

Aula 8: Teste de Aceitação

Apresentação

Nesta aula, vamos saber como desenvolver rotinas de teste com base no framework *Cucumber* e automação com o *Selenium WebDriver*. Vamos definir uma elaboração de testes de aceitação com o usuário final, identificar o relacionamento de requisitos a expectativas de teste, conhecer as metodologias utilizadas para testes de aceitação. Por fim, vamos conhecer a técnica de BDD - *Behavior Driven Development* (Desenvolvimento Orientado por Comportamento).

Objetivos

- Investigar o desenvolvimento de rotinas de teste com base no framework *Cucumber* e automação com o *Selenium WebDriver*;
- Elaborar testes de aceitação com o usuário final, relacionando requisitos a expectativas de teste e às metodologias utilizadas para testes de aceitação;
- Reconhecer as metodologias utilizadas para testes de aceitação e conhecer a técnica de BDD - *Behavior Driven Development* (Desenvolvimento Orientado por Comportamento).



 (Fonte: Gorodenkoff / Shutterstock).

O que é um teste de aceitação?

O teste de aceitação é aquele feito para aproximar o cliente final do resultado esperado pelo sistema. Já um teste de software é um processo sistemático que tem por objetivo identificar prováveis defeitos.

Para isso, verifica-se se o software realiza suas tarefas de forma correta, conforme os requisitos fornecidos pelas partes interessadas do sistema e também se faz o que não deveria fazer.

Alguns conceitos que devem ser entendidos:



Defeito

Contempla um ato inconsistente realizado por um indivíduo ao tentar compreender uma informação. Pode ser uma instrução ou um comando incorreto.



Erro

É um defeito encontrado em um artefato de software. Exemplo: diferença entre um valor obtido e um valor esperado.



Falha

É o comportamento do software diferente do esperado pelo usuário final.

Exemplo

Exemplo de falhas: um bug gerado por um programador pode ocasionar um erro que irá gerar uma situação de inconsistência em uma determinada funcionalidade. Lembrar a regra de 10 de Myers.

Desenvolvimento de rotinas de teste com base no framework Cucumber e automação com Selenium WebDriver

Ferramentas de teste automatizado de software

Um projeto de automação de software é um investimento alto e de longa duração. Os dirigentes das organizações têm expectativas em relação a custos e aos benefícios trazidos pela sua implementação.

Cuidados a serem tomados:



Infraestrutura



Metodologia



Ferramenta

É a disponibilidade de máquina e seus recursos. O projeto em desenvolvimento (em fase de testes) deve ser dedicado para o projeto de automação de testes.	É a existência de metodologias de desenvolvimento e testes consolidadas e usadas, que possam se integrar com a ferramenta escolhida.	Selecionar a ferramenta certa, adequada à tecnologia usada e que possa se integrar com as metodologias de desenvolvimento e teste.
-----------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------

Quais casos de testes são candidatos a automação?

- A cada nova versão de um software, será necessário realizar um novo conjunto de testes, ampliando as melhorias que foram implementadas.
- É necessário reexecutar um conjunto de casos de testes (todos ou partes), de forma a avaliar se as mudanças realizadas danificaram ou modificaram outras partes do software que já funcionava.

Para isso, precisamos ter a seguinte situação:



Os testes automatizados **não podem substituir** os testes manuais,, eles são complementares.

Atenção

Todo caso de teste é naturalmente candidato a automação, mas com toda a certeza nem todos são recomendáveis para a automação.

Automação com ferramenta Cucumber

A automação de testes é o processo da escrita de um programa para executar esses testes funcionais.

Vantagem:


- Economia de tempo e recursos durante a execução dos testes;
- Possibilidade de executar os mesmos testes repetidas vezes.

Por que automatizar os testes?

Por causa da qualidade final do produto, pois a execução de todos os testes funcionais que existem no sistema garante uma menor incidência de erros e falhas no programa.

Os testes automatizados como Cucumber e automatizados com Selenium WebDriver, são aplicados quando houver há tarefas repetitivas, aplicações com longos ciclos de vida e teste que devem feitos com maior frequência.

Cucumber

 Clique no botão acima.

Cucumber

O Cucumber é um framework BDD (Behavior-Driven Development ou Desenvolvimento Orientado a Comportamento), open source, baseado em RSpec.

O programa executa testes de aceitação automatizado escrito em estilo de BDD. Tem um analisador de linguagem simples chamado Gherkin, que permite que os comportamentos esperados sejam especificados em um idioma lógico, para que os clientes possam entender.

Três passos para tratarmos o Cucumber:

- Escrever uma história e executá-la (feature);
- Criar o arquivo de passos a partir dos snippets;
- Dar implementação aos passos.

Exemplo: O objetivo de negócio “negociação de câmbio” que contém um banco e conta bancária.

Nesse exemplo, vamos ver as funcionalidades que devemos assegurar que operem:

Primeira: consiste em possibilitar que o usuário realize as operações de câmbio utilizando sua conta, como:

1.1. Fazer saque e depósito, considerando as seguintes restrições:

- 1.1.1. Só liberar o saque se o valor deste for menor ou igual ao valor do saldo disponível na conta;
- 1.1.2. Só liberar o dep[ó]sito se o valor deste for menor ou igual ao valor do limite disponível na conta.

Segunda: possibilitar o usuário a realizar operações básicas de câmbio no banco, que são:

- 2.1. Obter o total de dinheiro no banco;
- 2.2. Obter o total de contas criadas no banco.

Para utilizar o Cucumber em nossos testes, precisamos instalar a Gem, a fim de que o programa em Ruby possa entender o contexto e ter seus comandos reconhecidos.

Como conseguir isso? Digitando o comando '**gem install cucumber**' no Cmder e pressionar [Enter].

Ao final da instalação, uso e teste, o cenário e passos são executados com sucesso.

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Saiba mais

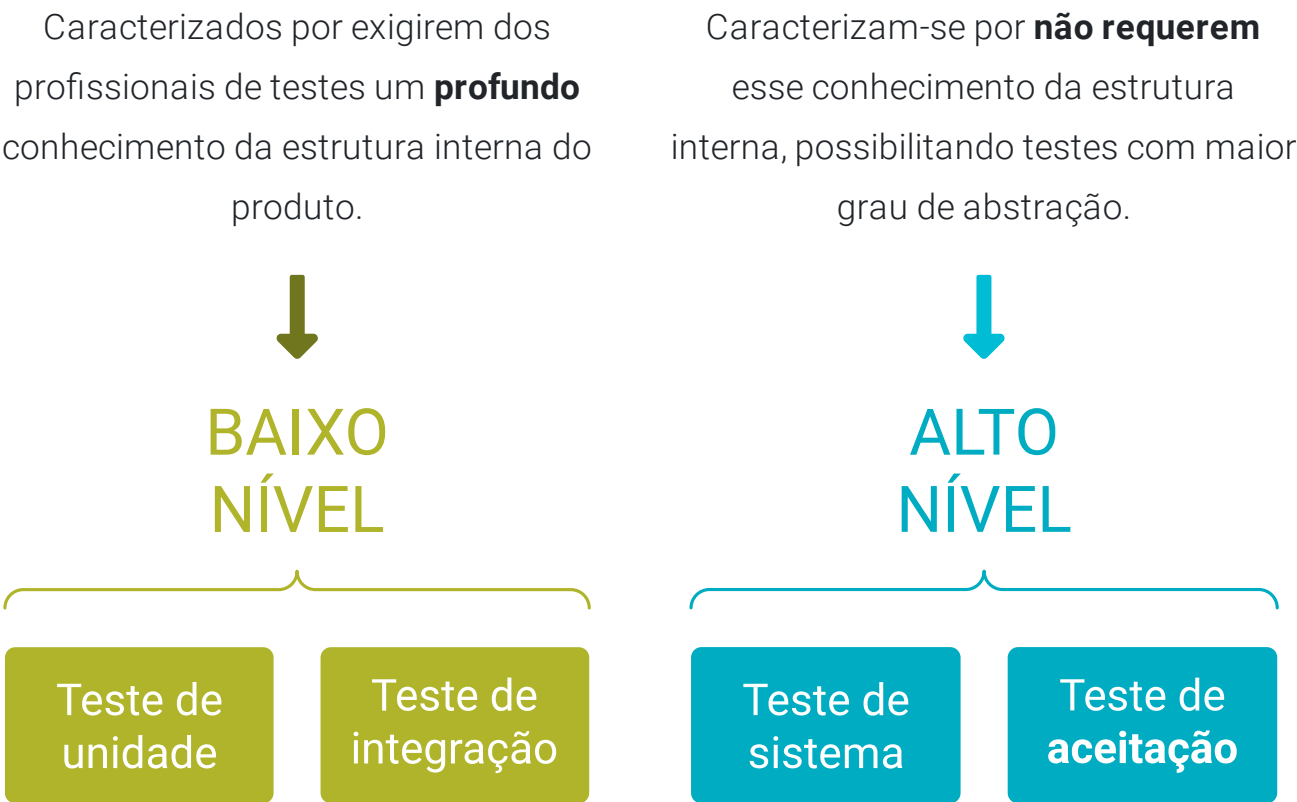
Veja mais no vídeo: [Cucumber](#)

Clique [aqui](#) para visualizar todos os passos e o resultado da instalação, teste e uso do Cucumber e automação com Selenium.

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Elaborando testes de aceitação com usuário final

Os testes de aceitação fazem parte do teste de validação de alto nível. Nesses testes, os mecanismos estão segmentados em dois níveis:



Quando terminamos a atividade de teste de integração, o software já está completamente montado. Os erros de interface foram descobertos e corrigidos, e precisamos iniciar mais alguns testes finais de software (que são os testes de validação).

Se pensarmos apenas no teste de **alto nível**, encontramos o **teste de aceitação**.

No teste de aceitação, é impossível prever como o cliente realmente usará um programa.

ALTO NÍVEL

Teste de
sistema

Teste de
aceitação

As instruções de uso podem ser mal interpretadas, e combinações anormais de dados acabam sendo usadas.

O teste de aceitação é de responsabilidade do cliente

Dependendo da abrangência dos usuários, podem ser aplicados de duas maneiras:



Software customizado para um cliente.



Software desenvolvido como produto para muitos clientes.

São utilizados testes alfa e teste beta.

Testes de aceitação devem ser definidos pelo cliente e conduzidos pelo usuário final, a fim de validar todos os requisitos do cliente.

Estratégias comuns para implementar um teste de aceitação:




Aceitação formal.



Aceitação informal ou teste alfa.



Teste beta.

 Clique nos botões para ver as informações.

É um processo que necessita ser bem gerenciado e costuma ser uma extensão do teste do sistema. Pode ser totalmente automatizado.

Importante: Não se distanciar de nenhuma forma dos casos de teste escolhidos.

Pode ser executado inteiramente pela coordenação do usuário final.

Vantagens:

- As funções e os recursos a serem testados são conhecidos;
- Os testes podem ser automatizados, o que permite o teste de regressão;
- O progresso dos testes pode ser medido e monitorado;

Os critérios de aceitabilidade são conhecidos.

Desvantagens:

- São necessários recursos e planejamento significativos;
- Os testes podem ser uma nova implementação dos testes do sistema;
- Os testes podem não revelar defeitos subjetivos no software, já que são procurados apenas os defeitos esperados.

Os procedimentos para executar o teste não são definidos com tanto rigor como no teste de aceitação formal. Nessa abordagem, o controle não é tão rigoroso como no teste formal.

Atenção: Quem realiza esse teste é o usuário final.

Se o software é desenvolvido como um produto a ser usado por vários clientes, é difícil realizar testes de aceitação com cada um. **Nesse caso, usam-se os testes alfa e beta.**

Teste alfa

- É conduzido na instalação do desenvolvedor com os usuários finais;
- É realizado com base no envolvimento de um cliente ou numa amostra de clientes;
- O software é utilizado num ambiente natural, sendo que o desenvolvedor observa o comportamento do cliente e vai registrando as anomalias detectadas durante a utilização.

Ambiente de teste e homologação

- Esse ambiente é o mais semelhante possível ao ambiente de produção;
- Fornece toda a infraestrutura necessária de hardware e software para a execução dos testes de requisitos do software e garante o ambiente ideal de simulação.

Quem utiliza este ambiente?

Quais testes normalmente são aplicados neste ambiente?

Sistema e aceitação

Esses testes são realizados através da técnica da **caixa-preta**, porque não requerem um conhecimento interno do software.

Testes de aceitação

É a última etapa de teste antes da implantação do software. Seu objetivo é verificar se o software está pronto e se pode ser utilizado pelos usuários finais executando as tarefas e funções para as quais foi criado.

O software é disponibilizado para clientes e usuários com o objetivo de validar todas as funcionalidades requisitadas no início do projeto.

Ambiente de produção

Tem que fornecer toda a infraestrutura necessária de hardware e software para que o produto desempenhe totalmente as funcionalidades para as quais foi projetado.

Quem utiliza este ambiente?

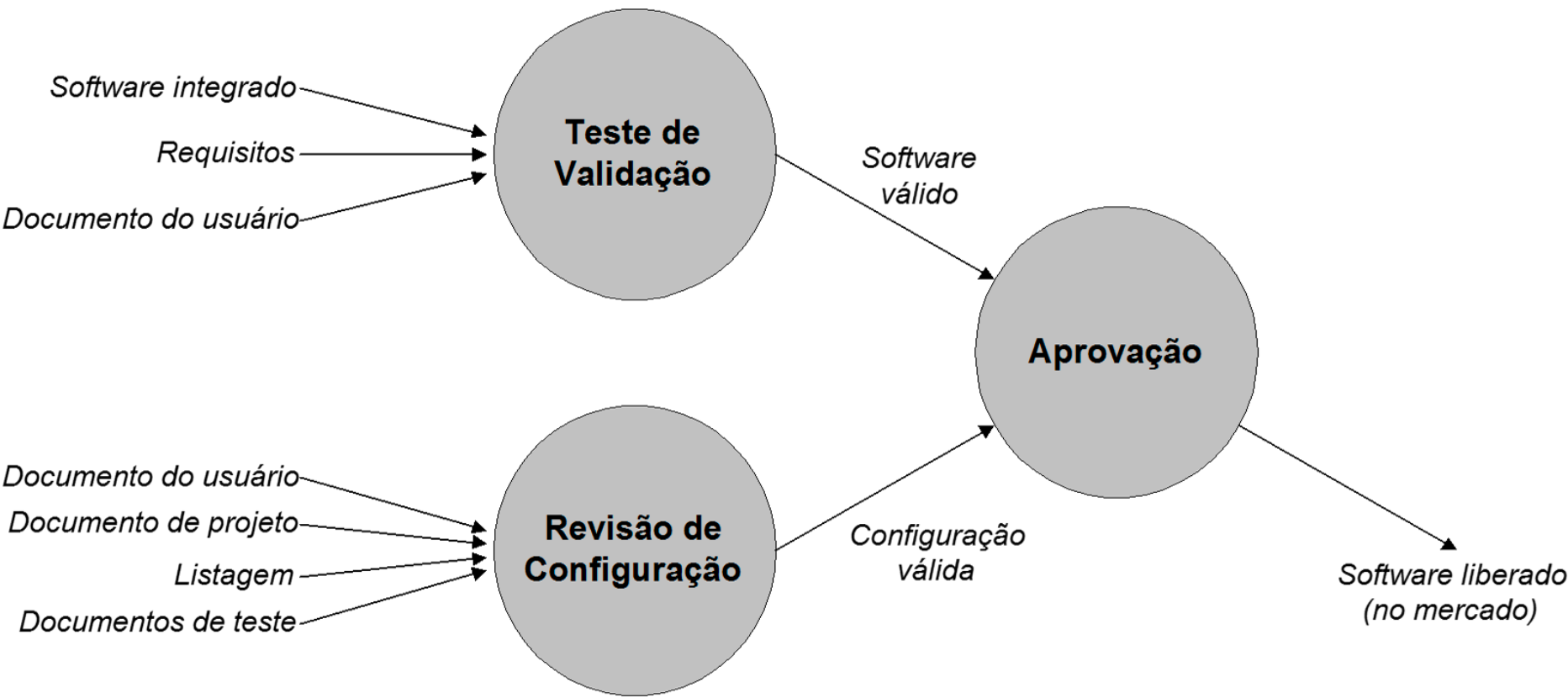
Nesse ambiente também são realizados testes de aceite, que neste caso será o beta-teste.

Apenas um grupo de usuários selecionados terá acesso a uma “nova versão” do produto (que ainda estará sob o acompanhamento da equipe de testes de software).

Quais testes normalmente são aplicados neste ambiente?

Conduzido nas instalações dos usuários finais. Algumas considerações sobre esse teste:

- O desenvolvedor não está presente;
- O cliente registra todos os problemas (reais ou imaginários) e os relata ao desenvolvedor em intervalos regulares;
- Depois que os problemas são corrigidos, o produto é liberado para toda a base de clientes;
- São conduzidos em uma ou mais instalações do cliente pelo usuário final do software.



No caso do teste beta, é praticamente impossível haver um controle da parte do desenvolvedor sobre os erros encontrados.

Relacionando requisitos a expectativas de teste

Estratégia de teste

Atenção

Conhecer e aplicar as técnicas de testes não é suficiente para garantir que bons testes sejam realizados. Também é necessário definir uma estratégia de teste para cada projeto de software.

Uma estratégia deve definir:



Quais as atividades.



Quem serão os envolvidos.



Quando e por quanto tempo
serão realizadas estas
atividades.

O que é necessário para que uma estratégia de testes tenha sucesso?

Observar alguns aspectos como: os requisitos não funcionais devem ser especificados de forma mensurável.

Não se deve dizer que o sistema deve ter um bom desempenho.

No plano de teste, devemos enunciar explicitamente os objetivos do teste.

O que deve conter um plano de teste?

Quem serão os responsáveis pelo teste.

Que tipos de testes serão realizados.

Quantos ciclos de testes são previstos.

A cobertura esperada.

O teste deve se concentrar **no uso real do produto**. Vale lembrar que o software é feito para satisfazer as necessidades dos usuários, e não dos desenvolvedores.

Como obter sucesso numa estratégia de testes?

- Buscar sempre assegurar a qualidade e a estabilidade por construção, evitando falhas e minimizando os testes necessários;
- Pensar na possibilidade de construir um software que “perceba” quando uma falha está ocorrendo e tome alguma atitude.

Exemplo


Em um sistema de saques em um banco, podem ser inseridas linhas de código para verificar que, caso o saque seja maior que o dobro do saldo da conta, suspenda o procedimento, pois seria quase impossível um resgate tão grande.

Atenção

Não se pode esperar ter sistemas com qualidade realizando apenas testes para aferir sua qualidade.

Testes baseados em requisitos

A MTS (Metodologia de Teste de Sistema) define cada um deles assim:

 Clique nos botões para ver as informações.

Requisitos de negócio



São escritos na linguagem da área de negócios e podem conter gráficos, tabelas e diagramas. Correspondem a objetivos, metas ou “desejos” da área de negócios.

Requisitos de usuário



Provêm dos requisitos de negócio e servem como base para os requisitos funcionais e não funcionais. Descreve o que o usuário estará habilitado a realizar com o sistema ou produto.

Requisitos funcionais



São base para os requisitos não funcionais e detalhados. Definem quais funcionalidades serão disponibilizadas pelo sistema ou produto.

Exemplo: Especifica as ações, processamentos, respostas, cálculos e relatórios.

Requisitos não funcionais



São a base para os requisitos detalhados. Fazem referência às qualidades, características e restrições dos requisitos funcionais e podem referir-se também aos requisitos de negócio ou de usuário.

Exemplos de tipos de requisitos não funcionais:

- Operacionais;
- Desempenho;
- Recuperação;
- Disponibilidade;
- Segurança;
- Testabilidade;
- Portabilidade.

Requisitos detalhados



Servem de base para o projeto físico e a programação. Informam como os requisitos funcionais e não funcionais serão implementados.

Requisito de teste



Informa em algumas linhas como o requisito será testado. A partir do requisito de teste, podemos especificar os **casos de testes**, que são detalhamentos do requisito de teste.

Exemplo: Continuando com o exemplo de saques na conta, para uma aplicação que permita saques entre R\$100,00 e R\$ 500,00 múltiplos de 20, um requisito de teste poderia ser “realizar saques superiores a R\$ 500,00”.

Esse requisito de teste poderia gerar um caso de teste com o valor de R\$520,00, cujo resultado esperado é “saque inválido”.

Metodologias utilizadas para testes de aceitação

Fases da Metodologia MTS

Segundo Sommerville, a MTS segue o princípio do “V”. Estabelece que, para cada fase de construção no desenvolvimento de sistemas, deve existir uma fase de teste correspondente.

Comentário

Aquilo que é definido nos levantamentos com o usuário será testado na fase de homologação, e o que é especificado para os programas será testado nos testes de unidade.


Os testes podem, e é aconselhado que sejam, planejados e projetados antes das fases de testes: uma vez que os requisitos estejam definidos, já podemos especificar os testes que devem ser realizados para validá-los.

Conectando estes dois princípios, com certeza o que é feito em uma determinada fase de construção será testado numa fase de testes correspondente.

Saiba mais

O que é definido nos levantamentos com o usuário será testado na fase de homologação, e a especificação desses testes pode ser realizada ainda na fase de levantamento.

Como a MTS define cada uma das fases de testes?

 Clique nos botões para ver as informações.

Fase de teste de unidade



Através de testes caixa preta e caixa branca, fazemos a verificação de um **componente** de software para saber se o sistema satisfaz os requisitos detalhados especificados.

Fase de teste de integração



Através de testes **caixa preta** e **caixa branca**, fazemos a verificação das **interfaces entre componentes** de um software, para saber se o sistema satisfaz aos requisitos detalhados especificados.

Fase de teste de sistema



Nesta fase, é considerado um sistema integrado de hardware e software para verificar se o sistema satisfaz os requisitos de usuário, funcionais e não funcionais especificados.



Fase executada pelos usuários, na qual são verificados os requisitos de usuário e funcionais referentes aos critérios de aceitação para permitir ao cliente determinar se aceita o sistema ou não.

Fase de teste de implantação



Nesta fase, são verificados os requisitos de negócio.

Se, com a implantação do sistema, as vendas cresceram em 30%; se a agilização de um determinado processo foi alcançada.

Verificam-se também todos os requisitos referentes à diminuição do risco de solução de continuidade do negócio devido à implantação do sistema.

Para facilitar o teste, devemos fazer um checklist dos requisitos especificados inicialmente.


Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

BDD - Behavior Driven Development (Desenvolvimento Orientado por Comportamento)

Metodologias de desenvolvimento

Vejamos algumas: o **TDD (*Test Driven Development*)** e o **BDD (*Behaviour Driven Development*)**.

 TDD e BDD

 Clique no botão acima.

TDD e BDD

O **TDD** é uma abordagem de desenvolvimento de software em que os testes direcionam a implementação do software. O desenvolvimento deve ser guiado a testes, e um teste unitário deve ser escrito antes que uma funcionalidade do sistema o seja.

O objetivo desta metodologia é fazer com que o teste passe com sucesso, significando que assim a funcionalidade está pronta e conta com garantia de qualidade. Favorece o design de software e expressa o comportamento do código.

O **BDD** tem uma abordagem no estilo TDD. A documentação é executável, melhora o desempenho dos times e pode ser utilizada por todos os envolvidos no projeto.

No **BDD**, o desenvolvimento deve ser guiado aos comportamentos que o sistema deve apresentar. Desta forma, um comportamento (requisito/ especificação) é priorizado em relação ao teste unitário, o que não exclui a execução do fluxo do TDD neste processo.

O **Behavior Driven Development (BDD)** não é uma metodologia de desenvolvimento de software, nem tampouco um substituto para as metodologias XP, Scrum, Kanban, OpenUP, RUP ou qualquer metodologia que o mercado atualmente oferece.

Na realidade, **o BDD incorpora e melhora as ideias de muitas dessas metodologias**, ajudando, melhorando e tornando a vida da equipe de software mais fácil.

Princípios do BDD

- **O bastante é o bastante**

Trabalhar para atingir as expectativas das partes interessadas, mas evitar fazer mais do que o necessário.

- **Entregar valor às partes interessadas**

Se estivermos fazendo algo que não entrega valor ou não aumenta a sua habilidade de entrega de valor, é melhor parar e fazer outra coisa.

- **Tudo é sobre o comportamento**

Assim como podemos descrever o comportamento a partir da perspectiva das partes interessadas, também podemos descrever o comportamento de um código a partir de outro código que o utiliza.

Como é o ciclo do BDD?

1. Partes interessadas discutem os requisitos

- Os requisitos são organizados em funcionalidades;
- Podem ser quebrados em estórias;
- Fazem sentido para as partes interessadas.

2. Partes interessadas, analista e testador determinam o escopo das estórias (narrativo)

- O analista pensa na funcionalidade de uma forma geral;
- O testador pensa em cenários concretos, com valores de entrada e saída.

3. Os cenários prioritários são identificados

- As partes interessadas especificam exatamente o que quer que seja entregue;
- Os desenvolvedores implantam o bastante para satisfazer os cenários e nada mais.

4. Por último, os desenvolvedores

- Automatizam os cenários que orientam o desenvolvimento;

- Descrevem o comportamento esperado;
- Implantam os comportamentos.

Funcionamento do BDD



Exemplo:

Vamos explicar usando o seguinte **exemplo**: uma **equipe praticante de BDD** decide implementar uma nova funcionalidade. Para isso, eles trabalham em conjunto com os usuários e outras partes interessadas para definir as histórias e cenários do que os usuários esperam dessa funcionalidade.

Os usuários ajudam a definir um conjunto de exemplos concretos que ilustram resultados que a nova funcionalidade deve fornecer.

Esses exemplos devem ser criados utilizando um vocabulário simples para ser facilmente compreendidos pelos usuários finais e membros da equipe de desenvolvimento de software.

São expressos usando: **cenário** (*scenario*), **dado** (*given*), **quando** (*when*) e **então** (*then*).

Com base no BDD, a equipe identificou e especificou o objetivo de negócio, definido com um exemplo concreto.

Como fica no nosso exemplo de saldo de conta:

- **Cenário**: transferir dinheiro para uma conta poupança;
- **Dado** que eu tenho uma conta corrente com 2.000,00;
- **E** que eu tenho uma conta de poupança com 3.000,00;
- **Quando** eu transferir 1.000,00 a partir de minha conta corrente para a minha conta poupança;
- **Então** eu deveria ter 1.000,00 em minha conta corrente;
- **E** eu deveria ter 4.000,00 em minha conta poupança.

Depois de especificada a nova funcionalidade, sempre que possível estes exemplos concretos serão automatizados sob a forma de especificações executáveis, que tanto **validam o software** quanto **fornece uma documentação atualizada**, tanto técnica quanto funcional.

Atividade

1. Nos testes de validação, os mecanismos de testes estão segmentados em dois níveis distintos de testes de baixo nível e de alto nível. Assinale a ÚNICA opção que apresenta os 2 testes de alto nível:

- a) Teste de sistema e teste de aceitação.
 - b) Teste da caixa branca e teste da caixa preta.
 - c) Teste de integração e teste de unidade.
 - d) Teste de regressão e teste fumaça.
 - e) Teste de sistema e teste de integração.
-

2. O termo automação de teste de software significa a utilização:

- a) De um software que imita a interação com a aplicação no que se refere ao teste tal qual um ser humano faria.
 - b) Do desenvolvimento de scripts de testes para simular a massa de teste a ser utilizada.
 - c) De casos de testes automatizados que imitam a interação com a aplicação.
 - d) De uma metodologia de teste para simular os sistema em produção.
 - e) De um ambiente de teste, de ferramentas e de uma massa de teste.
-

3. Para a implementação de um projeto de automatização de testes, precisamos de:

- a) Recurso, infraestrutura, ferramenta e metodologia.
 - b) Ferramenta, equipe de teste, processo de teste e caso de teste.
 - c) Scripts, software de teste, testador e um sistema para testar.
 - d) Ferramenta, equipe de teste, sistema para testar e hardware.
 - e) Hardware, software, script e os dados para teste.
-

Referências

4. Nos testes de validação, os mecanismos de testes estão segmentados em dois níveis: testes de baixo nível e de alto nível. BARTIÉ, Alexandre. **Garantia de qualidade de software**. 1. ed. Rio de Janeiro: Campus, 2002. Descreva quais os testes que são considerados de alto nível e quando são aplicados.

CAETANO, C. **Automação e gerenciamento de testes**: aumentando a produtividade com as principais soluções open source e gratuitas. Disponível em: <http://www.linhadecodigo.com.br/artigo/1566/automacao-e-gerenciamento-de-testes-aumentando-a-produtividade-com-as-principais-solucoes-open-source-e-gratuitas-2a-edicao.aspx>. Acesso em: 06 jun. 2019.

COCKBURN, A. **Escrevendo casos de uso eficazes**. Porto Alegre: Bookman, 2005.

CUCUMBER. **Executable Specifications**. Disponível em: <https://cucumber.io/docs>. Acesso em: 1 jun. 2019.

GITHUB. **Join GitHub today**. Disponível em: <https://github.com/thiagomarquessp/capybaraforall>. Acesso em: 1 jun. 2019.

LARMAN, C. **Utilizando UML e padrões**: uma introdução à análise e ao projeto orientado a objetos e ao desenvolvimento iterativo. 3. ed. Porto Alegre: Bookman, 2007.

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

RUBY. **Documentação**. Disponível em: <https://www.ruby-lang.org/pt/documentation/>. Acesso em: 1 jun. 2019.

SELENIUM. ***Selenium Documentation***. Disponível em: <//docs.seleniumhq.org/docs/>. Acesso em: 1 jun. 2019.

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

Próxima aula

- Mantis BugTracker;
- Configuração e utilização;
- Cadastrando defeitos.

Explore mais

Assista aos vídeos:

- [Tutorial Testlink - Disciplina Teste e Qualidade de Software](#);
- [Criar projeto, caso de teste e utilizar ferramenta TestLink](#).

Leia o textos

- [TestLink](#)