Treinamento TDD

Victor Correia de Moura 15/0150792

Lucas Henrique Araujo Malta 13/0122050

Vitor Falcão H Costa 15/0151624

Romeu Carvalho Antunes 15/0147554

**Introdução**

Para realizar o treinamento de TDD (Test Driven Development), definimos uma metodologia de dinâmica em grupo e a utilizamos em uma atividade. A metodologia escolhida foi uma prática de pareamento já que, assim, podemos aplicar o treinamento em grupos grandes de pessoas. A ideia principal é dividir o TDD em duas partes, para que cada elemento da dupla realize uma etapa.

Em qualquer aplicação da metodologia TDD existem três partes básicas: a declaração do problema, a transformação dele em testes e por fim a codificação para que os testes sejam aceitos. Sendo assim no treinamento é apresentado o problema, um dos elementos da dupla fará os testes enquanto o outro faz o código. Por fim a ideia é que seja percebido que não é tão trivial quanto parece e vem a ser algo que se aperfeiçoa com prática.

Além da metodologia, devemos nos atentar a qual linguagem de programação utilizaremos. Tal fator é variável dependendo do grupo que está sendo treinado. Aconselha-se utilizar alguma linguagem que possua um framework de testes e que não exija longas configurações, logo o grupo possa se familiarizar rapidamente. No exemplo dado, utilizamos a linguagem Python com a biblioteca *unittest* como ferramenta de testes unitários, pois a mesma consegue atender todos esses objetivos.

Por conta da apresentação teórica, que foi feita em slides, precisamos de um projetor ou televisor no ambiente de treinamento. Além disso, precisamos de um número de computadores maior ou igual à metade do número de participantes. Caso o ambiente de desenvolvimento necessite de maiores configurações, disponibilize-as antes do treinamento para os participantes que desejarem utilizar máquinas próprias e realize os procedimentos necessários nos computadores locais. No exemplo que será trabalhado, foi utilizada uma linguagem que possui suporte nativo na maioria das máquinas dos participantes. Caso não houvesse suporte, seria possível desenvolver em IDEs online, como o IDEOne.

O treinamento é dividido em três etapas:

1 - Apresentação teórica sobre o Test Driven Development, tendo como ênfase suas vantagens em uma produção.

2 - Breve introdução às ferramentas que serão utilizadas no treinamento.

3 - Trabalho prático com aplicação de TDD.

**Desenvolvimento das etapas**

**Primeira etapa**

Composta pela apresentação teórica sobre TDD, nela ressaltamos as vantagens da prática:

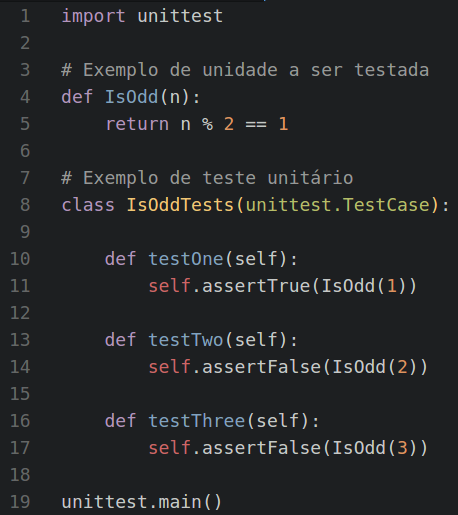
* Não há influência de um código pronto ao codificar os testes, logo você consegue ter testes melhores e consequentemente melhor qualidade de código.
* Com o TDD o programador é forçado a olhar seu código de maneira diferente, isso gera *designs* diferentes para o código.
* A segurança que o TDD gera para o *software* é incomparável, pois com testes qualquer mínima alteração tem de ser muito cuidadosa para não quebrar testes anteriores e nem os da *feature* em si.
* Novos programadores tem vantagem ao entrar no projeto. Em alguns casos o programador novo já tem projetos entregáveis no primeiro dia pois elimina uma necessidade de adaptação gigante.
* Testes descritivos ajudam muito na documentação fácil e enxuta.

Porém nem tudo é um mar de rosas, há também dificuldades (desvantagens) que fazem muitos quererem fugir desta prática:

* A curva de aprendizado é difícil, é algo que necessita de adaptação e prática, porém vem muito bem recompensada.
* O início no meio do projeto é difícil pois os *testers* tentam aplicar testes a código já existente perdendo algumas vantagens do TDD.
* O início da prática TDD é demorada, logo muitas empresas visando que tempo é dinheiro não se interessam em aplicar o método.

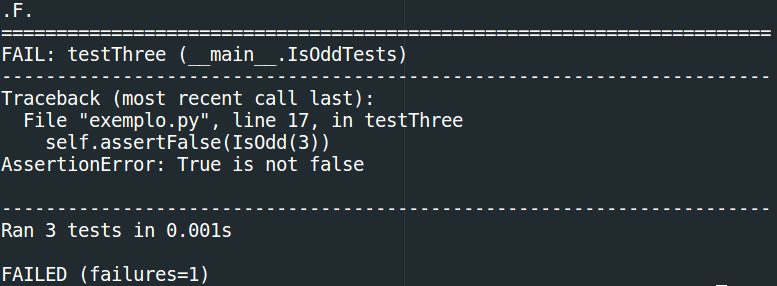
**Segunda etapa**

Na segunda etapa, a ideia principal é o nivelamento do conhecimento prático de criação de testes na linguagem específica escolhida, Python. Para isso não é necessário muito tempo graças a simplicidade da própria. É mostrado o seguinte código:



A partir dele, mencionamos as principais peculiaridades da linguagem para que seja possível desenvolver pequenos programas. Caso o problema a ser apresentado na terceira etapa tenha como requisito alguma estrutura mais avançada, esta deve ser mencionada neste momento do treinamento para que o foco permaneça apenas na prática do TDD. Além disso deixa-se o código projetado para que todos possam consultá-lo caso necessário.

Utilizamos a biblioteca “unittest”, que possibilita realizar testes unitários como demonstrado no código de exemplo. O resultado da execução do exemplo, que contém dois testes válidos e um falhando, é:



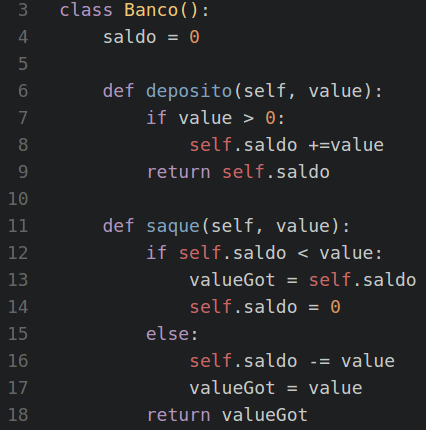
Apresenta-se um teste falhando para exemplificar o funcionamento da biblioteca de testes e, após esclarecimentos necessários sobre sintaxe, partimos para a terceira e última etapa do treinamento.

**Terceira etapa**

Neste último momento, é apresentado um problema real que deve ser resolvido com o uso do TDD. É orientado aos treinados para se unirem em pares e desenvolverem os testes. Dá-se o tempo necessário para desenvolver os testes baseado na problemática apresentada e, logo em seguida, apresenta-se a solução esperada para o problema até o momento. Como exemplo, temos uma classe Banco com alguns métodos vazios que devem ser preenchidos para tornarem os testes verdadeiros.



Perceba que os testes validam os valores retornados por cada função em determinadas situações. Apresente o código parcial com a classe montada e os testes falhando. A partir daqui, oriente os participantes a tornarem um dos testes verdadeiro e, só após isso, ir para o próximo teste. Dá-se mais tempo para escrever a solução e, ao final, mostra-se o código completo. O código que completa as funções de forma que os testes mostrados como exemplo pode ser visto abaixo.



Por fim, aponte as vantagens obtidas na prática. No exemplo, podemos mencionar que caso não houvesse o teste de saque inválido, a implementação da função saque poderia não considerar operações que solicitassem valores maiores que o saldo, o que resultaria em saldo negativo. Assim, exemplificamos uma maior qualidade de código como resultado do desenvolvimento orientado a testes. O código que foi utilizado como exemplo e um modelo de apresentação em slides estão disponíveis no repositório da disciplina.