Curso: Testes Automatizados

Micro Learning 3 – Test Driven Development (Teoria)

Olá! Seja bem-vindo(a)!

Durante esse curso, você aprendeu que a automação de testes nada mais é do que o emprego de ferramentas de controle e execução de scripts de teste de software, ou seja, você aplica técnicas e estratégias com o objetivo principal de escrever um código que poderá testar o programa por você.

O objetivo dessa aula é apresentar para você o conceito de uma das técnicas de desenvolvimento de software, o TDD (do inglês, Test Driven Development). Ficou curioso(a)? Vamos lá!

A partir da metodologia ágil de desenvolvimento, diversos frameworks e ferramentas surgiram, com o intuito de facilitar a implementação dos testes ágeis, com relação à base do processo de testes, que são os testes unitários (fase onde cada unidade do sistema é testada individualmente).

O TDD, portanto, aparece como uma abordagem de desenvolvimento orientada a testes, que tem seu principal objetivo na concepção do teste, que especifica e valida o que o código fará. Em termos simples, os casos de teste são criados antes do código da funcionalidade ser gerado.A ideia do TDD é tornar o código mais claro, simples e sem erros e baseia-se em pequenos ciclos de repetições, onde para cada funcionalidade do sistema um teste é criado antes. Este novo teste criado inicialmente falha, já que você ainda não tem a implementação da funcionalidade em questão e, em seguida, você a implementará para fazer com que este teste passe.

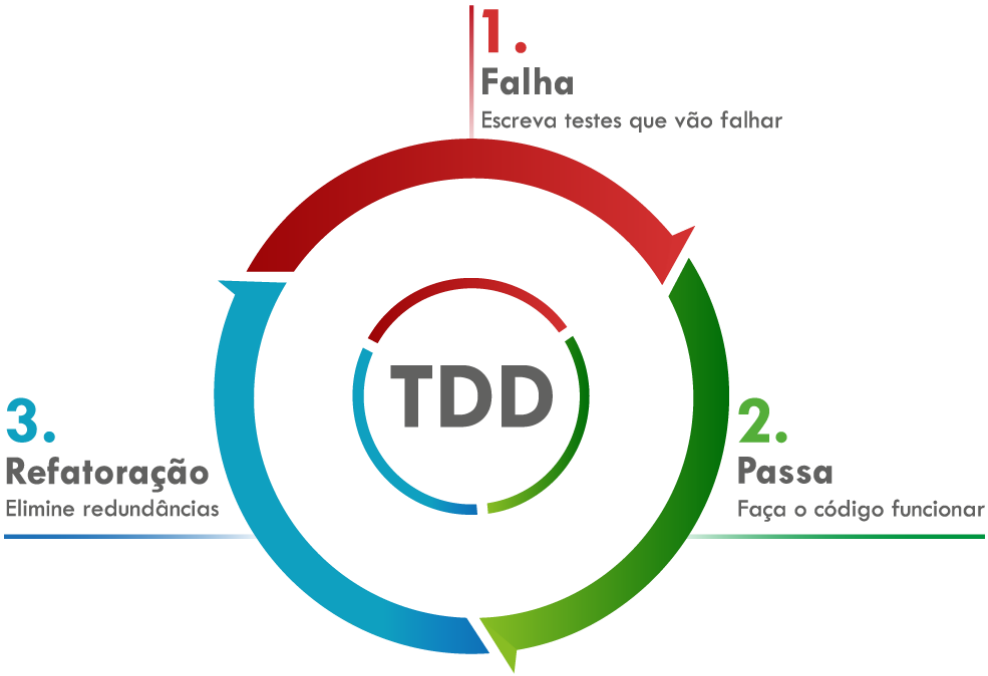
Vale ressaltar que o TDD também instrui os desenvolvedores a escrever um novo código, somente se um teste automatizado falhar. Você consegue perceber como isso evita a duplicação de código? Então, os testes passam a ser condições de requisitos que você precisa testar, para cumpri-los. A abordagem TDD é principalmente uma técnica de especificação, que garante que o seu código-fonte seja exaustivamente testado no nível confirmatório.

Nos testes tradicionais, um teste bem-sucedido encontra um ou mais defeitos. É o mesmo que acontece no TDD. Quando um teste falha, você fez progressos porque sabe que precisa resolver um problema, já que o sistema precisa atender aos requisitos definidos para ele. Isso ajuda a aumentar sua confiança no software.

No TDD, o foco está no código de produção, que verifica se o teste funcionará corretamente. Nele, você realiza 100% de teste de cobertura, pois cada linha de código é testada, diferentemente dos testes tradicionais, onde o foco está no design de casos de teste, ou seja, o teste mostrará a execução adequada ou inadequada do sistema, para atender às especificações.

A combinação dos testes tradicionais e do TDD leva à ideia da importância de testar o sistema em vez de obter a perfeição desse sistema. Na modelagem ágil, você deve "testar com uma finalidade". Você deve saber por que está testando algo e qual o nível que precisa ser testado.

A seguir, a imagem de um ciclo, no sentido horário, ilustra o Test Driven Development (desenvolvimento guiado por testes). O ciclo é formado por três cores distintas, interligadas, representando as três fases do processo. A primeira fase é representada pelo trecho vermelho, ao lado, há o título: “Falha” e abaixo dele o texto: “Escreva testes que vão falhar”. A segunda fase é representada pelo trecho verde, ao lado, há o título: “Passa” e abaixo dele o texto: “Faça o código funcionar”. A terceira fase do ciclo é representada pelo trecho azul, ao lado, há o título: “Refatoração” e abaixo dele o texto: “Elimine redundâncias”. No centro do ciclo, há um ciclo com as mesmas cores e está preenchido pela sigla “TDD”.



Test Driven Development

Para que o conceito fique mais claro na sua mente, imagine esta abordagem como um círculo, que está sempre girando, até que não existam mais novas funcionalidades ou mudanças na codificação. Neste ciclo, o ponto inicial indica que quando o TDD é aplicado, sempre será necessário escrever os testes primeiro. Ao executá-los, estes testes falharão, visto que ainda não temos o código da funcionalidade implementado. A perspectiva das falhas permite que o desenvolvedor pense melhor nas técnicas e padrões de desenvolvimento, construindo um código mais descomplicado e de acordo com os critérios do cliente, ou seja, o desenvolvedor desenvolve a funcionalidade de maneira que o teste passe, conforme indica o segundo ponto no ciclo. O terceiro e último ponto sugere que, quando possível, os testes e, consequentemente, o código sejam refatorados, de maneira a eliminar prováveis redundâncias.

Tendo em mente essa abordagem, imagine por exemplo, se você tivesse que desenvolver um cadastro de alunos para uma academia, como seria esta implementação com TDD? Primeiro, você precisa pensar no teste. Para que o cadastro de um aluno seja feito corretamente, você deve garantir que os campos obrigatórios sejam preenchidos, com suas respectivas particularidades. Este teste facilitará o desenvolvimento do código para cadastro dos alunos (classes e objetos).

Não escreva o código funcional primeiro e depois o código de teste como uma reflexão tardia. É necessário que você faça isso em etapas muito pequenas - um teste e um pequeno pedaço do código funcional correspondente, por vez. Um programador que adota uma abordagem TDD se recusa a escrever uma nova função até que primeiro ocorra um teste que falha, porque essa função ainda não está criada. Parece simples em princípio, mas quando você está aprendendo a adotar e colocar em prática, exige muita disciplina.

Aproveitando o contexto de disciplina e observação, é importante também você conhecer como essas características foram fundamentais para que o TDD tivesse origem, em 2003, através de pesquisas do americano **Kent Beck**.O engenheiro de software criou e popularizou o TDD, definindo duas regras simples para a abordagem: A primeira regra estabelece que você deve escrever um novo código comercial apenas quando o teste automatizado falhar. A segunda regra determina que você deve eliminar qualquer duplicação que encontrar.Beck explica como essas duas regras simples geram comportamentos individuais e de grupo complexos, posto que:

- você desenvolve organicamente, com o código em execução fornecendo feedbacks entre as decisões;
- você escreve seus próprios testes, porque não pode esperar para que alguém os escreva para você;
- seu ambiente de desenvolvimento deve fornecer resposta rápida a pequenas alterações e seus projetos devem consistir em componentes altamente coesos e pouco acoplados.

Ficou claro para você? Nesta aula, você entendeu o que é TDD, em sua teoria. É importante ressaltar que para uma melhor fixação do conteúdo, é essencial que você coloque em prática o que aprendeu na aula. Fique à vontade para explorar as possibilidades desta abordagem e proceder da melhor forma que se adeque às suas necessidades.

Por hoje é só!

Bons estudos!

Referências:

Ciclo do TDD. Fundamentos do Desenvolvimento orientado a Testes. Disponível em: <https:// devmedia.com.br>. Acesso em: 22 mai. 2020.

GUERNSEY, M. Test-Driven Database development: Unlocking Agility. Addison-Wesley Professional, 2013 – First Edition.

BECK, K. et. Test-Driven Development by Example. Addison-Wesley Professional, 2002 – First Edition.