Teste de software

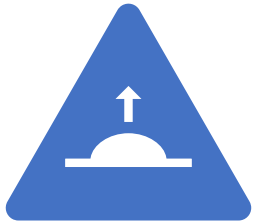
Elton Morais

Objetivos

- Distinguir teste de validação e teste de defeitos;
- Compreender os princípios de teste de sistemas e teste de componentes;
- Conhecer estratégias para casos de teste de sistema;
- Conhecer ferramentas de software que apoiam a automação de testes.

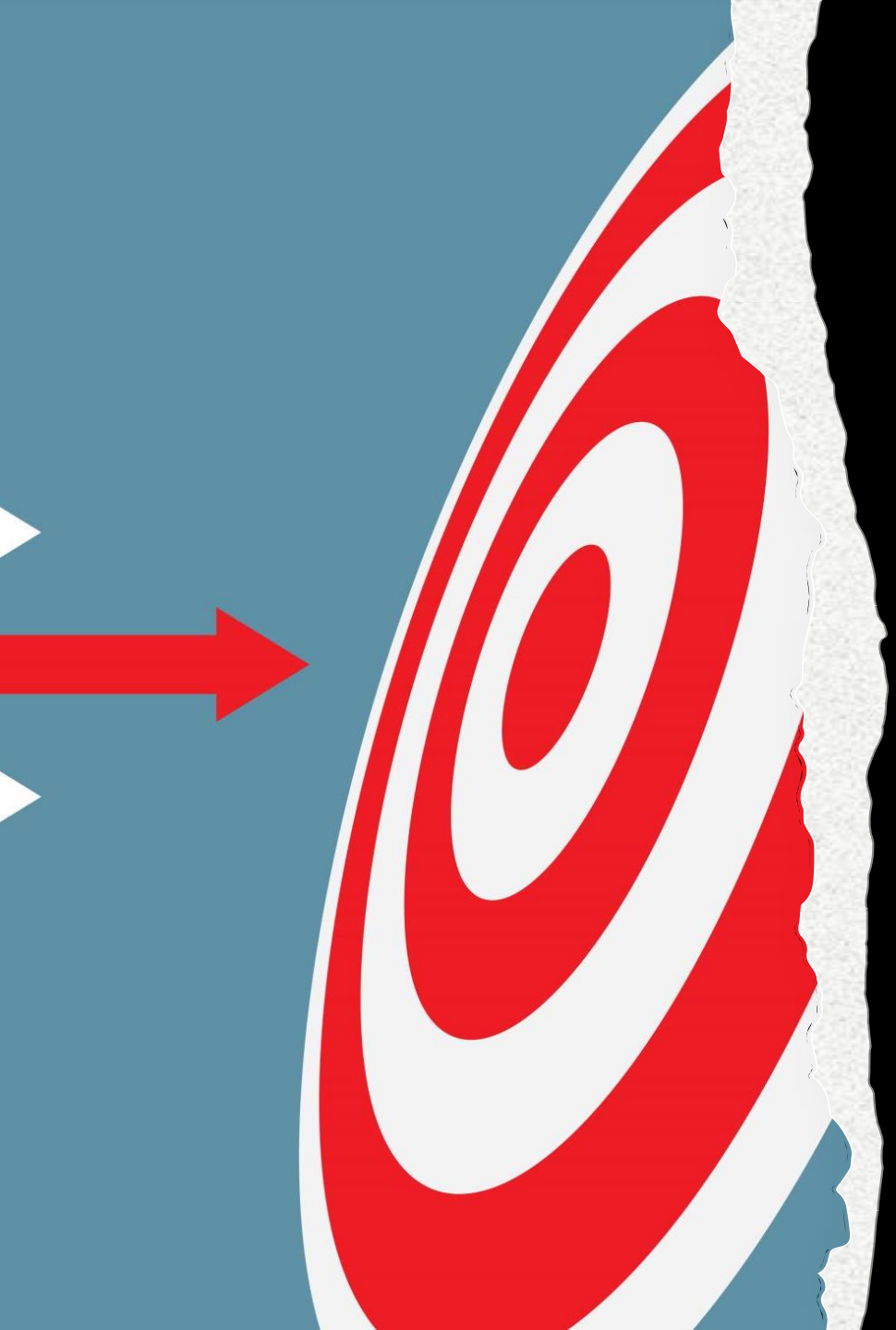
Introdução

A construção de software não é uma tarefa simples. Entretanto, diversos tipos de problemas podem resultar em um produto diferente daquele que se esperava.



Existem diversos fatores que podem causar tais problemas, porém, em sua maioria possui uma única origem: erro humano.

Para que esses erros não perdurem, uma série de atividades de **validação**, **verificação** e **teste** podem garantir que o produto esteja em conformidade com o especificado.



Metas de teste de software

Metas de teste de software

Demonstrar ao desenvolvedor e ao cliente que o software atende aos requisitos.

- Software sob encomenda: pelo menos um teste para cada requisito;
- Software genéricos: testes para todas as características de sistema;

Descobrir falhas ou defeitos no software que apresenta comportamento incorreto, não desejável ou em não conformidade com sua especificação.

Testes de sistemas

Teste de sistema

O teste de sistema verifica se as funcionalidades especificadas nos documentos de requisitos estão todas **corretamente implementadas**.

Concentra-se no teste de um incremento que será entregue ao cliente.

Em geral, para sistemas mais complexos, divide-se em:

- Testes de integração;
- Testes de releases;
- Testes de desempenho.

Testes de integração

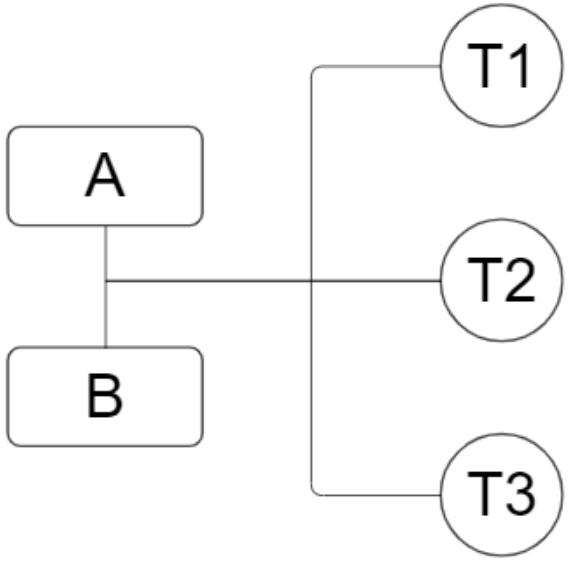
Nos testes de integração, a equipe de testes deve acessar o código-fonte do sistema. Uma vez descoberto um problema, a equipe tenta encontrar a origem do problema e identificar os componentes que devem ser depurados.

O teste de integração verifica se esses componentes funcionam realmente em conjunto, se são chamados corretamente e se transferem dados corretos no tempo correto por meio de suas interfaces.

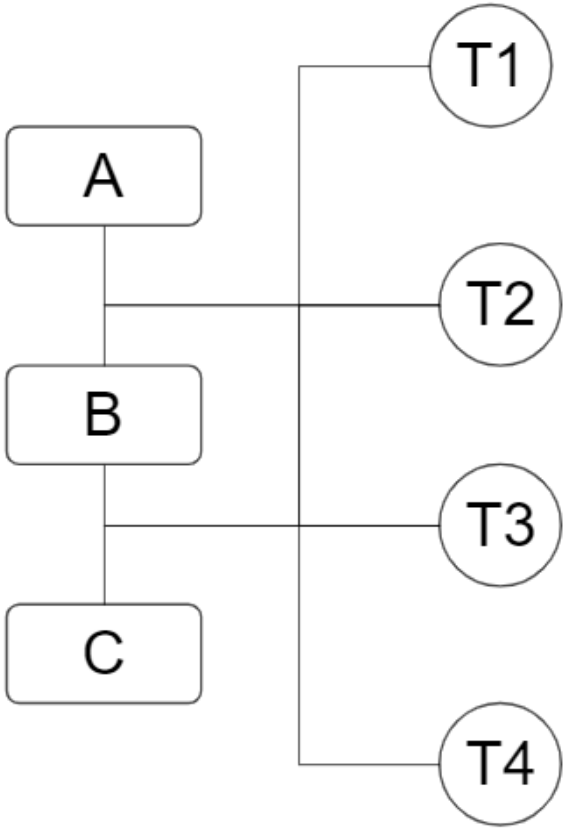
Testes de integração

Geralmente, um esqueleto geral do sistema é desenvolvido primeiro, e os componentes são adicionados a ele – ***integração top-down***. Ou, pode integrar componentes de infraestrutura que fornecem serviços comuns, como acesso à rede e ao banco de dados, e, em seguida, adicionar os componentes funcionais – ***integração bottow-up***.

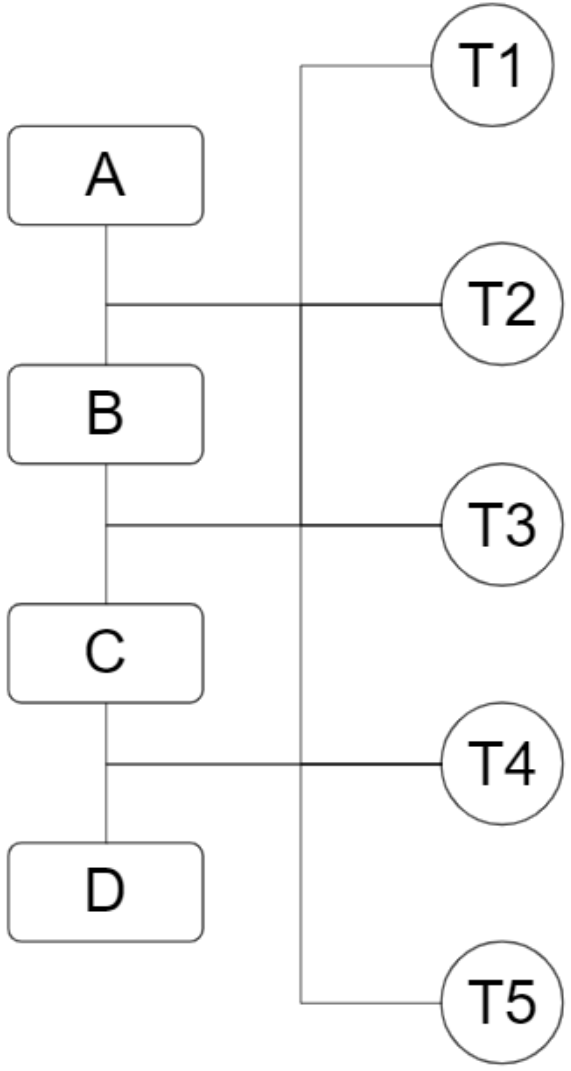
O problema principal é a localização de **erros**. Para tornar a localização de erros mais fácil, deve-se integrar uma configuração mínima do sistema e testá-lo.



Sequência de testes 1



Sequência de testes 2



Sequência de testes 3

Testes de integração

A integração e o teste de um novo componente podem mudar o padrão das interações de componentes já testados. Erros não expostos nos testes anteriores, podem ser revelados.

Nesses casos, novos testes devem ser executados no incremento anterior – ***teste de regressão***.

Caso o teste de regressão revelar problemas, deve-se verificar se é do incremento anterior ou do novo incremento adicionado.

Teste de releases

O teste de releases é o processo de teste do sistema que será distribuído aos clientes. O objetivo é demonstrar que o sistema atende aos **requisitos**, mostrando que as **funcionalidades**, o **desempenho** e a **confiabilidade** especificados não apresentam filhas durante o uso normal.

Os testes de releases geralmente são teste de **caixa-preta**.

Os testes do tipo caixa preta, são conduzidos “às cegas”, e tem como objetivo observar como um usuário executa uma ação ou tarefa dentro da aplicação. Fatores como estrutura interna, código, design e estratégias de desenvolvimento são desconhecidos pelo testador.

Teste de releases

O testador fornece as entradas para o componente ou o sistema e examina as saídas correspondentes. Se as saídas **não forem as previstas**, o teste detectou um problema com o software.

Para que os testes sejam bem-sucedidos:

- Escolha entradas que forcem o sistema a gerar todas as mensagens de erros.
- Projete entradas que causem overflow dos buffers;
- Repita a mesma entrada ou série de entradas várias vezes;
- Force a geração de saídas inválidas;
- Force os resultados de cálculos a serem muito grandes ou muito pequenos.

Teste de releases

A melhor abordagem a ser usada para validar requisitos de sistema é a **baseada em cenários**.

De acordo com o cenários, cria-se uma série de testes que podem ser aplicados. Esses testes devem incluir entradas válidas e inválidas e que gere saídas válidas e inválidas.

Usualmente os testes são organizados dos cenários mais prováveis para os mais incomuns ou excepcionais.

Teste de desempenho

Os testes de desempenho asseguram que o sistema pode operar na carga necessária. O objetivo é demonstrar que o sistema atende aos requisitos, assim como também descobrir problemas e defeitos.

Para os testes, pode-se construir um perfil operacional, que reflete uma combinação real de trabalho a que o sistema será submetido ou com testes baseados nos limites do sistema – ***teste de estresse***.

Teste de componentes

Teste de componentes

O teste de componentes – *teste de unidade* – é o processo de teste de componentes individuais do sistema. O objetivo é expor possíveis defeitos nos componentes.

Existem diferentes tipos de componentes que podem ser testados:

- Funções ou métodos individuais de um objeto;
- Classes de objeto com vários atributos e métodos;
- Componentes compostos que constituem diferentes objetos ou funções.

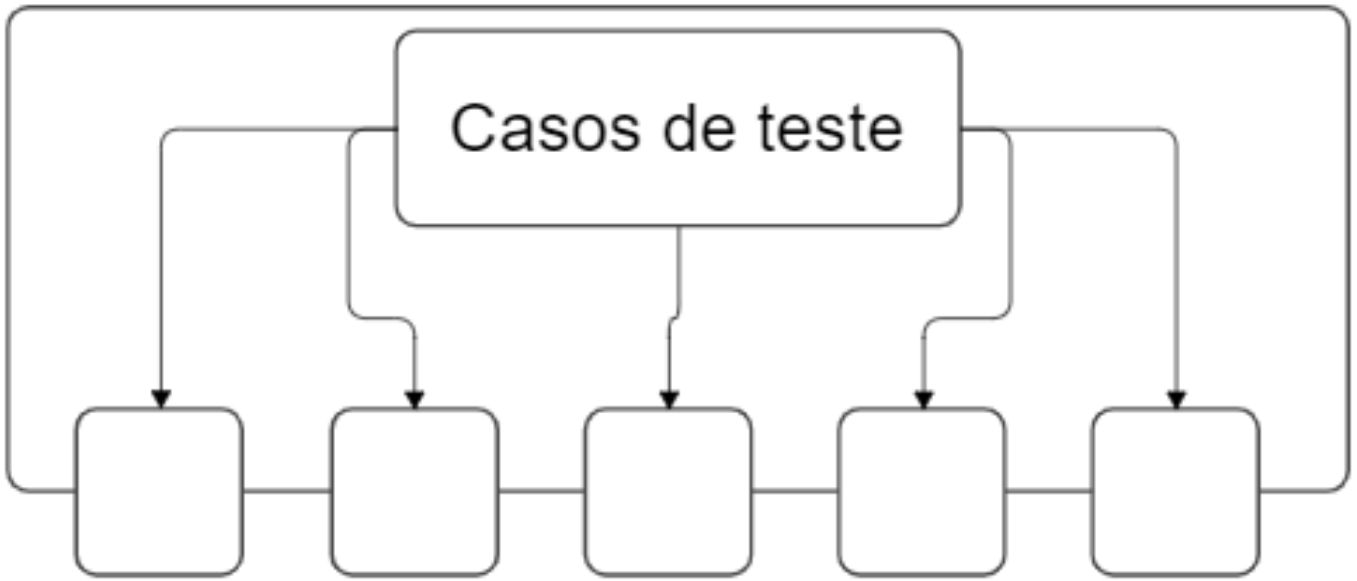
Teste de componentes

As funções ou métodos individuais são os tipos mais simples de componentes e seus testes são um conjunto de chamadas dessas rotinas com diferentes parâmetros de entrada.

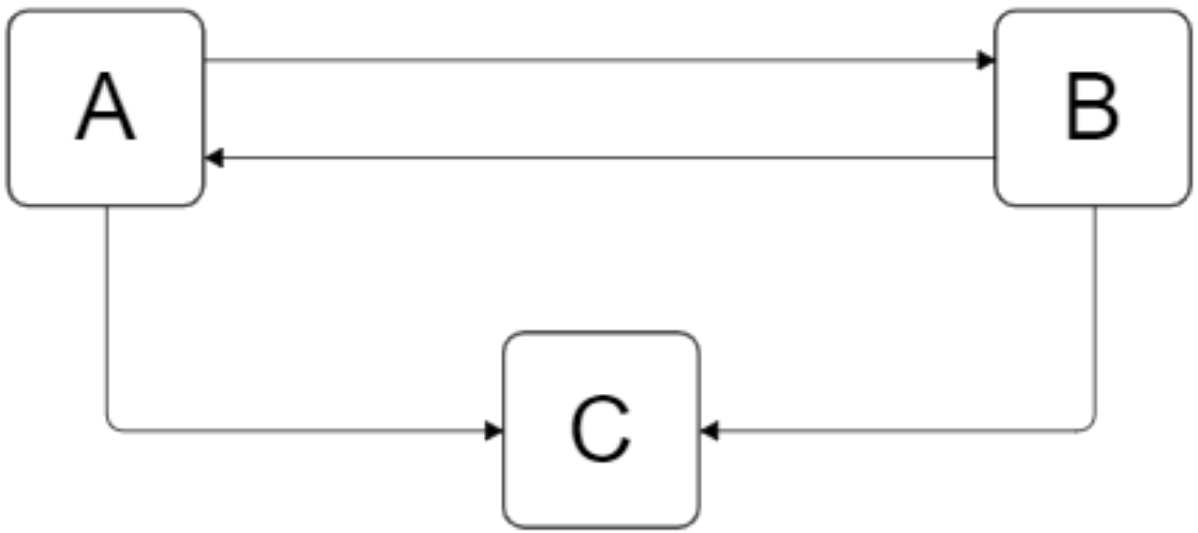
Os testes de classes de objeto devem testar isoladamente todas as operações associadas ao objeto; atribuir e interrogar todos atributos associados ao objeto; e, simular todos os estados possíveis do objeto.

Teste de interfaces

- Um sistema não possui apenas funções ou objetos simples, mas sim componentes compostos constituídos de vários objetos que interagem. Tais funcionalidades são acessadas através de suas **interfaces**.
- O teste de interfaces é particularmente importante para desenvolvimento orientado a objetos e baseado em componentes, devido as possibilidades de combinações com componentes em diferentes sistemas.



Interfaces



Componentes

Testes de interfaces

Existem diferentes tipos de interfaces:

- Interfaces de parâmetros
  - Interfaces onde dados ou referências a funções são passadas de um componente para outro.
- Interfaces de memória compartilha
  - Interfaces que compartilham um bloco de memória, onde os dados são colocados na memória por um subsistema e recuperados por outros.
- Interfaces de passagem de mensagem
  - Interfaces em que um componente solicita um serviço de um outro passando uma mensagem para ele.

Testes de interfaces

Os erros de interface são divididos em três categorias:

- Mau uso de interface
  - Um componente chama algum outro componente, e pode passar a ordem incorreta ou uma quantidade incorreta de parâmetros.
- Mau entendimento da interface
  - Um componente chamador perde a especificação da interface do componente chamado e faz suposições sobre seu comportamento.
- Erros de timing
  - Ocorrem em sistemas de tempo real que usam memória compartilhada ou uma interface de passagem de mensagens.

Testes de interfaces

Diretrizes gerais para teste de interfaces:

- Exame o código a ser testado e liste cada chamada a componentes externos;
- Teste sempre a interface com ponteiros nulos onde os ponteiros são passados por meio de uma interface;
- Use o teste de estresse;
- Projete testes que causem a falha do componente onde este é chamado por meio de uma interface de procedimento.
- Projete testes que variam a ordem na qual os componentes que compartilham memória são ativados.

Projeto de casos de teste

Projeto de casos de teste

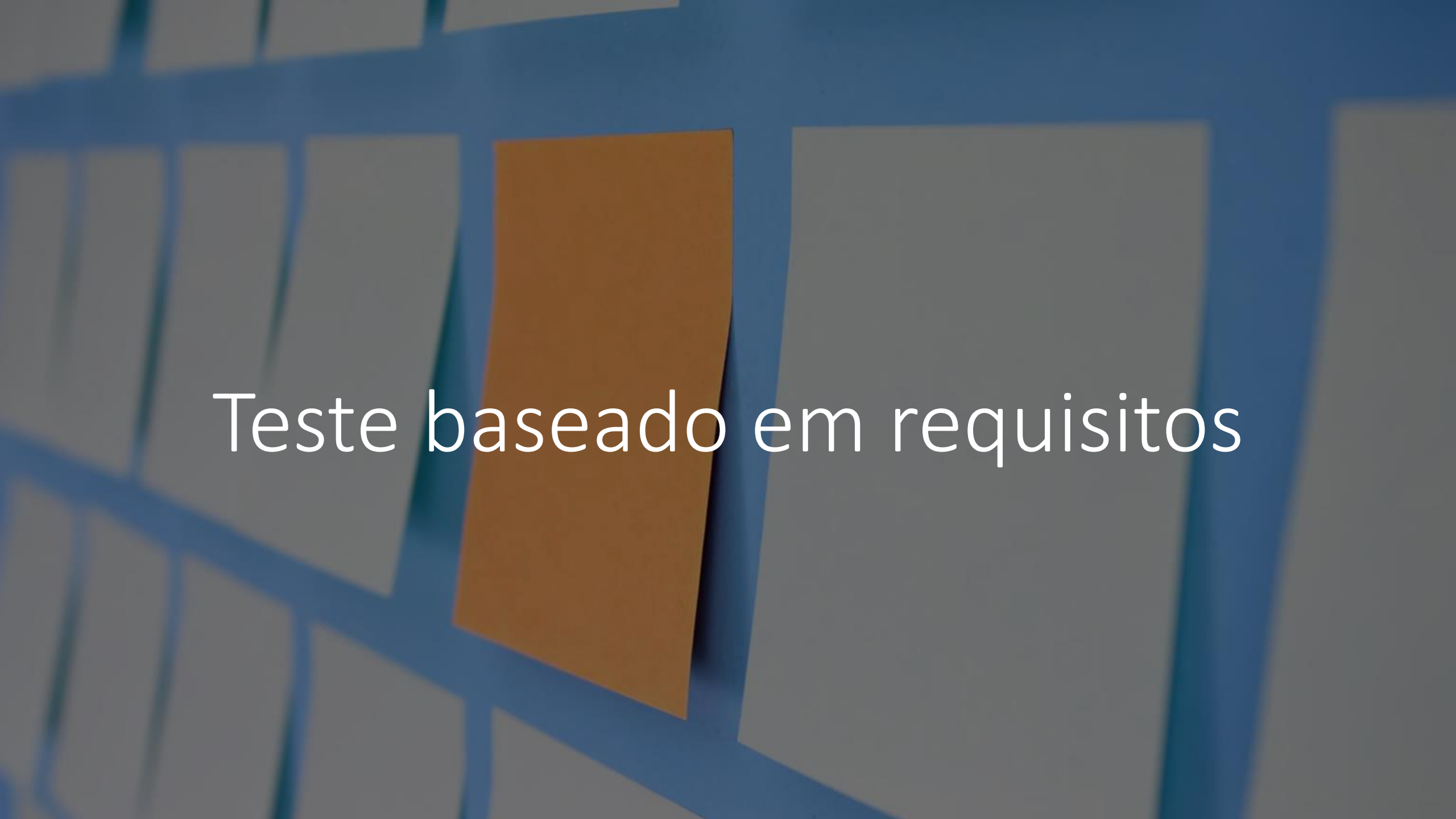
Projetar casos de teste é criar um conjunto de entradas e saídas **esperadas**, para descobrir possíveis defeitos do programa e demonstrar que o sistema **atende aos requisitos**.

Para projetar uma caso de teste, deve-se selecionar uma característica do sistema, executar aquela característica e documentar todas as saídas esperadas.

Projeto de casos de teste

Para projetar casos de teste, existem várias abordagens:

- Teste baseado em requisitos
  - Testes projetados para testar os requisitos de sistema
- Teste de partições
  - Testes que identificam partições de entrada e saída, de modo que o sistema processe as entradas de todas as partições e gere as saídas em todas as partições.
- Teste estrutural
  - Teste que é usado o conhecimento da estrutura do programa para projetar testes que exercitem todas as partes do programa.



Teste baseado em requisitos

Teste baseado em requisitos

Testes baseados em requisitos são uma abordagem sistemática para projetar casos de testes no quais cada requisito é considerado e um conjunto de testes é derivado para ele.

Testes baseados em requisitos são teste de validação.

Para testar um requisito, normalmente são escritos vários testes para assegurar que o requisito foi coberto.


Teste de partições

Teste de partições

O particionamento de equivalência divide as entradas do usuário na aplicação em partições – *classes de equivalência*, e então as divide em faixas de valores possíveis, para que então, um desses valores seja eleito como base para o teste.

Existem partições de equivalência para valores válidos – aqueles que devem ser aceitos pelo sistema, e valores inválidos – aqueles que devem ser rejeitados pelo sistema.

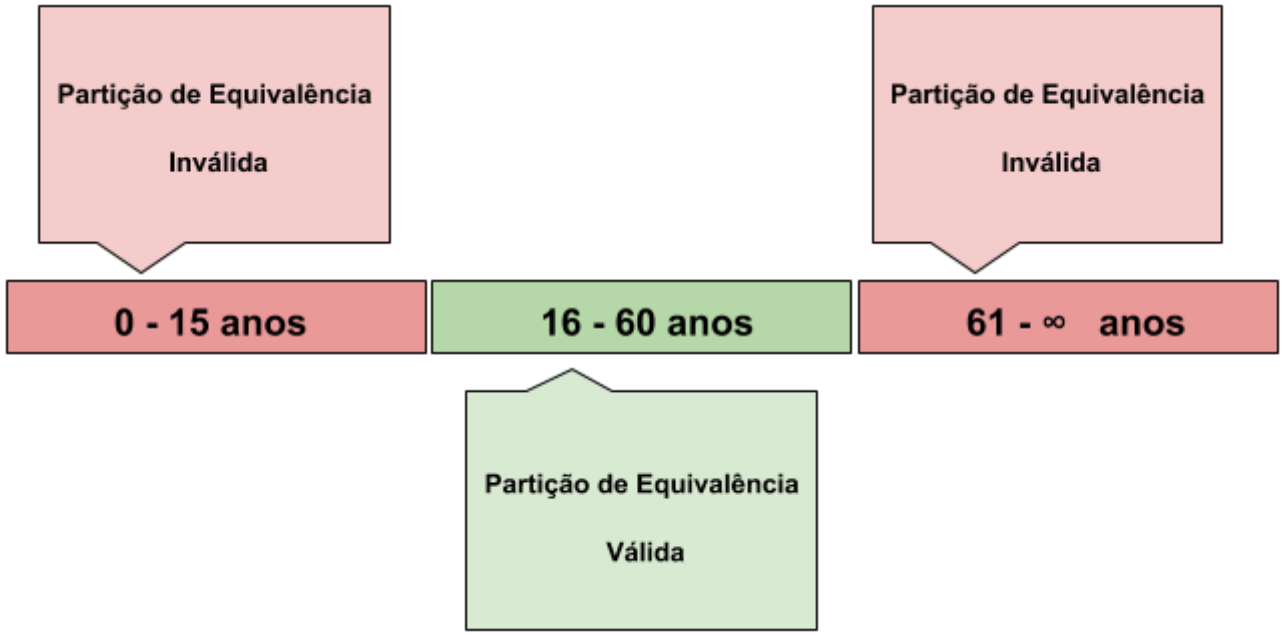
Exemplo

Considere um sistema de recursos humanos que processa pedidos de emprego bom base na idade de uma pessoa e que possui as seguintes regras de negócio:

Pessoas menores de 16 anos não devem trabalhar;

Pessoas entre 16 e 60 anos podem trabalhar;

Pessoas com mais de 60 anos não podem trabalhar.



Fonte: https://medium.com/revista-tspi/t%C3%A9cnica-de-teste-particionamento-de-equival%C3%AAncia-d32a7d689d82

Teste estrutural

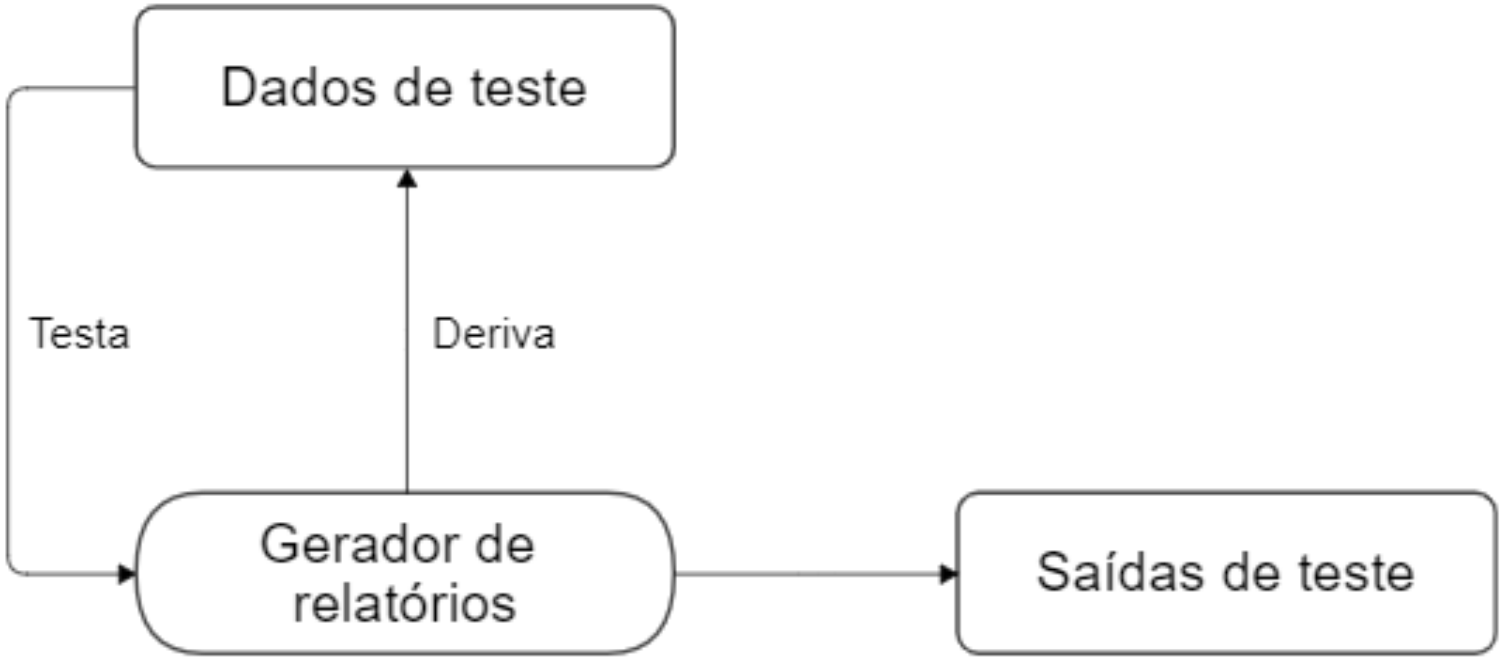
Teste estrutural

O teste estrutural é uma abordagem para projetar casos de teste na qual os testes são **derivados do conhecimento da estrutura** e da **implementação** do software.

Essa abordagem é, algumas vezes, chamada de teste caixa branca.

Os testes do tipo caixa branca são aqueles em que a estrutura interna, código, design, intencionalidade e estratégias de desenvolvimento são conhecidas pelo testador e consideradas no ato do teste. Normalmente são mais relacionados à infraestrutura de software do que à usabilidade.

Teste estrutural



Teste de caminho

Teste de caminho

O teste de caminho é uma estratégia de teste estrutural cujo objetivo é exercitar cada caminho independentemente da execução de um componente ou de um programa.

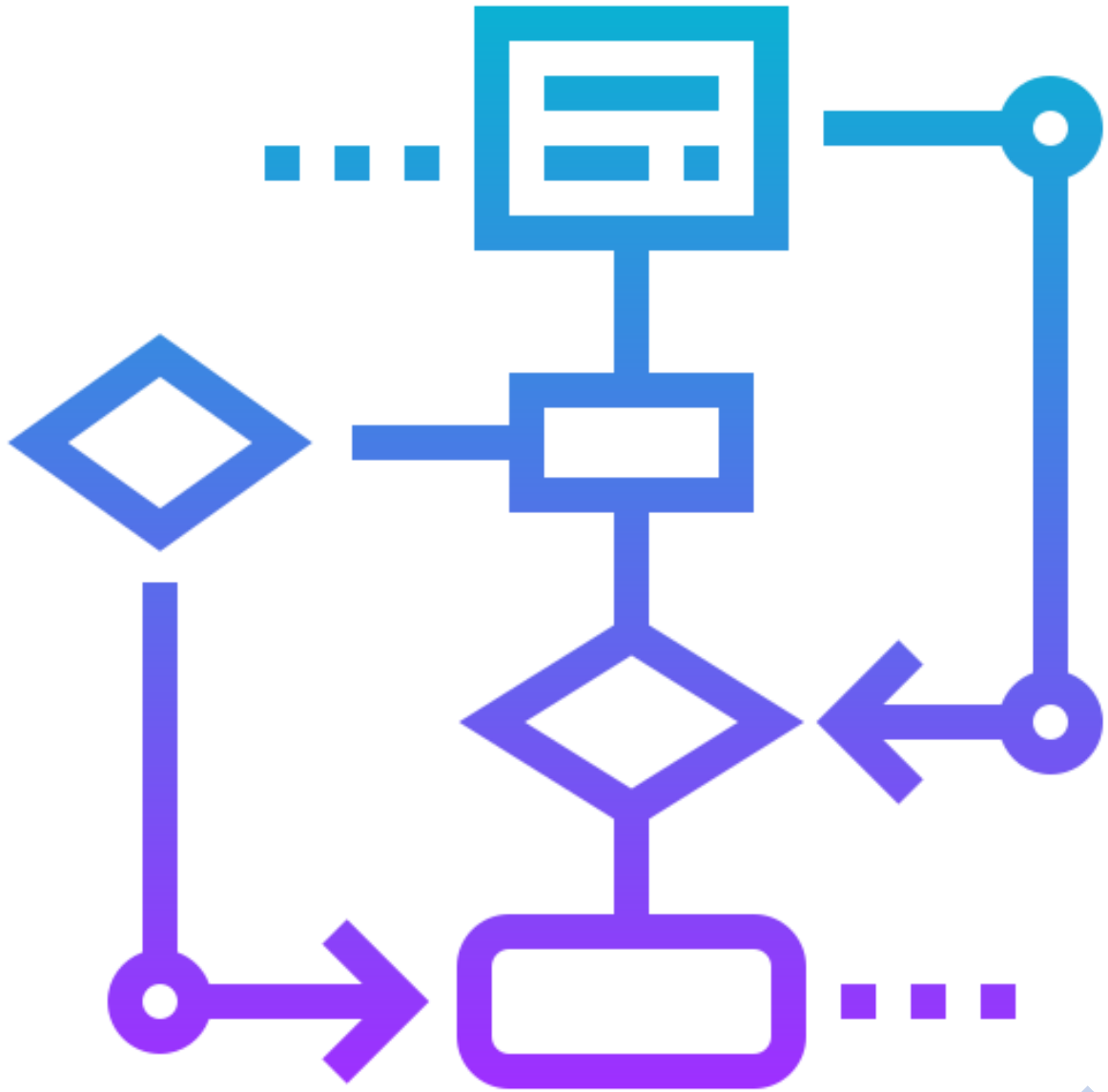
Uma vez um caminho executado, todas as declarações terão sido executadas pelo menos uma vez, e todas as declarações condicionais são testadas para ambos os casos, verdadeiro e falsa.

Teste de caminho

O número de caminhos de um programa é proporcional a seu tamanho.

Os testes de caminho são mais usados durante o teste de componente.

O ponto de partida para o teste de caminho é um fluxograma do programa.



Automação de testes

Automação de testes

A fase de testes é dispendiosa e trabalhosa, portanto, as ferramentas de teste estão entre as primeiras ferramentas de software a serem desenvolvidas.

Atualmente, tais ferramentas oferecem uma variedade de recursos e seu uso pode reduzir significativamente os **custos** de teste.

Automação de testes

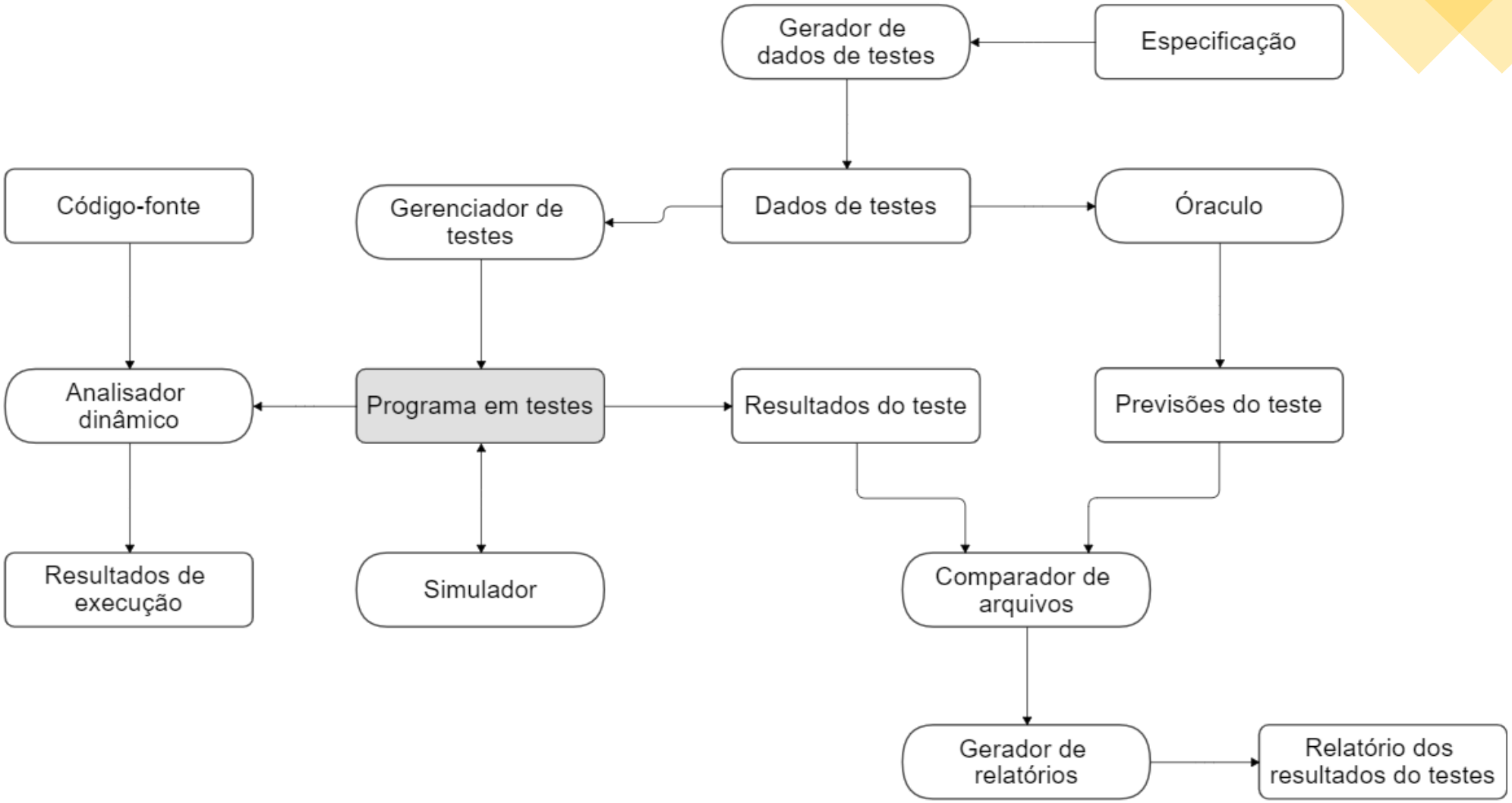
As ferramentas de testes podem incluir:

- Gerenciador de testes
  - Mantém o acompanhamento dos dados de teste, resultados esperados e os recursos de programa testados.
- Gerenciador de dados de teste
  - Gera os dados de testes, que pode ser seleção de dados de um banco de dados ou por geração aleatória.
- Oráculo
  - Gera previsões de resultados de teste esperados. Podem ser versões anteriores do programa ou protótipos do sistema.

Automação de testes

As ferramentas de testes podem incluir:

- Comparador de arquivos
  - Compara os resultados dos testes com os resultados anteriores e relata as diferenças entre elas.
- Gerador de relatórios
  - Fornece recursos de definição e de geração de relatórios para os resultados de teste.
- Analisador dinâmico
  - Adiciona código ao programa para contar o número de vezes em que cada declaração foi executada.
- Simulador
  - Simuladores de máquina, interface, entrada e saída.


Gerador de dados de testes
Especificação
Código-fonte
Gerenciador de testes
Dados de testes
Óraculo
Analisador dinâmico
Programa em testes
Resultados do teste
Previsões do teste
Resultados de execução
Simulador
Comparador de arquivos
Gerador de relatórios
Relatório dos resultados do testes

Ferramentas para testes

Testar um software é uma etapa indispensável no desenvolvimento de qualquer aplicação para garantir a qualidade do software.

Ferramenta de testes para aplicações web pelo browser. Utiliza testes de regressão. Gratuita *open source*, oferece recursos de reprodução e gravação para este tipo de teste.

Plataforma que automatiza testes do celular, desktop e aplicações web. Permite utilização de diversas linguagens, como JavaScript, VBScript e Python, além de ter as funções de teste orientado por dados, teste regressão e distribuído.

Ferramenta altamente ajustável e fácil de usar, com IDE completa e APIs abertas para especialistas em automação. Suporta teste de ponta a ponta em desktop, web e dispositivos móveis. Apenas versão paga.

Framework de código aberto, com painel próprio que exibe exatamente o acontece durante os testes. Possui versão gratuita e paga.

Ferramenta para testes automatizados entre navegadores. Possui suporte as principais linguagens e frameworks. Versão paga e gratuita.

Ferramenta de testes para aplicações web e móveis. Ótima opção para equipes pequenas e médias. Versão paga e gratuita.

Resumindo

Os testes mostram **somente a presença de erros** e não demonstram que **não existam defeitos remanescentes**.

Testes de componentes é de responsabilidade do desenvolvedor de componentes.

Testes do sistema são, geralmente, de responsabilidade de outra equipe.

O teste de integração é a atividade inicial de testes, onde são testados os componentes integrados para detectar defeitos.

Resumindo

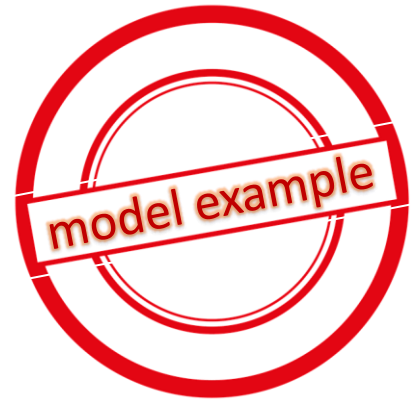
O teste de releases concentra-se em testar releases do cliente e em validar se o sistema atende aos requisitos.

O teste de interfaces tenta descobrir defeitos nas interfaces dos componentes compostos.

A automação reduz os custos de teste pelo apoio ao processo de teste com uma variedade de ferramentas de software.

Modelo de Plano de Teste

Documento



Plano de Teste

nome do sistema

versão x.x

Histórico das alterações

Data	Versão	Descrição	Autor (a)
dd/mm/aaaa	x.x	Release inicial	Godofredo

1 – Introdução

Este documento descreve os requisitos a testar, os tipos de testes definidos para cada

Referências Bibliográficas

DELAMARO, Marcio. **Introdução ao Teste de Software**. Rio de Janeiro: Grupo GEN, 2016. E-book. ISBN 9788595155732. ;

SOMMERVILLE, Ian. **Engenharia de software**. 8. ed. São Paulo: Pearson, 2007. ISBN 978-85-88639-28-7

Atividade

- Escreva um cenário que possa ser usado como base para teste para um sistema de lançamento de notas.