

TESTES DE SISTEMAS

O que são *

E qual sua finalidade?

Introdução

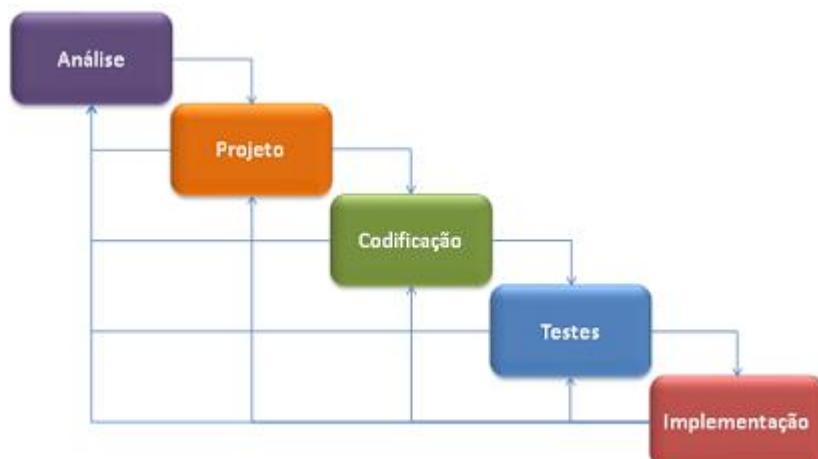
Tudo o que consumimos no dia a dia é testado antes de chegar às prateleiras, seja para garantir nossa segurança, como no caso de produtos físicos, seja para garantir nossa saúde, como no caso de alimentos.

Então, se tudo é testado, programas de computador também são, correto? Correto. No entanto, se softwares não são palpáveis, como são testados? Quais os parâmetros para esses testes e, sobre-tudo, o que eles asseguram?

Testes de sistemas

× O que é um teste de software?

O teste faz parte de um processo no desenvolvimento do programa, podendo ser feito pelos próprios desenvolvedores ou, em alguns casos, feito por profissionais especializados na área. O procedimento tem como objetivo antecipar e corrigir falhas e bugs que apareceriam para o usuário final.



Testes de sistemas

Adotando uma definição mais técnica, teste de software é todo e qualquer procedimento que ajuda a determinar se o programa atinge as expectativas para as quais foi criado (DIAS NETO, 2010).



Testes de sistemas

Justamente por conter essa característica preventiva, podemos dizer que os testes de software são de natureza destrutiva, jamais construtiva (DIAS NETO, 2010).



Defeitos x Falhas

Figura 1.1 Defeito versus erro versus falha.



Defeitos x Falhas

O International Software Testing Qualifications Board (ISTQB) traz as seguintes definições:

■ **Erro:**

- Falha humana; produz resultado incorreto.
- É, por exemplo, a falha na escrita de um código específico.

■ **Defeito:**

- Resultado de um código mal escrito, ou seja, um erro.
- Causa anomalia no funcionamento do sistema. O usuário final normalmente não vê o defeito propriamente dito.
- Também é conhecido como *bug*.

■ **Falha:**

- Quando um código que apresenta um defeito é executado, temos uma falha.
- Funcionamento inesperado das funções do software.
- É a camada que chega aos olhos do usuário.

Figura 1.1 Defeito versus erro versus falha.



Defeitos x Falhas

Podemos dizer, então, que temos um efeito cascata:

1. Toda falha precisa de um defeito para ocorrer, mas nem todo defeito acarreta uma falha.
2. Todo defeito precisa de um erro para ocorrer, mas nem todo erro acarreta um defeito.
3. Toda falha sempre se origina em um erro (ISTQB, 2016).

Figura 1.1 Defeito versus erro versus falha.



Exemplo



No caso de solicitar um saque de R\$100,00 mas o caixa eletrônico te fornecer 100 folhas de cheque, temos um erro, defeito ou falha?

Exemplo

Neste caso



Conceito	Definição	Onde acontece
Erro	Código (da função "saque") mal escrito. Esse código mal escrito pode ou não acarretar defeitos.	No universo físico, enquanto o programador escreve o programa.
Defeito	Quando o cliente seleciona "saque" e ativa o código mal escrito, que reflete um funcionamento anormal, não esperado pelo sistema.	No universo da informação, ou seja, na troca de dados e chamados entre elementos do programa.
Falha	Ato de entregar cheques, e não notas, decorrente da anomalia da execução de um código com erro. É o resultado que não bate com a expectativa.	No universo do usuário. É o resultado final após a execução das tarefas. O usuário não vê, necessariamente, erros e defeitos. Ele tem acesso apenas às falhas.

Fonte: adaptado de Dias Neto (2010).

Testes de sistemas



Assim, é possível concluir que o teste de software detectará apenas falhas. Ora, se a intenção dos testes é evitar que o programa saia daquilo que é esperado, então, ele é focado justamente no uso final, no universo do usuário.

Teste x Depuração

Após a realização dos testes é necessário corrigir eventuais falhas em um processo conhecido como depuração

Porque acontecem tantas falhas?

que toda falha é originada de um erro humano. É praticamente impossível desenvolver um software que passe pelos testes sem que nenhuma falha seja encontrada. E isso não está relacionado com a capacidade e as habilidades de quem programou esse software, muito menos com as tecnologias, os recursos e os métodos utilizados. Erros acontecem simplesmente porque tudo o que é criado é humano, e o ser humano é falho. É inevitável.

Exemplo

quando um funcionário da linha de produção de uma fábrica vai bater o ponto e, sem perceber, acessa uma área do software destinada apenas para gestores, temos uma falha no programa. Isso pode ser causado, entre outros motivos, por:

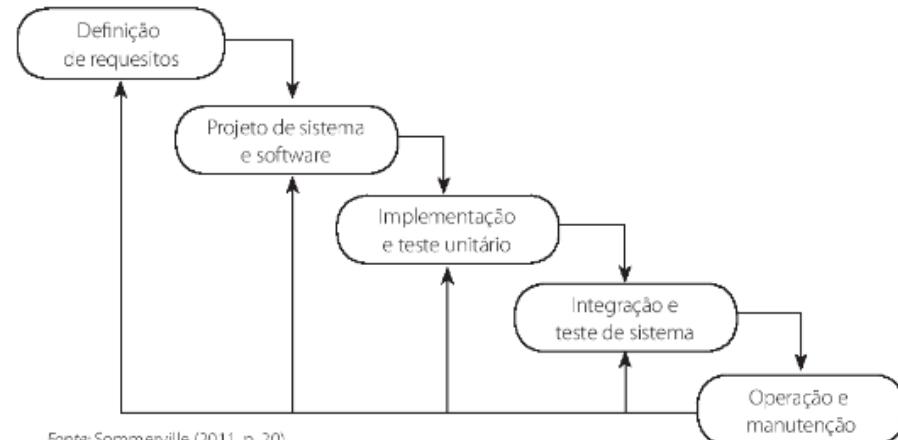
- códigos errados e incompletos;
- limitações de hardware ou software;
- inexistência de diferenciação entre tipos de usuários (produção, gestão, supervisão, direção etc.);
- defeito no controle de algoritmos, entre outros.

Independentemente do que possa ter causado determinada falha, sempre há maneiras de corrigi-la.

Principais causas de erros

Pensando de modo prático, podemos citar duas grandes causas de erros:

1. Uma equipe muito grande e variada.
2. Diversas modificações ao longo das etapas do ciclo de desenvolvimento do software.

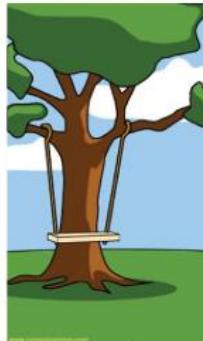


Fonte: Sommerville (2011, p. 20).

Principais causas de erros



Como o cliente
expliou



Como o líder de
projeto entendeu



Como o analista
planejou



Como o programador
codificou



O que a assistência
técnica instalou



O que o cliente
realmente necessitava

O analista de testes de software

O programador não trabalha sozinho, e não é qualquer pessoa da equipe que realizará os testes do software. Mais do que simplesmente submeter o programa a testes, é preciso responder a algumas perguntas: Quais testes? Por que serão feitos? Quais os resultados esperados? E quais os resultados obtidos?

O analista de testes de software

O que você entende por qualidade? A maioria das pessoas responderia que qualidade se resume a algo bem-fcito. Vamos pensar, então, no ramo da informática. Todo programa de computador tem uma razão de existir. Todo software tem um propósito. A fluidez com que esse propósito é alcançado durante o uso do software é o que determina se o programa tem ou não qualidade.

Tipos de testes

Nenhum teste é feito ao acaso. Todos têm um porquê e uma fase certa para serem realizados. Dizemos isso porque, como você viu no tema anterior, os critérios podem aparecer em qualquer fase da elaboração do software.

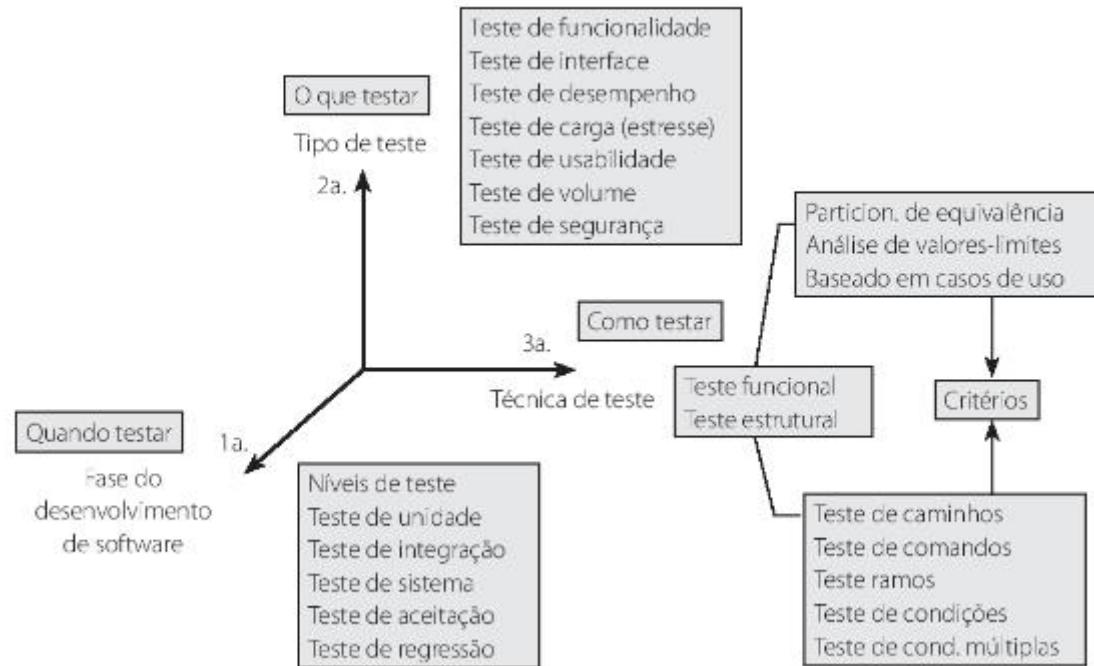
Tipos de testes



Quais são?

Tipos de testes

Figura 1.4 Relação entre níveis, tipos e técnicas de teste.

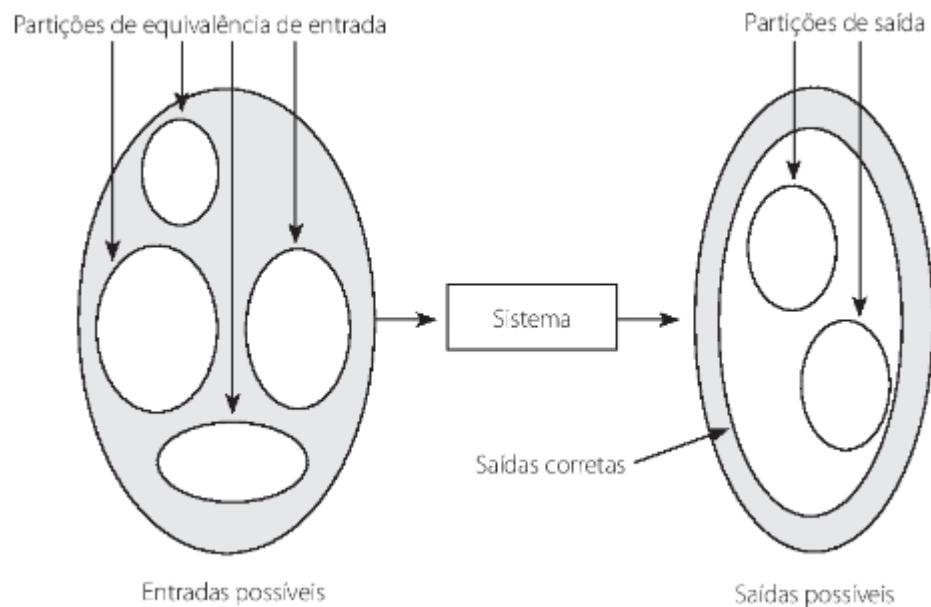


Teste unitário

Segundo Wazlawick (2011), o teste unitário, também conhecido como teste de unidade, possibilita que cada módulo do sistema seja analisado de maneira individual, verificando se sua funcionalidade está correta. Essa avaliação pode ser feita até mesmo com trechos de códigos específicos.

Teste unitário

De acordo com Sommerville (2011), devemos considerar entradas corretas e incorretas justamente para verificar se as entradas que se aplicam ao esperado fornecem resultados esperados, enquanto as entradas incorretas nos mostram que tipos de saídas geram e quais falhas esse processo pode conter.

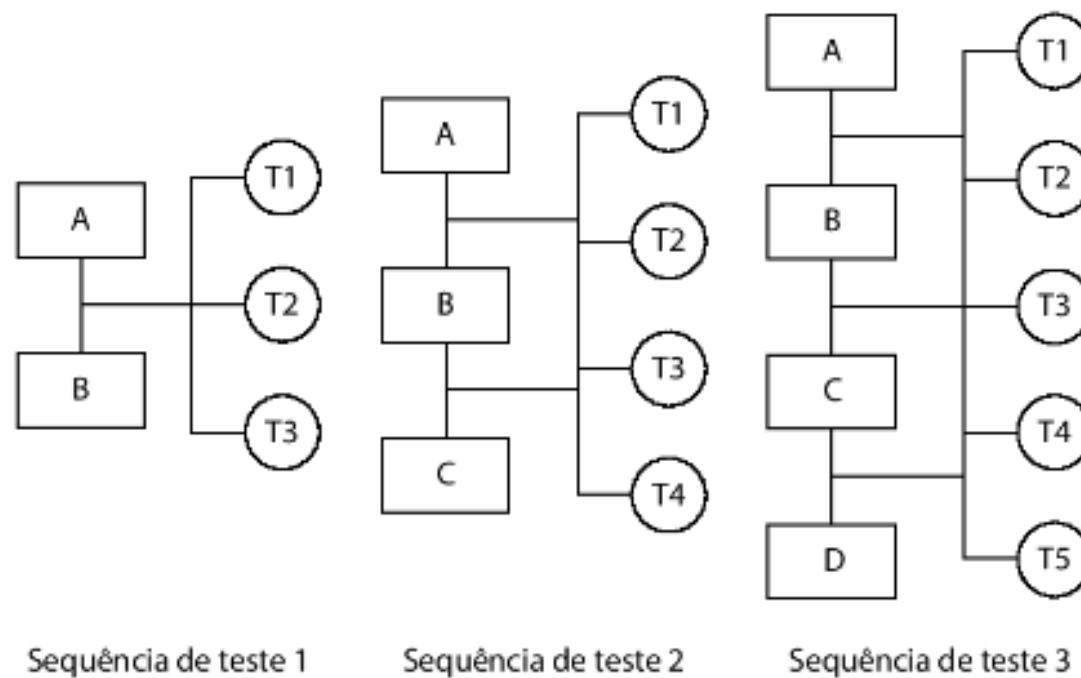


Teste de integração

O objetivo desse teste é verificar se a comunicação entre módulos é feita de modo correto. Diferente do teste unitário, o de integração visa, justamente, agrupar os módulos que têm funções específicas, até formar um sistema completo, ou seja, você vai integrar seus módulos e testá-los (WAZLAWICK, 2011).

Teste de integração

Figura 1.6 Teste de integração incremental.



De modo geral:

1. Primeiro, testam-se os módulos individualmente pelo teste de unidade, a fim de verificar falhas específicas nas funções de cada módulo.
2. Em seguida, agrupam-se os módulos para verificar erros na interface entre eles e medir seus níveis de desempenho e confiabilidade. Isso é feito por grupos, e não considerando o sistema completo.
3. Por fim, após testar todos os grupos de integração, realiza-se o teste de sistema, que será apresentado a seguir.

Teste de sistema

Grosso modo, é um teste no qual é possível encontrar falhas às quais o usuário final pode ter acesso. Wazlawick (2011) afirma que testes de sistemas são avaliados do ponto de vista do usuário. Assim, não exigem conhecimentos da lógica do programa – as linhas de código, a interação entre elementos, entre outros. O que está em jogo aqui é a funcionalidade geral do programa. Eles avaliam, dentre vários itens:

- A qualidade da interface gráfica.
- O funcionamento correto das funções do sistema.
- Se as precondições das operações no projeto são atendidas

Observação

Sobre esses três primeiros tipos de testes que estudamos, podemos ter uma noção geral de abstração: quanto mais específico (como o teste unitário), menos abstrato ele será. Quanto mais geral e mais amplo for seu alcance, como o teste de sistema, mais genérico e abstrato ele será. Veja:

Tipo de teste	Nível de abstração
Sistema	Ascendente
Integração	
Unitário	Descendente

Pergunta 1

1. Nem sempre um aplicativo precisa ser livre de erros antes de ser entregue ao cliente. Explique por quê.

Testes de aceitação

Os testes de aceitação, segundo a IBM, são parecidos com os testes de sistema. Mas essa verificação é muito mais voltada para a relação do usuário final com o programa do que com o sistema propriamente dito, que o usuário não vê (linhas de códigos e afins).

Testes de aceitação formal

Segundo dados da IBM, os testes de aceitação formal são projetados de acordo com dois parâmetros:

1. O que será testado.
2. Como será testado.

Esse teste pode ser feito de modo totalmente automatizado. Porém, há também a opção de o teste ser feito por um grupo de pessoas da organização à qual o software se destina. Ainda segundo a IBM, se nenhuma dessas opções for escolhida, um grupo de usuários pode ser previamente selecionado para realizar os testes.

Testes de aceitação formal

Vantagens	Os recursos e as funções que serão testados no sistema são conhecidos pelos testadores.
	Os detalhes e a finalidade do teste são conhecidos e, por isso mesmo, mensuráveis.
	Possibilidade de automação do teste, o que permite a realização do teste de regressão, que você verá adiante.
	Possibilidade de monitoração e mensuração do progresso do teste.
	Critérios de aceitabilidade conhecidos pelos testadores.
Desvantagens	Grande planejamento.
	Uso significativo de recursos.
	Pode-se chegar aos mesmos resultados do teste de sistema.
	São procurados apenas defeitos já esperados no uso final.

Testes de aceitação informal

Esse tipo de teste geralmente é realizado pela empresa que utilizará o software, e não pela equipe que o desenvolveu. Segundo Graham (2007), requisitos como *restauração, backup, recuperação de erros, verificação de manutenção de tarefas e vulnerabilidade* são testados dentro dos testes de aceitação informal.

Testes de aceitação informal

Vantagens	Recursos e funções testados também são conhecidos (assim como no teste de aceitação formal).
	Possibilidade de monitoração do progresso do teste.
	Critérios de aceitabilidade conhecidos.
	São apresentados e identificados erros mais subjetivos que no teste de aceitação formal, justamente por depender da interação de cada testador. Isso aumenta a possibilidade de correção de novos erros e, o mais importante, a prevenção de erros futuros.
Desvantagens	Grandes recursos de gerenciamento.
	Não é possível controlar o modo que o sistema será testado.
	Existe a possibilidade de os usuários testadores se adaptarem e gostarem do sistema e, exatamente por isso, não encontrarem possíveis erros presentes.
	Existe, também, a possibilidade de os usuários testadores compararem o sistema que estão testando com uma versão mais antiga. Isso desvia o foco do teste e, consequentemente, atrapalha a detecção de erros reais.
Os recursos do teste de aceitação informal não são controláveis.	

Testes beta

Dos três tipos de subtestes do teste de aceitação, o beta é o menos controlável. Isso porque seus requisitos são de inteira responsabilidade do testador. Geralmente é feito com usuários que não têm nenhum vínculo com a organização detentora do software. Isso significa que eles realmente testam o software no ambiente real (GRAHAM et al., 2007).

Testes beta

Vantagens	Pode ser realizado diretamente pelos usuários finais.
	Aquele que participa do teste tem sua satisfação elevada.
	Defeitos extremamente subjetivos são revelados.
Desvantagens	Nem todas as funções ou recursos podem ser testados, uma vez que o usuário é livre para usar o programa como deseja.
	A mensuração dos níveis de teste é quase impossível.
Desvantagens	Assim como no teste de aceitação informal, os usuários podem acabar se concentrando na comparação do sistema com uma versão antiga. Além disso, também podem se adaptar e gostar da versão beta. Em ambos os casos, possíveis erros podem não ser identificados.
	Os critérios de aceitabilidade não são conhecidos – são extremamente subjetivos e individuais.

Testes de regressão

O teste de regressão é, em linhas gerais, a recapilação dos testes que você viu anteriormente, nas novas versões do sistema, para verificar possíveis novos erros após a solução de erros anteriores (GRAHAM et al., 2007).

Isso ocorre porque as alterações que fazemos quando descobrimos falhas podem resultar em novas falhas em áreas do programa onde antes não existia nada de errado. Assim, teste de regressão “reverifica” o que foi visto.

Exemplo

Dois testes são feitos para identificar, cada um, os supostos erros A e B. Apesar o erro A é constatado, ou seja, o erro B não existe. Ao corrigir a falha A, desenvolve-se, involuntariamente, a falha B.

Naturalmente, o teste que identificou o erro A é reaplicado para verificar se ele não existe mais. Porém, se não aplicarmos novamente o teste que identificou a ausência do erro B, jamais descobriremos que esse erro apareceu somente após a alteração para correção da primeira falha. Essa reaplicação é, justamente, o teste de regressão.

Pergunta

Quais as vantagens de utilizar o teste de regressão?

Teste de integridade de dados

O teste de integridade de dados consiste no processo de validação da confiabilidade e da integridade dos dados de um determinado sistema.

Esse teste vai assegurar a robustez do programa, sua capacidade de resistir a falhas. É realizado em diversas etapas do desenvolvimento.

Teste de configuração e instalação

Enquanto o teste de instalação verifica possíveis erros em vários cenários de instalação – hardware, espaço insuficiente de disco, falta de energia durante a instalação – e possíveis falhas que essas situações podem acarretar, o teste de configuração verifica a capacidade do programa ser executado e fornecer todos os recursos em diferentes plataformas físicas – uma vez que, ainda que as plataformas não sejam todas idênticas, a função final do programa deve se manter inalterada (UFPE, 2006).

Teste de performance

Existem várias formas de verificar a performance do seu software. Isto é, quão bom ele é naquilo que foi criado para fazer, ou quanto ele suporta em questão de trabalho. Esse teste se subdivide em três tipos: são os *testes de estresse, de carga e de estabilidade*.

Teste de performance

Teste de estresse

É um tipo de teste não funcional (GRAHAM et al., 2007). Todo sistema é construído com base em alguns requisitos mínimos. Mas nem todos os usuários respeitam ou têm esses requisitos exigidos para um funcionamento perfeito.

Teste de performance

Teste de carga

Esse teste busca validar e avaliar os limites nos quais um sistema consegue operar de modo constante e normal (UFPE, 2006).

Teste de performance

Teste de estabilidade

Esse teste verificará por quanto tempo um sistema se mantém estável, funcionando de forma normal e correta, sem sofrer danos e alterações imprevistas (UFPE, 2006).

Pergunta 2

- 2.** O que você entende por testes de performance? Quais são seus tipos e sua importância?

Testes não funcionais

Testes não funcionais são todos os testes que têm foco, obviamente, em requisitos não funcionais. Segundo Graham et al. (2007), o melhor exemplo são os testes de performance, que você viu anteriormente.

Atividades

Teste de *software* é uma atividade que pode ser sistematicamente planejada e especificada. Um teste é bem-sucedido quando um caso de teste descobre um erro e, por consequência, inicia-se _____, um processo que tem por objetivo principal encontrar e corrigir a causa de um erro ou defeito de *software*.

- **A** a verificação por rastreamento
- **B** a depuração
- **C** a validação por rastreamento
- **D** o teste de regressão
- **E** o teste de desempenho

Abordagens de testes

Testes caixa branca x caixa preta?

Você já viu que existem várias maneiras de testar um software, dependendo do objetivo que se pretende alcançar. No entanto, de um modo geral, existem técnicas específicas que são muito utilizadas em modelos de linguagens estruturadas. Por vezes, dependendo do nível de complexidade de um sistema, seria impraticável a aplicação de um teste de estrutura lógica simples.

Abordagem baseada na caixa branca

analisa todos os nós das estruturas por meio de atalhos, considerando apenas resultados significativos.

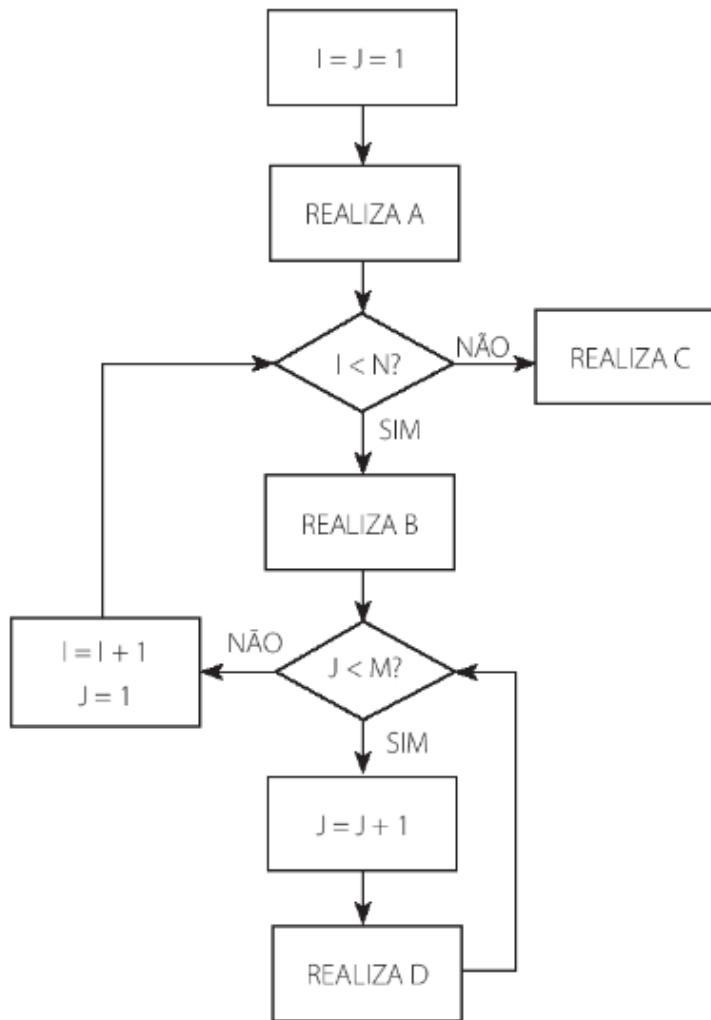
Agora, vamos supor que esse *loop* seja iniciado mn vezes, onde m e n , cada um, equivalha a 100 mil. Isso significa que um único caso de teste deveria completar o *loop* nada menos que 10 bilhões de vezes para que todas as opções de caminhos lógicos fossem testadas e verificadas.

Abordagem baseada na caixa branca

poderíamos atribuir I e J da seguinte maneira:

- Um valor para I que seja menor que n , igual a n e maior que n .
- Um valor para J que seja menor que m , igual a m e maior que m .
- Combinar os três valores de J com os três valores de I.

Figura 1.7 Exemplo de estrutura lógica.



Exemplo

Abordagem baseada na caixa preta

A abordagem da caixa-preta recebe esse nome por considerar o processamento do sistema de forma desconhecida; assim, apenas as entradas e as saídas do sistema são avaliadas (DIAS NETO, 2010).

Como você precisa ter um comportamento prévio esperado, precisará de alguns critérios de testes. Os mais relevantes estão

Abordagem baseada na caixa preta

Particionamento em classes de equivalência

Dias Neto (2008) afirma que os domínios de entrada são partidos em várias classes equivalentes, das quais os testes vão se originar. Isso diminui o número de testes, pois não é necessário repetir o mesmo teste em todas as classes equivalentes.

Abordagem baseada na caixa preta

Para isso, é preciso:

1. Identificar classes de equivalência:
 - a. Condições de entrada.
 - b. Condições de saída válidas e inválidas.
2. Definir casos de testes:
 - a. Enumerar as classes de equivalência.
 - b. Enumerar casos de testes para as classes válidas.
 - c. Enumerar casos de testes para as classes inválidas (DIAS NETO, 2010).

Abordagem baseada na caixa preta

Por exemplo:

- Se a condição de entrada é um intervalo de valor, então, você deve criar uma classe de equivalência dentro do limite (válida), uma abaixo do limite e uma acima do limite (ambas inválidas).
- Se a condição é um conjunto possível de caracteres, cria-se uma classe válida e uma inválida (DIAS NETO, 2010).

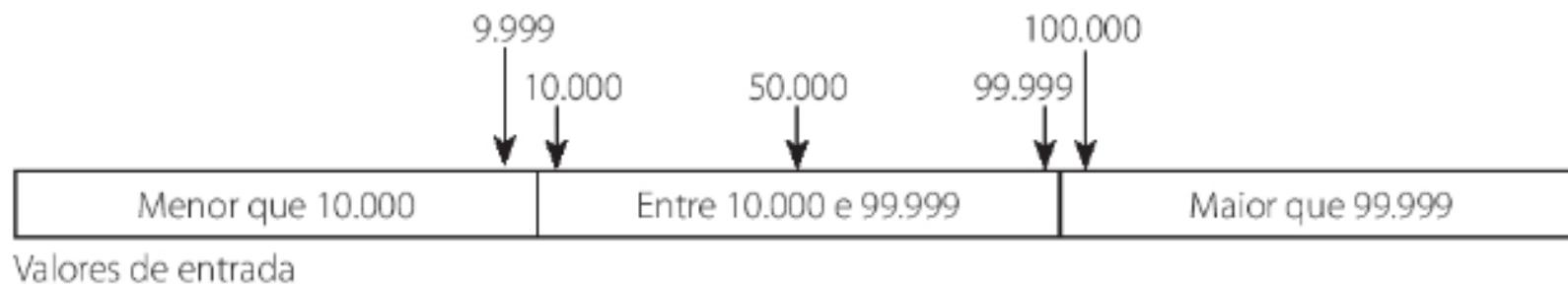
Abordagem baseada na caixa preta

Análise do valor-limite

Por exemplo, em um intervalo de $\{1\dots10\}$, os valores 1 e 10 são mais suscetíveis a falhas. Por isso mesmo, devemos organizar nossos casos de testes não para $\{1\dots10\}$, mas sim para um valor imediatamente acima e um abaixo do limite estipulado – ou seja, casos de testes para $\{1, 10, 0, 11\}$.

Abordagem baseada na caixa preta

Figura 1.8 Partições de equivalência.



Abordagem baseada na caixa preta

Grafo de causa e efeito

Talvez o critério mais importante para atuar com caixa-preta, o grafo de causa e efeito é uma árvore que exemplifica as decisões que o software vai tomar (DIAS NETO, 2010). Essas decisões obviamente variam conforme a entrada. Os resultados já são esperados.

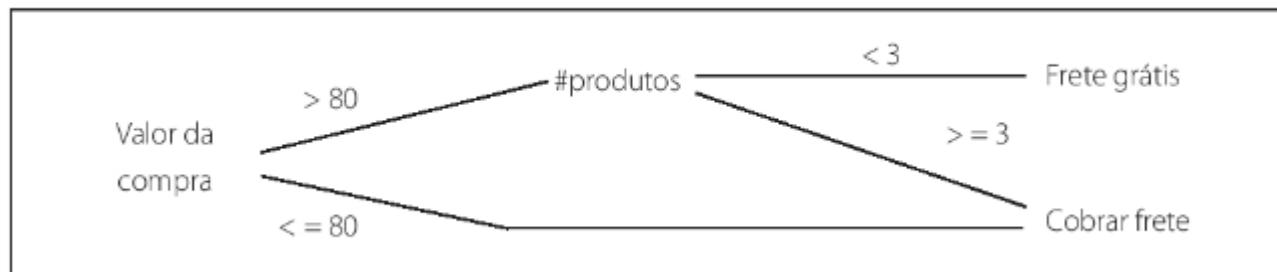
Exemplo

- Se uma compra for menor que R\$ 80,00, deve-se cobrar frete, independentemente do número de produtos.
- Se uma compra for maior que R\$ 80,00 e contiver menos que três produtos, não se deve cobrar frete.
- Se uma compra for maior que R\$ 80,00 e contiver três ou mais produtos, deve-se cobrar frete.

Exemplo

- Se uma compra for menor que R\$ 80,00, deve-se cobrar frete, independentemente do número de produtos.
- Se uma compra for maior que R\$ 80,00 e contiver menos que três produtos, não se deve cobrar frete.
- Se uma compra for maior que R\$ 80,00 e contiver três ou mais produtos, deve-se cobrar frete.

Figura 1.9 Grafo de causa e efeito.



Exemplo

Figura 1.9 Grafo de causa e efeito.

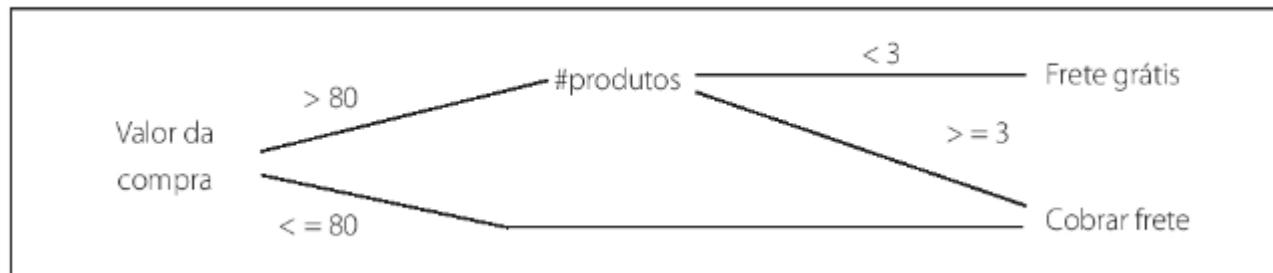


Tabela 1.1 Tabela de decisão.

CAUSA	Valor da compra	> 60	> 60	<= 60
	#Produtos	< 3	>= 3	--
EFEITO	Cobrar frete		V	V
	Frete gratis	V		

Atividades

Em relação a teste de software, é correto afirmar que

- A testes de caixa-preta são realizados por meio da interface do *software*, e são usados para demonstrar que as funções do *software* são operacionais, que uma entrada é aceita corretamente e a saída é produzida corretamente.
- B o teste de *software* de caixa-preta pode aprofundar-se em um exame minucioso dos detalhes processuais, considerando a lógica de execução do *software*. Os caminhos lógicos do *software* são testados, fornecendo casos de teste que exercem conjuntos específicos de condições e/ou *loops*.
- C uma notação simples, chamada de grafo de fluxo ou fluxograma, é comumente utilizada no planejamento e documentação de casos de testes do tipo caixa-preta, em que é possível testar exaustivamente todos os caminhos do programa.
- D o particionamento de equivalência é um método de teste de caixa-branca que divide o domínio de entrada de um programa em classes de dados, a partir dos quais os casos de teste podem ser derivados.

Abordagem baseada na caixa cinza

Dias Neto (2008) mostra que a abordagem da caixa-cinza é a mescla entre os processos de caixa-preta e caixa-branca. Como na caixa-branca, você deve ter acesso às estruturas do sistema, mas também deve ter acesso aos algoritmos do programa, manipulando suas entradas e saídas.

Pergunta 6

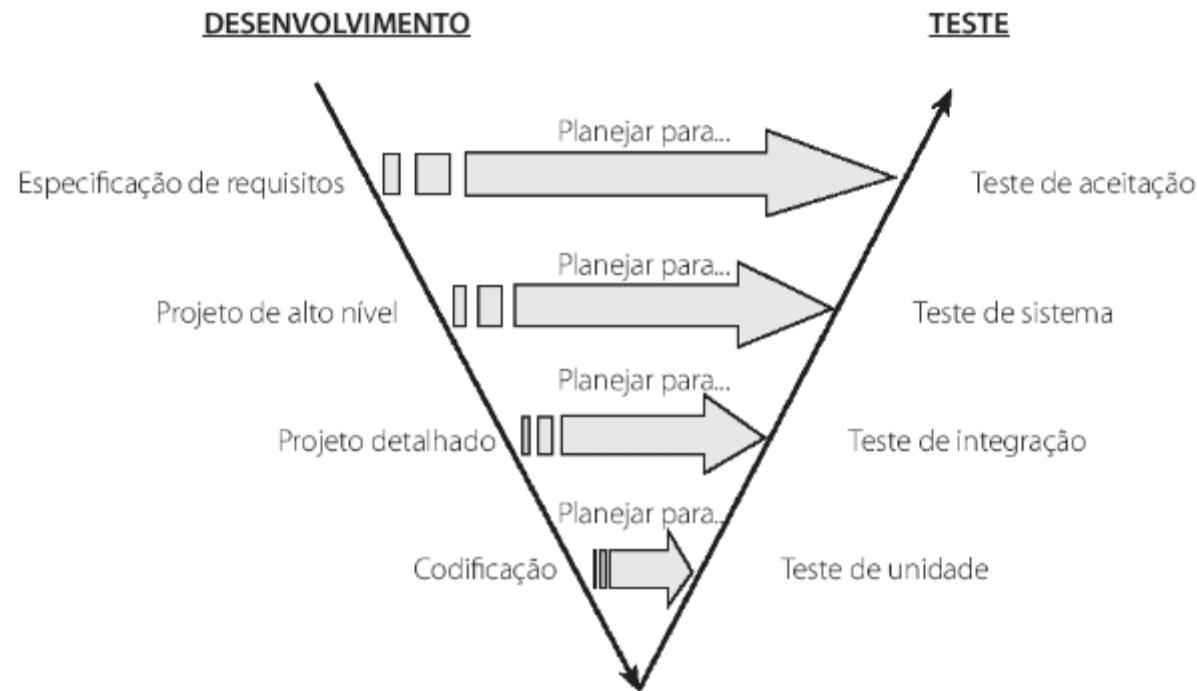
- 6.** Conceitue as abordagens de caixa-preta, caixa-branca e caixa-cinza, citando suas diferenças.

Abordagem baseada em regressão

Nenhuma das fases que sofre uma bateria de testes está imune a mudanças. Muito pelo contrário. Conforme novos erros são encontrados, novas versões são desenvolvidas com as correções. Isso é indiferente em relação à fase em que o software se encontra quando é testado.

Abordagem baseada em regressão

- **Figura 1.10** Modelo V descrevendo o paralelismo entre as atividades de desenvolvimento e teste de software.



Abordagem não funcional

Manter nossos projetos focados em táticas não funcionais pode ser crucial para determinar um bom nível de qualidade. Requisitos como confiabilidade e usabilidade, entre outros, são tão importantes quanto o funcionamento do programa em si. De

Atividades

- () O teste de software é um elemento de tópico mais amplo, conhecido como verificação e validação.
 - A V - F - V - V.
 - B F - F - V - V.
 - C V - F - F - V
 - D V - V - V - F.
 - E F - V - V - V.
- () Dentre as etapas de teste de um software, o teste de integração proporciona a garantia final de que o produto testado satisfaçõa todos os requisitos, inclusive o de desempenho.
- () Uma boa estratégia de testes, além de ter como objetivo principal encontrar erros, também avalia características como utilidade e portabilidade.
- () O teste de recuperação é um teste que força o software a falhar de diversas formas e verifica se a recuperação ocorreu de forma correta.

Atividades

No teste de _____, os módulos são combinados e testados em grupo. Ele sucede o teste de _____, em que os módulos são testados individualmente, e antecede o teste de _____, em que o sistema completo é testado em um ambiente que simula o ambiente de produção.

Assinale a alternativa que completa, correta e respectivamente, as lacunas do texto acima.

- A integração – unidade – sistema
- B *release* – integração – unidade
- C unidade – caminho – *release*
- D requisito – sistema – integração
- E partições – requisito – caminho

Atividades

I – Testes do tipo “caixa branca” são realizados apenas após o software estar completamente integrado.

II – Testes do tipo “caixa preta” não são aplicáveis a software de pequeno porte.

III – Testes do tipo “caixa preta” tem a finalidade de exercitar as interfaces do software sob teste.

- (A) Apenas o item I é verdadeiro.
- (B) Apenas o item II é verdadeiro.
- (C) Apenas o item III é verdadeiro.
- (D) Apenas os itens I e II são verdadeiros.
- (E) Todos os itens são verdadeiros.

Atividades

I – Um teste de aceitação foca em cada unidade do software, isto é, seu código-fonte.

II – Um teste bem-sucedido identifica defeitos.

III – Um teste “caixa preta” é um teste estrutural, em que partes específicas de componentes são testadas.

- Apenas o item I é verdadeiro.
- Apenas o item II é verdadeiro.
- Apenas o item III é verdadeiro.
- Apenas os itens I e III são verdadeiros.
- Todos os itens são verdadeiros.

Atividades

O planejamento de testes é governado pela necessidade de selecionar alguns poucos casos de teste de um grande conjunto de possíveis casos. O exame que avalia se um grupo de entrada de dados resultou nas saídas pretendidas, levando-se em consideração a especificação do programa, é denominado teste

- a) de stress.
- b) da caixa preta.
- c) da caixa branca.
- d) da caixa cinza.
- e) de integração.

Atividade

No que se refere aos vários tipos de testes, assinale com **V** (verdadeiro) ou **F** (falso) as afirmações abaixo.

- () Teste é a forma de avaliação de qualidade mais comum porque é a que fornece melhor custo-benefício, comparada com verificação formal e tolerância a falhas.
- () As atividades relacionadas ao teste do software devem começar assim que o primeiro módulo do sistema estiver codificado.
- () A diferença entre teste caixa-preta e teste caixa-branca está apenas na forma como os testes são gerados, pois os dois métodos detectam o mesmo tipo de falha.
- () Teste de unidade só pode ser aplicado quando o sistema estiver completamente desenvolvido, pois uma unidade raramente funciona independente de outras.
- () O teste funcional ou caixa-preta consiste em definir as classes de equivalência e escolher valores-limite em cada classe como dados de teste.
- () O conceito de cobertura de arcos só se aplica ao teste estrutural, uma vez que é preciso conhecimento do código para gerar o grafo de fluxo de controle.

Casos de teste



Como obtê-los

O que é um caso de teste?

A primeira coisa que você precisa saber é que um caso de teste é um conjunto de testes, e sempre feito de forma única, respondendo a perguntas como “o que eu irei testar?”, “o que será avaliado para que o programa seja considerado bom ou não?”

Exemplo

Caso de teste para verificar se um triângulo é equilátero

Caso de teste	
ID	TCS-111
Nome	Verificar um triângulo equilátero
Ambiente	RVM-1.8.7
Ator	Aluno de matemática
Precondições	Não existir nenhum triângulo antes da realização do teste
Procedimento	<ol style="list-style-type: none">1. Abrir aplicação2. Solicitar a consulta a um tipo de triângulo3. O sistema deve solicitar as medidas do triângulo4. Informar 7 para o lado A5. Informar 7 para o lado B6. Informar 7 para o lado C7. O sistema retorna a informação de que o triângulo é equilátero
Pós-condições	Existir um triângulo equilátero

Exemplo

- **ID** – a identificação do teste.
- **Nome** – ajuda a definir, de maneira breve, o que o teste fará.
- **Ambiente** – onde, ou a partir de qual ferramenta, o teste será realizado.
Nota: RVM-1.8.7 refere-se à ferramenta *Ruby Manager*. Em linhas gerais, essa ferramenta auxilia no desenvolvimento de alguns testes. Falaremos mais sobre esse tipo de ferramenta na próxima unidade.
- **Precondições** – situações predeterminadas do que deve ou não existir antes da realização do teste para que ele seja considerado válido.
- **Procedimento** – é como o teste será feito, passo a passo. São as entradas e as saídas do sistema na ordem em que devem acontecer.
- **Pós-condições** – é a condição correta que deve existir após a realização do teste.

Caso de teste	
ID	TCS-111
Nome	Verificar um triângulo equilátero
Ambiente	RVM-1.8.7
Ator	Aluno de matemática
Precondições	Não existir nenhum triângulo antes da realização do teste
Procedimento	<ol style="list-style-type: none">1. Abrir aplicação2. Solicitar a consulta a um tipo de triângulo3. O sistema deve solicitar as medidas do triângulo4. Informar 7 para o lado A5. Informar 7 para o lado B6. Informar 7 para o lado C7. O sistema retorna a informação de que o triângulo é equilátero
Pós-condições	Existir um triângulo equilátero

Casos de testes formais

Segundo Graham et al. (2007), testes são realizáveis em vários momentos do desenvolvimento do software. Existe um tipo de caso de teste, entretanto, que é voltado exclusivamente para o momento em que o software está quase finalizado: os testes formais. Eles são realizados em última instância porque esse tipo de avaliação leva em conta o funcionamento do projeto como um todo, e não necessariamente apenas uma pequena função.

Casos de testes de aceitação formal

Quando lidamos com um caso de teste de aceitação formal, lidamos, antes de mais nada, com pessoas e simulações do ambiente real em que o programa será usado (GRAHAM et al., 2007).

Uma maneira de alcançar resultados que correspondam à realidade é documentar todas as etapas de teste, do planejamento à etapa final. Dessa forma, é possível realizar a comparação em todas as etapas de teste.

Casos de testes de aceitação formal

Como saber se meu projeto está pronto?

O primeiro passo para você definir se seu projeto está pronto para um caso de teste de aceitação é assegurar dois itens básicos (UFPE, 2006):

1. O desempenho do software está de acordo com o desempenho requerido?
2. Os artefatos do software estão fisicamente presentes?

Se a resposta for “sim” para as duas perguntas, então você está apto a identificar casos de teste de aceitação formal.

Casos de testes de aceitação formal

Definir as responsabilidades do cliente

Graham et al. (2007) dizem que esse é um tipo de caso de teste amplo feito em conjunto entre desenvolvedor e cliente. Então, é bom que, antes de mais nada, você defina o que será responsabilidade de quem.

Por exemplo, os desenvolvedores fornecerão a plataforma de instalação do software? Quem realizará essa instalação – os desenvolvedores ou o cliente? Quem monitorará os testes e organizará os casos – uma empresa terceirizada ou o próprio cliente?

Casos de testes de aceitação formal

- Liberação e retirada do software.
- Instalação de plataformas de teste de hardware e/ou software.
- Fornecimento de dados de teste.
- Fornecimento de recursos para o andamento dos casos de testes.
- Repostas e/ou resultados dos casos de testes.

Casos de testes de aceitação formal

Documentar critérios de aceitação

O que será avaliado no teste? Ora, é um tipo de caso que envolve uma visão geral. Então o quê, dentro do geral, deverá ser considerado, analisado? Afinal de contas, para que serve, de fato, o que foi projetado? Qual a razão de ser do programa?

Casos de testes de aceitação formal

Esses critérios podem ser, por exemplo (UFPE, 2006):

- Conclusão com êxito das avaliações de artefatos.
- Conclusão com êxito do treinamento do cliente.
- Conclusão com êxito da instalação do software no local.
- Medidas que identifiquem até que ponto as especificações obtidas nos testes condizem com as especificações originais.
- Medidas que identifiquem até que ponto os objetivos dos casos de negócio foram alcançados.

Casos de testes de aceitação formal

Identificar artefatos e recursos necessários e iniciar os testes

Você simplesmente deve identificar e documentar quais os artefatos do projeto o cliente – o testador – considerará para criar uma avaliação (UFPE, 2006). O que se deve esperar? Em outras palavras, você especifica para quem testará o que deve acontecer e, então, espera os resultados.

Casos de testes de aceitação formal

- equipe testadora;
- hardware da máquina;
- software;
- dados;
- documentação;
- equipamentos e equipe especializada.

Casos de testes de aceitação formal

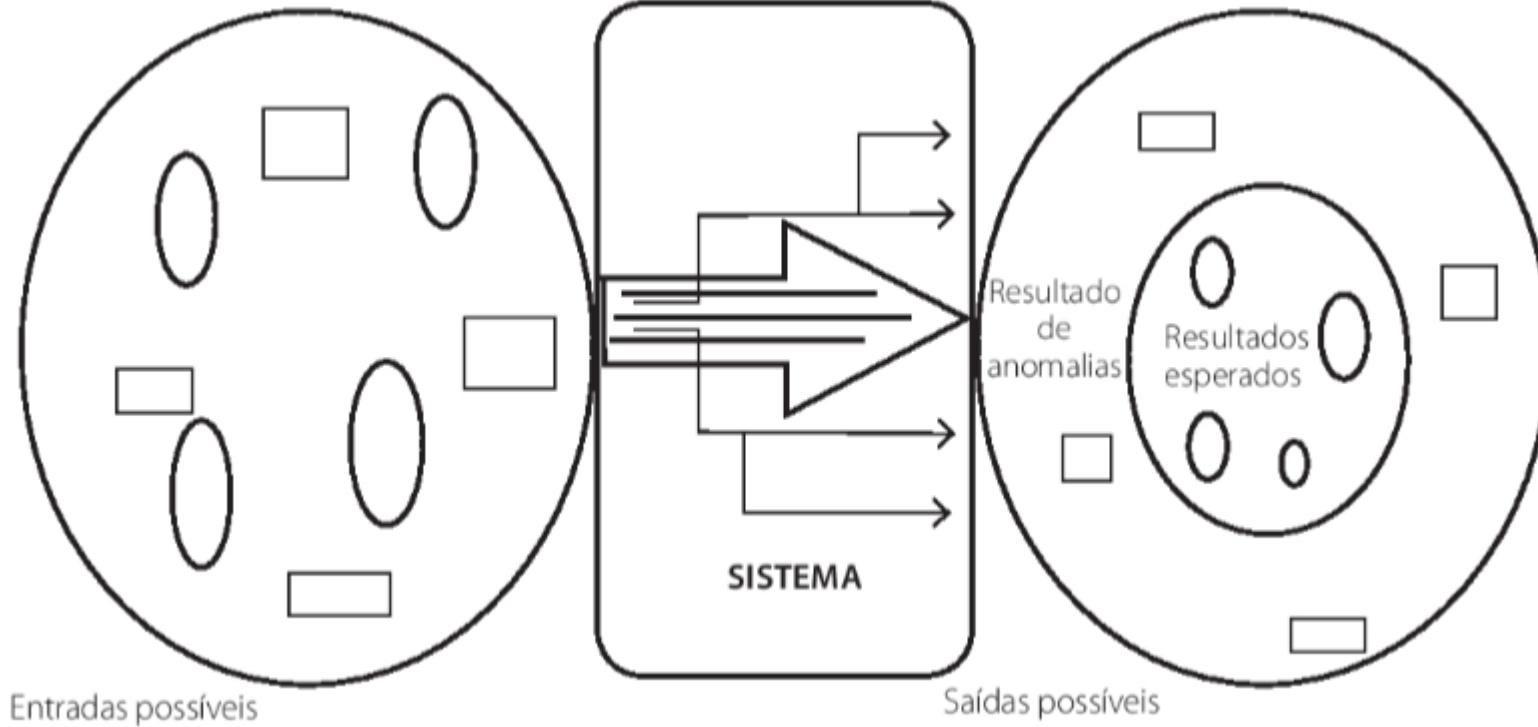
Reunião de revisão, aceitação ou rejeição do projeto

Agora é hora de verificar os resultados. Como são muitas pessoas envolvidas, de acordo com a UFPE (2006), é praxe realizar uma reunião previamente marcada para discutir o assunto.

Casos de testes de aceitação formal

Termo	Significado
Aceito	O cliente ou representante concorda com a liberação do produto de acordo com os critérios de aceitação. Tanto o produto quanto os materiais de suporte são liberados para o cliente.
Aceitação condicional	O cliente ou o representante aceita a liberação do projeto, porém somente depois que sejam determinadas e executadas ações corretivas.
Não aceito	O cliente ou o representante conclui que o projeto como um todo não corresponde àquilo que foi programado para ser feito. Para que seja aceito, outro ciclo de aceitação deve ser feito, após mudanças profundas no funcionamento de todo o sistema.

Casos de testes de aceitação formal



Casos de testes a partir de testes de desempenho

Em casos de testes para o teste de desempenho, você utilizará requisitos não funcionais (GRAHAM et al., 2007). O primeiro passo é identificar as entradas do sistema. As entradas são especificações para o funcionamento do seu software. Existem alguns fatores que você precisa levar em consideração quando for identificar essas entradas.

- Verificar se existe pelo menos um caso de teste que indique critério de desempenho (por exemplo, número de transações, número de usuários, tempo-limite de resposta, entre outros).
- Pelo menos um caso de teste que identifique o uso crítico do sistema.

Casos de testes a partir de testes de desempenho

Provavelmente, você terá mais de um caso de uso para cada requisito de desempenho. Isso é normal, uma vez que você deve levar em consideração:

- Um caso de teste abaixo do valor-limite de desempenho.
- Um caso de teste no valor exato do limite de desempenho.
- Um caso de teste acima do valor-limite de desempenho.

Exemplo

Um caso no limite exato suportado pelo software:

- 50 transações simultâneas, 1.000 usuários simultâneos com até 20 segundos de tempo de resposta.

Um caso abaixo do limite suportado pelo software:

- Qualquer valor abaixo de 50 transações simultâneas, 1.000 usuários simultâneos e tempo de resposta inferior a 20 segundos.

E um caso acima do limite suportado pelo software:

- Qualquer valor acima de 50 transações simultâneas, 1.000 usuários simultâneos, com tempo de resposta superior a 20 segundos.

Exemplo

— **Tabela 2.1** Conceitos de casos de teste para cada critério de desempenho.

Critério de desempenho	Caso de teste		
	TCI	TCII	TCIII
Transações simultâneas	< 50	= 50	> 50
Usuários simultâneos	< 1.000	= 1.000	> 1.000
Tempo máximo de resposta	< 20 s	= 20 s	> 20 s

Outros fatores importantes

- Extensão do banco de dados:
 - Quão grande é o banco de dados e como isso pode afetar a usabilidade do sistema.
- Carga de trabalho:
 - Quantos usuários finais podem usar o sistema simultaneamente.
 - Quantas transações simultâneas podem ser feitas.
- Dados do ambiente:
 - Hardware.
 - Software.

Outros fatores importantes

— **Quadro 2.3** Condições dos casos de teste de carga.

Identificador do caso de teste	Carga de trabalho	Condição	Resultado esperado
PCW1	1 Caixa eletrônico único	Transação de retirada concluída com êxito	A transação concluída (independente do autor) ocorre em < 20 segundos
PCW2	2 1.000 caixas eletrônicos simultâneos	Transação de retirada concluída com êxito	A transação concluída (independente do autor) ocorre em < 30 segundos
PCW3	3 10.000 caixas eletrônicos simultâneos	Transação de retirada concluída com êxito	A transação concluída (independente do autor) ocorre em < 50 segundos

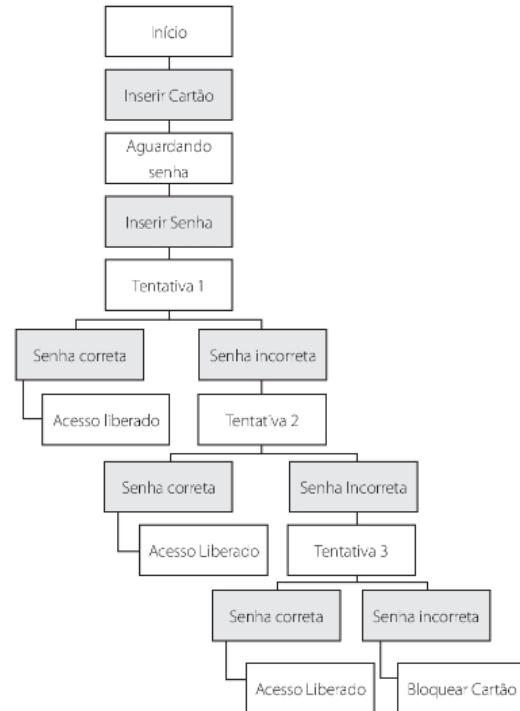
Outros fatores importantes

— **Quadro 2.4** Condições dos casos de teste de esforço.

Identificador do caso de teste	Carga de trabalho	Condição	Resultado esperado
SCW1	2 1.000 caixas eletrônicos simultâneos	Bloqueio de banco de dados – 2 caixas eletrônicos solicitando a mesma conta	Solicitações do caixa eletrônico em fila
SCW2	2 1.000 caixas eletrônicos simultâneos	A comunicação com o sistema bancário não está disponível	Transação está em fila e/ou seu tempo está esgotado
SCW3	2 1.000 caixas eletrônicos simultâneos	A comunicação com o sistema bancário termina antes do final da transação	Uma mensagem de aviso é exibida

A partir do teste de segurança

Neste caso, segundo as diretrizes da UFPE (2006), você levará muito em consideração os atores e os casos de uso do sistema.



Estados:

1. Início.
2. Aguardando senha.
3. Tentativa 1.
4. Tentativa 2.
5. Tentativa 3.
6. Acesso liberado.
7. Bloqueio de cartão.

Eventos possíveis:

1. Inserir cartão.
2. Inserir senha.
3. Senha correta.
4. Senha incorreta.

A partir do teste de configuração

Os casos de teste de configuração trabalham com os níveis de funcionamento do sistema em plataformas distintas (UFPE, 2006). Por exemplo, sistemas operacionais inferiores, diferentes hardwares, navegadores e velocidades de CPU.

Esses casos também verificam erros de DLLs, que podem entrar em conflito com versões anteriores já instaladas na máquina.

A partir do teste de configuração

Verificar se existe pelo menos um caso de teste para cada configuração crítica do sistema. Para isso, você deve simplesmente priorizar as configurações básicas de hardware e software necessárias para o funcionamento correto do programa. É recomendável que teste primeiro as configurações mais simples. Por exemplo:

- suporte a impressoras;
- conexão de longa distância;
- conexão de rede;
- drivers de hardware de servidor;
- versões de software essenciais.

A partir do teste de configuração

Você deve assegurar-se, também, de que exista pelo menos um caso de teste para cada configuração que pode denotar problemas. Veja alguns exemplos:

- hardware com desempenho abaixo do mínimo necessário;
- software com histórico de problemas de compatibilidade;
- recursos não suficientes:
 - CPU lenta;
 - memória insuficiente;
 - baixa resolução;
 - pouco espaço em disco.

Internet insuficiente:

- Clientes com conexão LAN/WAN extremamente lenta (isso se for necessário acesso à Internet).

A partir do teste de Instalação

Talvez os de definição mais simples, os casos de teste de instalação devem abranger todos os cenários possíveis da instalação de um software (UFPE, 2006). Quando dizemos “instalação”, sempre nos lembramos de programas domésticos, softwares cuja instalação realizamos em nossos computadores pessoais. Vale considerar que esses casos consideram a instalação de sistemas tanto domésticos quanto empresariais.

A partir do teste de Instalação

Mídia:

- É onde o instalador está armazenado (CD, DVD, *pen-drive*, HD externo, entre outras mídias).

Nova instalação.

Instalação completa:

- Por exemplo, quando você instala todos os programas referentes ao Pacote Office, da Microsoft, a partir da mesma mídia, você realizou uma instalação completa.

Instalação personalizada:

- Por exemplo, quando você instala o Microsoft Word e o Microsoft Excel, do Pacote Office, a partir da mesma mídia, mas opta por não instalar os demais programas do mesmo pacote, então você realizou uma instalação personalizada.

A partir do teste de Instalação

Instalações de atualização:

- Tomando ainda os programas do Pacote Office como exemplo, podemos considerar quaisquer atualizações de versões e de sistema baixadas para cada programa, que devem ser instaladas posteriormente – devemos sempre nos assegurar de que os programas continuaram funcionando corretamente após a instalação de atualizações.

Testes automatizados

Como obtê-los

Introdução



Automação é o funcionamento de uma máquina ou grupo de máquinas sem intervenção humana

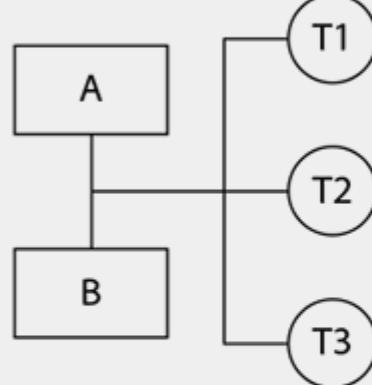
Introdução

Aplicando o termo à nossa realidade de testes se software, automação de testes seria, então, justamente o processo automático de testes, reduzindo o processo humano (CAETANO, 2008). De fato, existem diversas ferramentas e APIs que facilitam – e realizam por completo – nossa tarefa de testar um software.

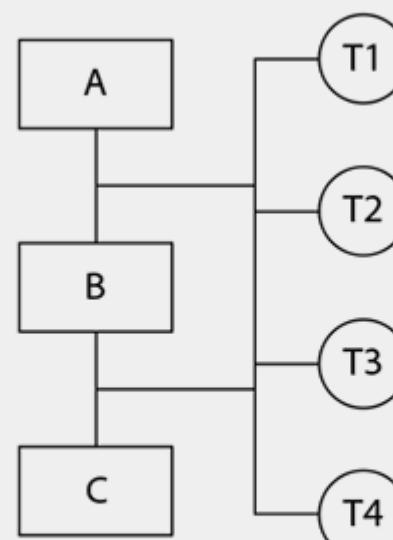
O que é API

API - *application programming interface* (interface de programação de aplicativos, em português) - é uma interface executada em um plano ao qual o usuário não tem acesso. Isto é, enquanto a interface com o usuário executa e "traduz" o programa para seu utilizador, a interface da aplicação pode fazer a ligação, por exemplo, do programa que está sendo utilizado com algum site da web - tudo isso sem que o usuário final precise se preocupar ou ter conhecimento prévio para compreensão.

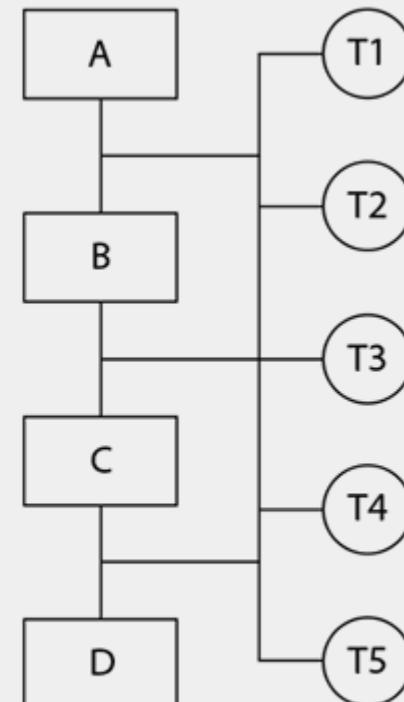
Por que automatizar?



Sequência de teste 1



Sequência de teste 2



Sequência de teste 3

Vantagens

Muitos desenvolvedores que não automatizam seus testes acabam não tendo tempo hábil para completar os testes dos softwares que desenvolvem, e os entregam sem testá-los na sua totalidade. Surgem, então, diversos bugs e falhas – o que é ruim para o desenvolvedor e para o cliente.

Outro aspecto que conta a favor da automação é o desempenho. Você já experimentou, por exemplo, ler um livro por 12 horas consecutivas? Você acha que manteria sua atenção na leitura, sem ter sono ou qualquer outro tipo de desgaste mental?

Tipos de automação

Quadro 3.1 Tipos de automação de testes.

Paradigma	Tipo de teste automatizado
Baseado na interface gráfica	Baseado na interface gráfica (capture/playback)
	Dirigido a dados (data-driven)
	Dirigido à palavra-chave (keyword-driven)
Baseado na lógica de negócio	Baseado na linha de comando (command line interface)
	Baseado em API (application programming interface)
	Test harness

Exemplo

Imagine que, para um calouro se matricular em uma universidade, ele precise preencher os dados listados a seguir (no caso de documentos, não basta apenas preencher os dados. É necessário, também, que ele entregue uma cópia para a faculdade anexar ao seu cadastro).

- nome completo;
- data de nascimento;
- sexo;
- RG;
- CPF;
- Título de Eleitor;
- Certificado de Reservista (se o estudante for do sexo masculino);
- endereço completo;
- histórico escolar do Ensino Médio.

Exemplo

Todas essas entradas fazem parte de um único sistema. Imagine, agora, que o estudante não forneça algum desses dados. O sistema concluiria a matrícula?

Naturalmente, nós temos de ter essa resposta antes de a faculdade começar a usar o software. Assim, descobrimos isso pelos testes. Ou seja, realizamos um único teste em que apenas as entradas são trocadas.

Exemplo

The screenshot shows a Windows application window titled "Test Datapool". The window has a title bar with the title and standard window controls. Below the title bar is a toolbar with several icons. The main area is a table with four columns: "Index", "Titulo::", "AbertoPor::", and "Responsavel::". The table contains six rows of data. Rows 0, 1, and 2 have visible data in all columns. Row 3 is empty. Rows 4 and 5 have data in the first two columns and long, identical strings of characters in the last two columns.

	Titulo::	AbertoPor::	Responsavel::
0	Problemas na impressora	Marcos	Pedro
1	Problemas na impressora	Marcos	
2	Problemas na impressora		Pedro
3			
4	@#\$\$%% *** &= /;;	@#\$\$%% *** &= /;;	@#\$\$%% *** &= /;;
5	@#\$\$%% *** &= /;;	@#\$\$%% *** &= /;;	@#\$\$%% *** &= /;;

Exemplo

```
import resources.CadastroHelper;  
  
import com.rational.test.ft.*;  
  
import com.rational.test.ft.object.interfaces.*;  
  
import com.rational.test.ft.script.*;  
import com.rational.test.ft.value.*;  
  
import com.rational.test.ft.vp.*;  
  
public class Cadastro extends  
CadastroHelper  
{  
  
    public void testMain(Object[] args)  
    { Descricao().click(atPoint(147,8)); Formulario().  
        inputChars(dpString("Titulo")); AbertoPor().click(atPoint(147,8)); Formulario().inputChars(dpString  
        ("AbertoPor")); Responsavel().click(atPoint(115,13)); Formulario().inputChars(dpString  
        ("Responsavel")); BotaoGravar().click(atPoint(42,8)); }  
}
```

Exemplo

- Vantagens:
 - Reutilização de scripts.
 - Diminuição da complexidade dos testes realizados.
 - Redução do tempo gasto na realização dos testes em questão.
- Desvantagens:
 - Grande dependência da estabilidade da interface gráfica. Uma falha na interface resulta em uma falha na realização do teste.
 - Ferramentas de automação de testes muito robustas. Geralmente, não é possível a realização de outros testes em conjunto, ou de apenas um grupo específico de subconjunto de dados.

Testes baseados em interface

Testes automatizados baseados na interface gráfica do usuário (como os TDDs, apresentados anteriormente) são ferramentas de automação baseadas em ferramentas de captura e execução (*capture/playback*).

Isso significa que esse tipo de automação permite capturar (*capture*) as ações do usuário enquanto ele usa o software. Em seguida, elas são traduzidas em uma linguagem de script compreendida pelo programa de automação utilizado para que possam, por fim, ser executadas posteriormente (*playback*).

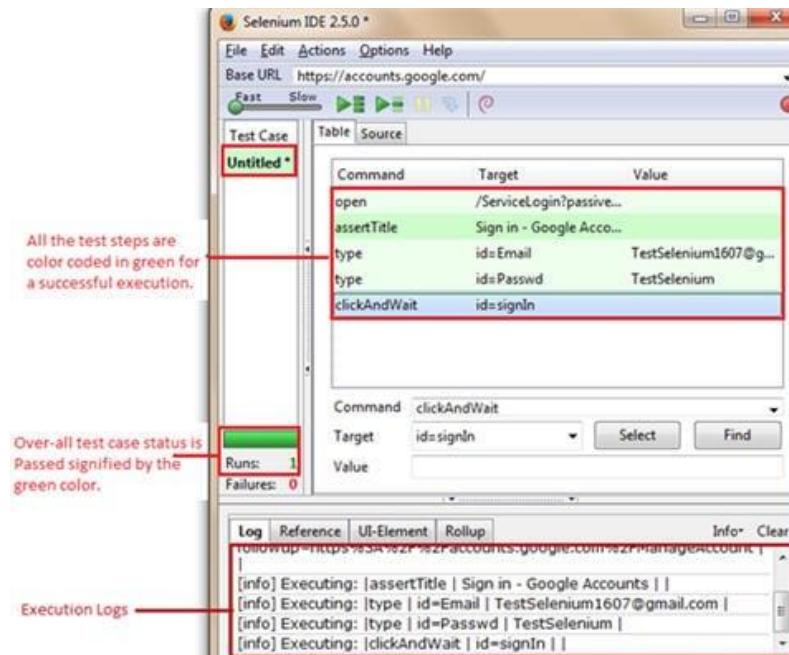
Testes baseados em interface

É uma técnica com muitos pontos positivos porque:

- Não é necessário modificar o software para realizar os testes de automação.
- Não é preciso tornar o software mais fácil de testar, uma vez que a base dessa automação é a própria interface a que o usuário terá acesso.

Selenium IDE

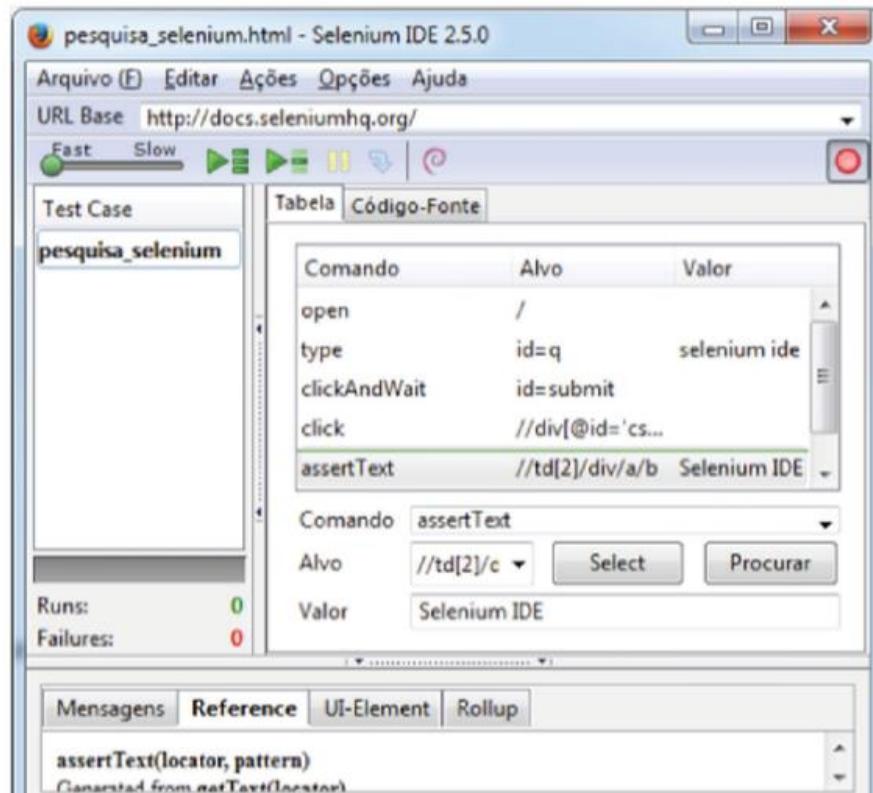
Uma das maneiras mais práticas de gravar, editar e depurar nossos testes, é utilizar o Selenium IDE (Integrated Development Environment). Ele é uma extensão que roda no navegador Firefox que permite a "gravação e reprodução de testes no ambiente real no qual o software será rodado".





- a. ***Speed control*** – controla quanto rápido o script é executado.
- b. ***Run all*** – executa vários casos de teste contidos em uma única suíte.
- c. ***Run*** – executa apenas um único caso de teste selecionado. *Nota:* se você tiver apenas um caso de teste, tanto a opção *Run* quanto a opção *Run all* terão a mesma função.
- d. ***Pause/Resume*** – pausa e reinicia a execução de casos de testes em andamento.
- e. ***Step*** – possibilita a execução de apenas alguns passos do script. É o que chamamos de *step by step*.
- f. ***Apply rollup rules*** – executa várias tarefas repetitivas através de uma única ação de comandos do Selenium IDE.
- g. ***Record*** – possibilita a gravação de todas as ações do usuário no navegador.

Selenium IDE



- Painel *test case*:

- Exibe o caso de teste atual ou o grupo de casos de testes executados.

- Duas abas:

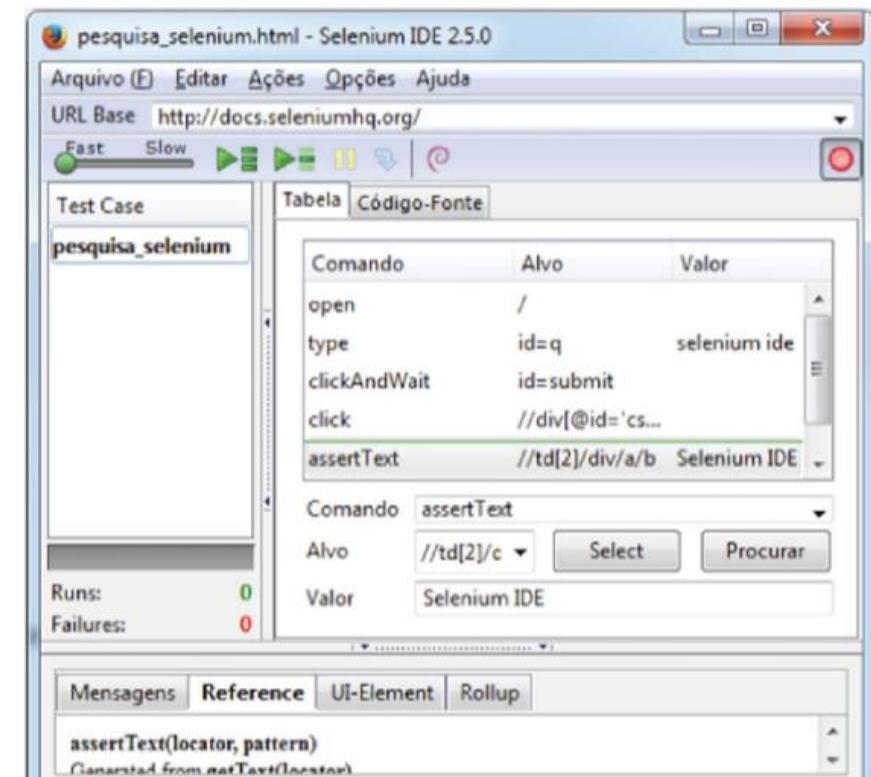
- Tabela:

- Exibe os comandos do Selenium IDE.

- Código-fonte:

- Exibe o código HTML dos comandos.

Selenium IDE



Comando:

- O comando que o Selenium IDE está executando.

Alvo:

- É, geralmente, um elemento em HTML.

Valor:

- Varia de acordo com o comando. Quando o alvo for um elemento HTML, o valor pode ser, por exemplo, o texto de comparação ou para digitação.

Selenium IDE



```
Mensagens Reference UI-Element Rollup Informações Limpar
[info] Executing: |open| / |
[info] Executing: |type| id=q selenium ide |
[info] Executing: |clickAndWait| id=submit ||
[info] Executing: |click| //div[@id='cse']/div/div/div/div[5]/div[2]
/div/div/div/table/tbody/tr/td[2]/div ||
[info] Executing: |assertText| //td[2]/div/a/b| Selenium IDE |
```

Mensagens – diz respeito a informações sobre a execução do script de testes e sobre os erros identificados.

Reference – exibe a documentação dos comandos do Selenium IDE.

UI-Element – possibilita o mapeamento dos nomes dos elementos em uma página da web.

Rollup – são comandos agrupados pelo *Rollup Rules*.

Selenium IDE

1. **Verificação (verify)** – essa opção verificará se o resultado esperado se confirma. Caso ele não se confirme, o Selenium IDE marca o passo como erro e *continua* a execução do script normalmente.
2. **Asserção (asserts)** – essa opção verificará se o resultado esperado se confirma. Caso ele não se confirme, o Selenium IDE marca o passo como erro e *encerra* a execução do script.

Selenium IDE

ID

- É a forma de localização mais comum.
- Quando o elemento HTML tem um atributo como ID, ele pode ser utilizado para localizar o elemento.
- Veja o exemplo a seguir. Neste caso, o Selenium IDE coloca na caixa de texto Alvo o valor de id=username.

```
<input type="text" id="username" />
```

Name

- Quando um elemento HTML tem um atributo *name*, o Selenium IDE pode usá-lo como forma de localização de elemento.

Exercícios

