixa branca** é considerado um teste procedural. Então, quais são os exames de detalhes desses testes procedurais?

- **Teste de caminhos lógicos** - Casos de teste para exercitar conjuntos específicos de condições e/ou laços;
- **O estado do programa** - Examinado em vários pontos para determinar se o estado esperado ou declarado corresponde ao estado real.

Importante: Não é possível testar todos os caminhos lógicos de um programa grande.

A abordagem deste tipo de teste tem as seguintes peculiaridades:

- Seleção de um **número limitado** de caminhos a serem exercitados;
- Estruturas de dados importantes podem ser investigadas para validação;
- Pode haver combinação dos dois tipos de teste, funcional e estrutural, para validar a interface do software e assegurar (seletivamente) que as operações internas do software estejam corretas.

A estrutura de controle do projeto procedimental deve ter casos de teste que:

- **Garantam** que todos os caminhos independentes dentro de um módulo tenham sido percorridos pelo menos uma vez;
- **Exercitem** todas as decisões lógicas em todos os seus ramos de saída;
- **Executem** todos os laços nos seus limites e dentro de suas fronteiras operacionais;
- **Exercitem** estruturas de dados internas para assegurar sua validade.

Modelo de caso de teste do RUP

Descrição do caso de teste	Uma descrição da condição, caminho do programa ou objetivo que este conjunto de dados implementa / executa.
Entradas de teste	Os objetos ou campos que os atores interagem e os valores dos dados de entrada pelo ator quando executa este caso de teste.
Resultados esperados	Estado resultante ou dado recebido após completa execução do caso de teste .

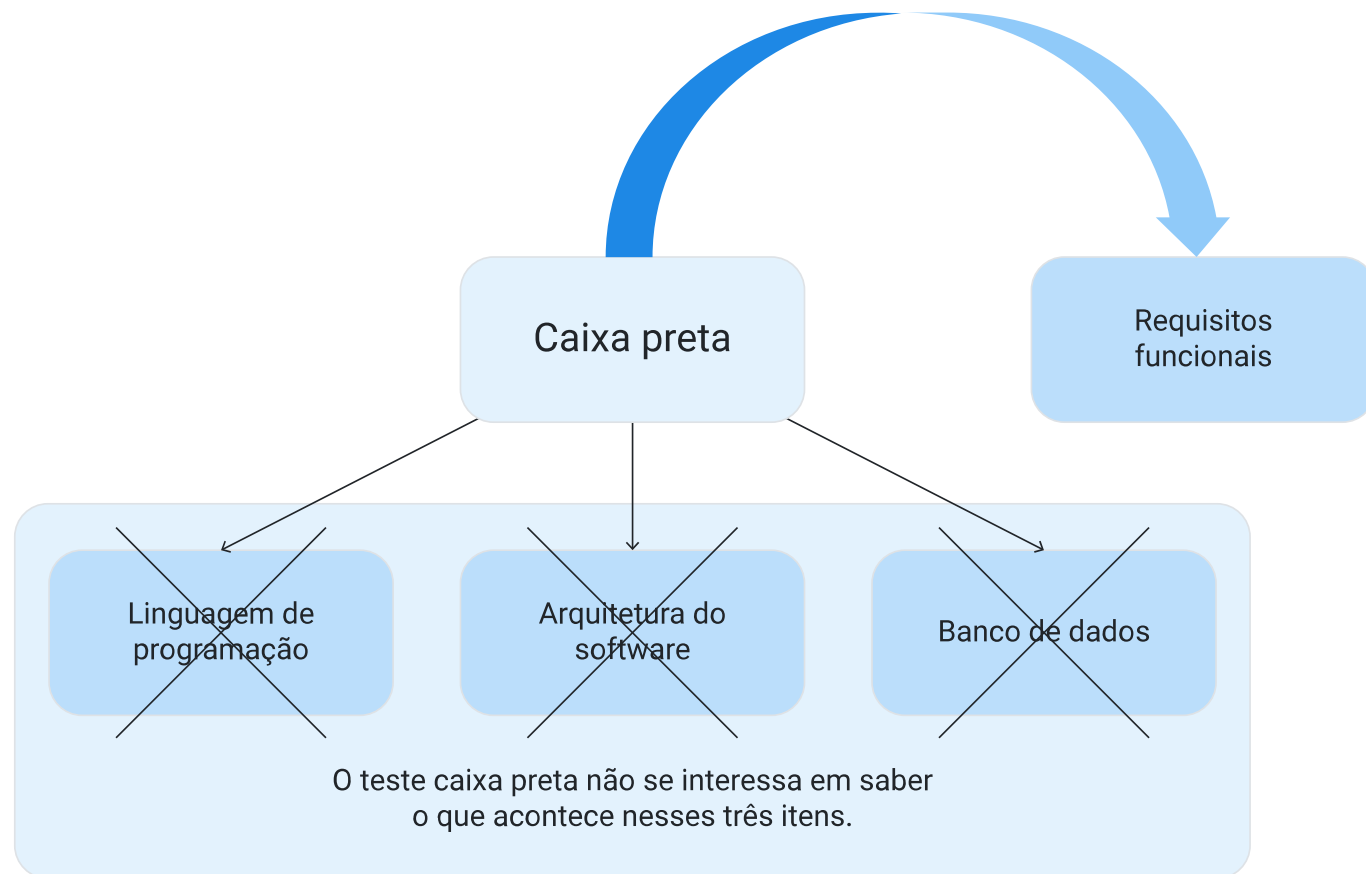
Exemplo de caso de teste

Id	Cenário / Condição	Dados de entrada	Resultados esperados
1112	Efetuar login - Operação com sucesso	Login - Válido Password - Válido	Usuário logado, página principal exibida

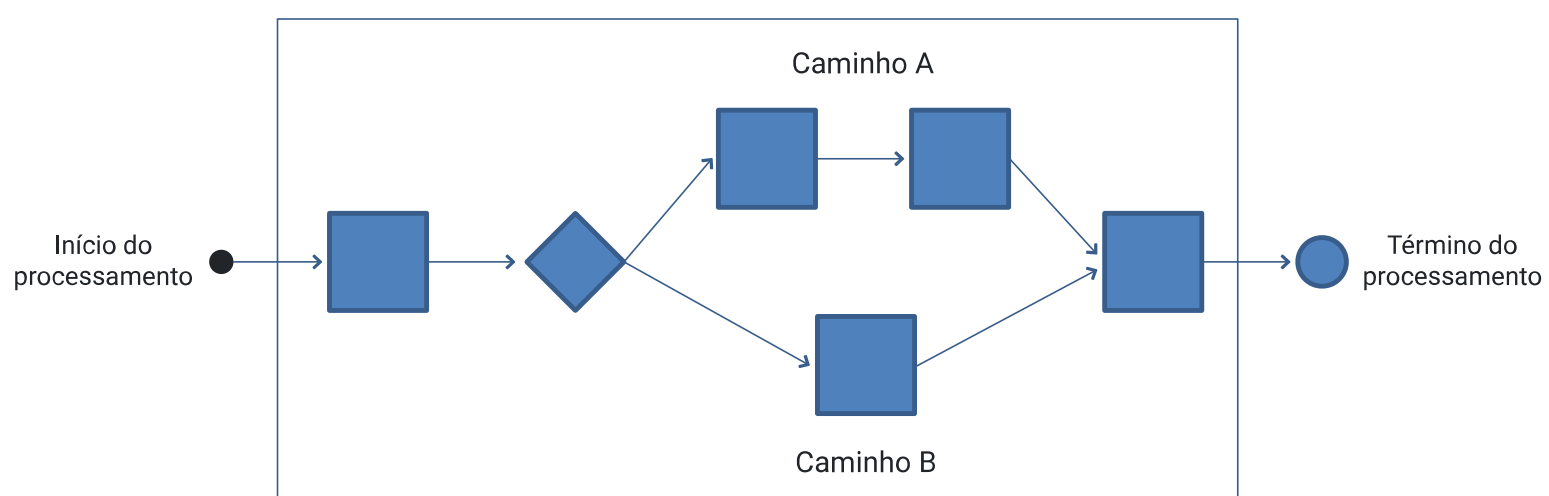
Teste caixa preta

Os **testes caixa preta**, também conhecidos como **testes comportamentais**, destacam os requisitos funcionais do software e utilizam técnicas para garantir que os requisitos do sistema sejam amplamente atendidos pelo software construído.

Esses testes não requerem o conhecimento da tecnologia empregada e dos conceitos de implementação do software, o que os diferencia dos testes caixa branca.



Exemplo



Para o teste caixa preta, devemos verificar:

- **Requisitos** (importante conhecer);
- **Características** (importante conhecer);
- **Comportamentos** (importante conhecer);
- **Maior variedade de cenários** (maiores simulações avaliadas).

O que é necessário que o profissional de testes conheça no teste caixa preta para que o software seja avaliado através dos resultados gerados pela aplicação? Os requisitos, suas características e comportamentos esperados.

Nesse caso, quanto maior a variedade de cenários, maior será o conjunto de simulações que serão avaliadas e comparadas com os requisitos da aplicação, implementados através da construção de casos de testes.

O que é importante o testador saber:

- Como a validade funcional é testada;
- Como o comportamento e o desempenho do sistema são testados;
- Que classes (**módulos** – procedimentos e dados [sistema]; **classes** – atributos e métodos [objeto]) de entrada farão bons casos de teste;
- Se o sistema é particularmente sensível a certos valores de entrada;
- Como as fronteiras de uma classe de dados podem ser isoladas;
- Que taxas e volumes de dados o sistema pode tolerar;
- Que efeito “combinações específicas de dados” terá sobre a operação do sistema?

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Abordagens de teste

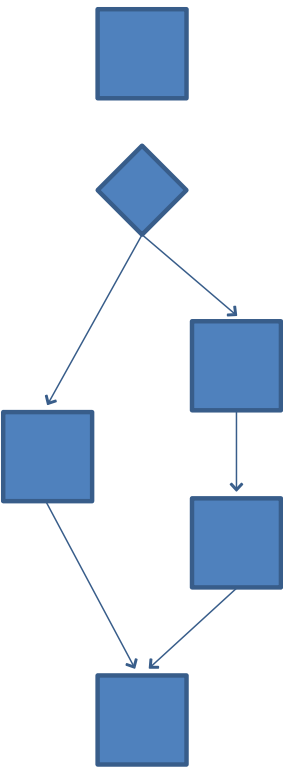
Segundo Bartié (2002), há várias formas de identificar e planejar os casos de testes a serem aplicados nos **testes de validação**; porém, o direcionamento dos testes (conhecido com **testdrive**) baseia-se exclusivamente em:

- Requisitos da solução tecnológica a ser desenvolvida (teste caixa preta); ou
- Estrutura interna do código-fonte a ser implementado (teste caixa branca).

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

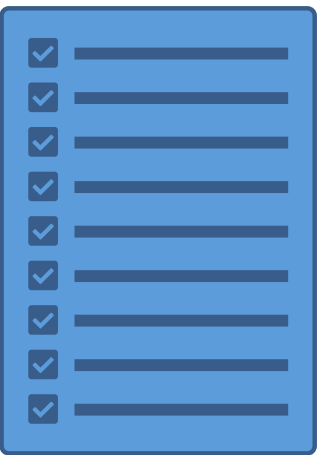
Caixa branca

Testes baseados na estrutura interna



Caixa preta

Testes baseados nos requisitos



Leia o a seguir para entender melhor sobre esses testes.

Abordagens de teste

Clique no botão acima.

Baseados na estrutura interna

Os testes requerem conhecimento profundo da tecnologia empregada e do projeto desenvolvido, de forma a praticarem adequadamente todas as estruturas internas do projeto.

Este tipo de teste requer que o testador tenha um amplo conhecimento interno do software e subdivide-se em:

**Teste de caminho
básico**

**Teste de fluxo de
dados**

Teste de ciclo

Teste de condição

Vejamos cada um deles.

Teste de caminho básico

Permite ao projetista do caso de teste apartar uma medida da complexidade lógica (medida usada como guia para definir um conjunto básico de caminhos de execução) de um projeto procedimental e usá-la para definir um conjunto básico de caminhos de execução.

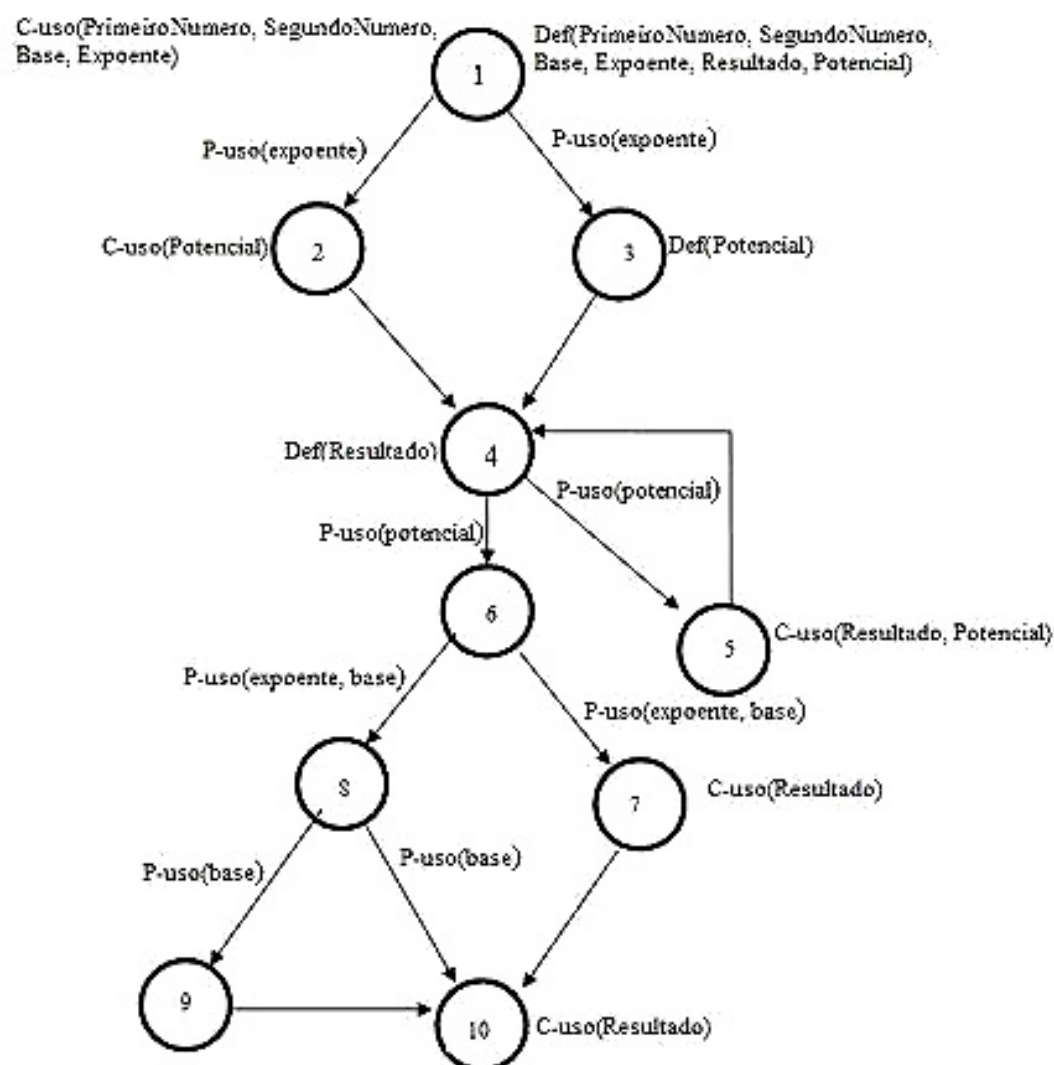
Os casos de teste apartados para exercitarem o conjunto básico têm a garantia de executar cada instrução do programa pelo menos uma vez durante a atividade de teste.

Teste de fluxo de dados

Tenta encontrar erros nas seguintes categorias:

- Seleciona caminhos de teste de um programa de acordo com as localizações de definições e usos de variáveis no programa;
- Seleciona caminhos de teste de um programa que contenha instruções de laços e *if* aninhados.

Vejamos um exemplo demonstrando como usar o teste de fluxo de dados em um grafo de fluxo:



Teste do ciclo

Tipos de abordagens:

- Simples;
- Aninhados;
- Concatenados;
- Não estruturados.

Simples

O conjunto de testes pode ser aplicado a ciclos simples, nos quais “n” é o número máximo de passadas permitidas através do ciclo.

Aninhados

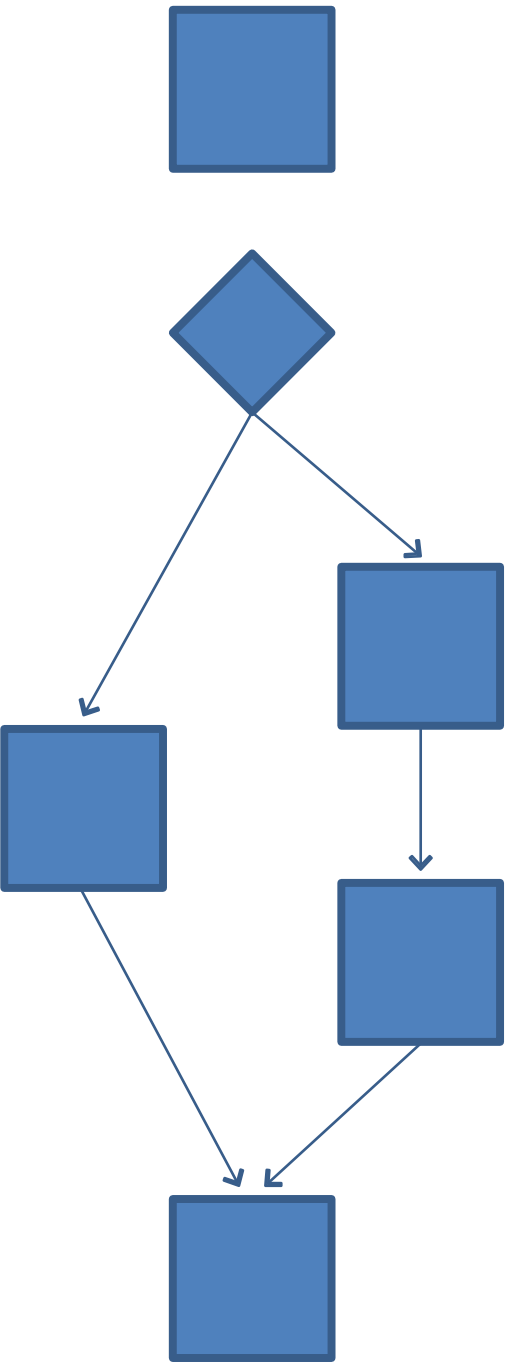
Testa se os ciclos mais internos que são aplicados à abordagem de ciclos simples e os outros externos permanecem com o valor mínimo.

Concatenados

- Os ciclos concatenados podem ser testados usando a abordagem definida para simples, se cada ciclo for independente do outro.
- Se dois ciclos forem concatenados e a contagem para o ciclo 1 for usado valor individual para o ciclo 2, então os ciclos não são considerados independentes.
- Quando os ciclos não são independentes, é recomendada a abordagem aplicada a ciclos aninhados. (Pressman, 2011)

Não estruturados

Segundo Pressman, sempre que possível, essa classe de ciclos deverá ser redesenhada para refletir o uso das construções de programação estruturada.



Baseados em requisitos

Os testes são baseados nos documentos de requisitos e modelados através de especificações funcionais e suplementares. Os requisitos devem ser decompostos em casos de testes de forma a avaliarem todos os cenários existentes e validarem todas as variações. (Bartié, 2002)

Baseado em grafo	Particionamento me equivalência	Análise do valor limite	Teste de matriz ortogonal
------------------	---------------------------------	-------------------------	---------------------------

Vejamos cada um deles a seguir.

Baseado em grafo

(A teoria dos grafos é um ramo da matemática que estuda as relações entre os objetos de um determinado conjunto. Para tal, são empregadas estruturas chamadas de grafos, G, onde V é um conjunto não vazio de objetos denominados vértices e A é um conjunto de pares não ordenados de V, chamado de aresta.)

Esse tipo de teste leva em consideração os objetos modelados no software e as relações que os unem. A ideia é definir uma série de testes que examinam se os objetos têm a relação esperada uns com outros.

Quando se tem uma coleção de nós que representam objetos, ligações que representam as relações entre objetos, pesos de nó que descrevem as propriedades de um nó e pesos de ligação que descrevem alguma característica de uma ligação, isso é um grafo.

Particionamento de equivalência

- Se uma condição de entrada especifica um intervalo, são definidas:
 - Uma classe de equivalência válida;
 - Duas classes de equivalência inválida.
- Se uma condição de entrada requer um valor específico, são definidas:
 - Uma classe de equivalência válida;
 - Duas classes de equivalência inválida.
- Se uma condição de entrada especifica um membro de um conjunto, são definidas:
 - Uma classe de equivalência válida;
 - Uma classe de equivalência inválida.
- Se uma condição de entrada for booleana, são definidas:
 - Uma classe válida;
 - Uma inválida.

Exemplo: Um programa valida um campo numérico no qual valores:

- Inferiores ou iguais a zero são rejeitados;
- Entre 1 e 150 são aceitos;
- Maiores ou iguais a 151 são rejeitados.

Neste caso:

- **Partição válida:** valores entre 1 e 150;
- **Partição inválida:** valores acima ou abaixo do valor válido.

Análise de valor-limite

Técnica que complementa o particionamento de equivalência, levando em consideração, na construção dos casos de teste, os valores limites das condições de entrada e saída.

Exemplo:

Um campo de entrada referente a data do nascimento e que aceita valores entre 1950 até 2018 terá como valores limites:

- Valor limite mínimo inválido: 1949;
- Valor limite mínimo válido: 1950;
- Valor limite máximo válido: 2018;
- Valor limite máximo inválido: 2019.

Matriz ortogonal

(Em álgebra linear, uma matriz ortogonal é uma matriz quadrada real M cuja inversa coincide com a sua transposta, isto é: note que uma matriz é ortogonal se e somente se as colunas são vetores ortonormais (vetores ortogonais e unitários).

Pode ser aplicada em problemas nos quais o domínio de entrada é relativamente pequeno, mas muito grande para acomodar um teste exaustivo.

Seu objetivo é a construção de casos de teste com uma visualização geométrica associada aos valores de entrada de uma aplicação.

Exemplo: Na função enviar para uma aplicação de web, são passados quatro parâmetros: P1, P2, P3 e P4, nos quais cada parâmetro assume três valores discretos.

P1 assume os seguintes valores:

- P1 = 1, enviar agora;
- P1 = 2, enviar após meia-hora;
- P1 = 3, enviar após 1 hora.

P2, P3 e P4 também assumem valores, 1, 2 e 3, significando outras funções de envio.

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Atividade

1. A equipe de desenvolvimento recebe o documento de resultados de testes gerado pelos homologadores. Responda:

- a. Qual processo os desenvolvedores devem executar agora?
- b. Como se desenvolve esse processo?
- c. Que documentos são utilizados como apoio a esse processo?

2. O gerente de um departamento de sistemas decidiu que os produtos de software criados pela equipe A serão homologados pela equipe B e vice-versa. Percebeu-se com o tempo, no entanto, o surgimento de diversos conflitos entre as equipes A e B. Responda:

- a. Qual a origem desses conflitos?
- b. Indique uma proposta de solução para minimizá-los.

3. A equipe Z realizou a codificação de uma nova tela para o sistema de controle de estoque. O objetivo da equipe é garantir que não existam erros, considerando apenas a parte “nova” do produto. Responda:

- a. Qual o tipo de teste que deve ser realizado?
- b. Quais diferentes visões devem ser consideradas ao aplicarmos este tipo de teste?

4. Ao testarmos a emissão de um relatório novo criado no sistema, para nossa surpresa, ao confirmarmos a impressão, o sistema emite a mensagem *invalid type conversion in line 453*, e o relatório não é emitido. Responda:

- a. Qual visão do teste de unidade não foi aplicada neste caso?
- b. Que cuidados devemos ter nesta visão dos testes de unidade?

Referências

BARTIÉ, Alexandre. **Garantia de qualidade de software**. 1. ed. Rio de Janeiro: Campus, 2002.

PRESSMAN, R. S. **Engenharia de Software**. 6. ed. São Paulo: Makron Books, 1995.

SOMMERVILLE, Ian. **Engenharia de Software**. 9. ed. São Paulo: Pearson, 2011.

Próxima aula

- Planos de teste e casos de teste com a ferramenta TestLink;
- Planejamento e execução dos testes;
- Importância da UML.

Explore mais

Leia o texto:

[Qualidade de Software: estratégias de testes](#)