01

Jasmine Aula 4

Transcrição

[00:00] Agora que você já aprendeu a escrever um monte de testes, já aprendeu um pouco mais sobre qualidade de códigos de testes e tudo mais, já está mais maduro nisso, está na hora de faze uma pequena discussão filosófica, que é a seguinte: você pode ver que os códigos que eu estou dando para você testar, eles são códigos que estão bem escritos, isolados, assim fica fácil, você invoca um método e verifica as saídas. Mas, eu tenho certeza que você já viu ou até já escreveu códigos diferentes do que estamos acostumados. Quer ver só? Dá uma olhada nesse trecho de código.

[00:40] É um código bem comum em JavaScript, eu tenho duas variáveis, peso e altura, que estou pegando os dados dela via de JQuery, mas pouco importa, imagina que está vindo de um put do meu HTML. Embaixo eu faço uma regra de negócio qualquer, estou calculando o IMC e, no final, eu exibo em um div, o IMC é tanto. Então, generaliza esse código. É muito comum ver código de JavaScript que mistura um pouquinho de manipulação de interface, HTML, com regra de negócio, com mais manipulação de interface, com mais regras de negócio. E a pergunta que fica é como testar esse tipo de código.

[01:16] Porque o código que vemos por aí é o código que estávamos acostumados a escrever até levar JavaScript a sério. E aí que vem a charada, a discussão filosófica, que é bem difícil testar esse tipo de código. E não fica triste porque não é difícil testar esse código porque ele está escrito em JavaScript, seria difícil testar esse código em qualquer linguagem. Quanto mais acoplado o código está com um monte de coisas, mais complicado é testar.

[01:45] Então, se você está programando uma linguagem server-side como Java, C#, Ruby, você já percebeu que se você escreve um código, eu estou aqui na minha cabeça com exemplo das JSP, ou pensem naqueles arquivos as ASP, PHP antigos, onde você lida com interfaces, escreve HTML e acesso a um banco de dados, tem uma regra de negócio, consome um webservice, tem mais HTML, acessa mais o banco. O código é um macarrão, faz de tudo um pouco.

[02:12] Como você escreve um teste de unidade para aquilo? Não consegue. É muito, muito difícil. E no mundo server-side as pessoas já aprenderam que precisa separar as responsabilidades. É por isso que todo mundo que programa hoje para a web pensa um pouquinho na MVC, aquele padrão Model View Controller. Aqui no curso, se você não entende ele, não precisa entender tão a fundo, apesar de eu recomendar, faça os nossos cursos de web.

[02:39] O povo lá do Java, do C#, do Rails aprendeu que tem que separar as responsabilidades. Regra de negócio de um lado, manipulação de interface de outro, infraestrutura de outro e assim por diante. Porque a hora que você separa as coisas, você vai acabar com um código parecido com o que temos agora, que é a classe paciente, que é a classe consulta, bem isolada, bem pequena, fácil de ser testado.

[03:02] No mundo JavaScript, em particular, o que vem ficando bastante famoso são aqueles frameworks, como o AngularJS, que usam o MVVM, que no fim das contas é a mesma coisa, é tentar separar códigos de interface de regras de negócio. Se você nunca viu a AngularJS, eu até recomendo você fortemente aprender AngularJS. Uma discussão que eu não fiz até agora, mas que você já deve ter reparado, ainda mais com esse nosso começo de discussão, é que não adianta só aprender a testar, você precisa também escrever código que favoreça a testabilidade. Um código que facilite você a escrever um teste para ele. Tem que pensar nisso hoje em dia, não tem como fugir.

[03:42] Agora vou mostrar para vocês, não bem como funciona o AngularJS, mas de maneira genérica qual é a ideia. Primeira coisa, é separar tudo que manipula interface. Interface de um lado, regras de negócio de outro. Em regras de negócio nós já temos a nossa classe paciente, nossa classe consulta, está tudo perfeito. Agora dá uma olhada nessa tela de pacientes, é uma classe cuja responsabilidade é conhecer o HTML, saber manipular aquela página. No caso, a tela de pacientes.

[04:09] Tem uma função, pega o paciente, que instancia o objeto paciente, pegando os dados da interface. Então, tô pegando o nome, idade, peso e altura, que são input text do HTML, e ele me devolve um objeto paciente, é um objeto rico que eu sei manipular. Veja só o método exibe, ele recebe uma variável, o IMC e ele pega esse parâmetro e exibe no div, com a mensagem bonitinha, "o IMC é tanto".

[04:35] Separei aquela parte de manipular a interface em um arquivo. E tenho também o código que vai fazer o bind, ele vai usar essa tela de pacientes, vai usar a entidade paciente e vai fazer as coisas acontecerem. Dá uma olhada nesse código. Instancia a tela de pacientes, pega o paciente na interface, invoca IMC e exibe o resultado de novo. Estou alternando entre invocar métodos da minha entidade paciente e métodos da tela de pacientes. Separei as responsabilidades.

[05:04] Agora escrever teste para o paciente você já sabe, não tem mais segredo para isso. Agora, a próxima pergunta que também merece uma resposta filosófica é: tem como testar os dois códigos que acabei de mostrar? Como testar o tela de pacientes e como testar esse código que faz o binding das coisas? Se você já programou server-side, você percebe que esse código parece muito o código de um controller do seu asp.net MVC, ou do seu spring MVC, ou do seu Rails.

[05:34] É um código que liga a parte de interface com o modelo. E a minha resposta particular para isso é que você não consegue testar isso com o teste de unidade. Por quê? Porque olha o que esse código faz, ele basicamente repassa invocações de métodos para as outras classes. Ele chama métodos do paciente, ele chama métodos do tela de pacientes. Escrever um teste para isso vai ser perda de tempo. "Ah, mas esse código pode ter um erro, ou o código da tela de pacientes pode ter um erro", pode, mas como eu garanto isso?

[06:04] Escrevendo um outro tipo de teste, que é o que nós chamamos de testes de sistema. Até agora tudo o que eu mostrei para vocês é o que nós chamamos de testes de unidade, que é o teste que está testando aquela classe. No caso, é o paciente, a consulta isolada do resto. Eu tenho um outro nível de teste que nós chamamos de teste de sistema, que também é conhecido como teste de caixa preta, que é aquele teste onde a aplicação está fechada e o usuário abre um browser e faz o teste como se fosse o mundo real.

[06:36] Só que, óbvio, eu vou automatizar isso, então é a máquina que vai abrir o browser e vai clicar nas coisas, vai dar entrada, vai digitar, vai verificar o que aparecer no HTML. Esse tipo de código eu testo, então, com esse tipo de teste. Teste de sistema. Ferramentas para isso? O Selenium no mundo Java, no mundo DotNet, muito popular. O Capybara no mundo Ruby, que por baixo dos panos também usa Selenium.

[07:02] Temos curso de Selenium aqui nessa formação de testes, então faça também que vai te enriquecer bastante, mas a ideia é que esse tipo de código eu prefiro testar com um teste de sistema. Eu me sinto mais produtivo, eu acho que o feedback é maior. Então, a regra é: você tem regras de negócio e tudo mais, códigos como eu mostrei para vocês, que tem for, if, contas, algoritmos, que pertencem a um negócio específico, isola esse código ao máximo que você puder para escrever um teste de unidades usando Jasmine. Vai ficar tudo lindo e maravilhoso.

[07:38] E o código que faz o binding com a interface, escreve um teste de sistema com o Selenium pra testar e para garantir que ele funcione. Então, no fim das contas a discussão é que você tem que parar, pensar e começar a escrever um código que facilite a testabilidade. É isolar as responsabilidades. Aquela maneira antiga de programar, seja em JavaScript, ou seja em ASP, PHP, onde você tem código macarrônico que faz tudo, é legal, o mundo usa bastante, mas isso dificulta a testabilidade.

[08:10] Mais legal então é separar as coisas. É pensar lá no AngularJS, é pensar em separar as responsabilidades, é ter o código mais coeso, pra conseguir escrever testes como esse. Uma frase que eu sempre falo para todo mundo, é que escrever teste tem que ser fácil, porque o teste nada mais é do que um código que dá um new numa classe, invoca um método e verifica a saída desse método. Se está difícil é porque seu código não está modular suficiente, isolado o suficiente para que isso seja fácil. E tem que ser.

[08:46] Então, para resumir o que eu falei nesse capítulo para vocês, eu discuti aqui código macarrônico é muito difícil de ser testado com teste de unidades, usando Jasmine. A ideia é isolar as responsabilidades, para você conseguir testar o coração do sistema, que são aquelas regras de negócio, usando tudo o que eu mostrei até agora. E aí o código que faz o binding com o resto infraestrutura que você tem, você escreve um teste em outro nível, que é o teste de sistema, que vamos discutir isso em um outro curso.

[09:14] Programe pensando em testabilidade e entenda esses diferentes níveis de testes. Unidade, sistema. Então, esse é o capítulo quatro, era essa a discussão que eu queria trazer para você, porque isso devia estar na sua cabeça, porque provavelmente o tipo de código que eu mostrei aqui é um código que você vê com frequência. E você precisa pensar primeiro em programar para facilitar a testabilidade pra depois testar. Esse foi o capítulo 4, obrigado e nos vemos no próximo capítulo.