TDD, por quê usar?

TDD é o desenvolvimento de software orientado a testes, ou em inglês, *Test Driven Development*. Mas mais do que simplesmente testar seu código, TDD é uma filosofia, uma cultura. Neste artigo você conhecerá um pouco sobre sua motivação e também saberá os fatores que contribuem e dificultam sua prática.

Muitos fatores contribuem para que inúmeros programadores deixem de utilizar TDD logo no início. Alguns deles são:

Dificuldade em começar

Apesar de uma extensa e clara documentação, iniciar o desenvolvimento orientado a testes pode ser um trabalho árduo para muitos pelo simples fato de que geralmente muitos iniciantes tentam praticá-lo em código já existente. Este definitivamente não é o caminho. A principal característica do desenvolvimento orientado a testes é que ele seja **orientado a testes**. Em outras palavras o código que realizará sua lógica deve ser criado somente após a criação do teste e isso torna-se algo de difícil aceitação pois ainda não se tem nada e já se faz necessário testar.

Curva de apendizado

Complementando o item anterior, este é outro motivo que faz programadores desistirem do desenvolvimento orientado a testes. Como qualquer nova tecnologia, para a pratica de TDD leva-se um bom tempo dependendo disponibilidade e principalmente da vontade do programador.

Tempo

Engana-se quem pensa que produzirá mais código pelo simples fato de utilizar TDD. O TDD na verdade chega a desacelerar a produção de código. Quando falo em produção de código, me refiro à quantidade de linhas escritas. Mas nisso tudo há vantagens e elas serão descritas mais a frente.

Cultura

Muito fala-se de TDD no Brasil, mas ao questionarmos programadores de diversas empresas muitos apresentam os motivos citados acima para não utilizá-lo. Existem sim muitas empresas e programadores que levam a prática a sério e a evangelizam justamente por conhecerem as vantagens que o TDD nos traz.

Nem tudo é tão ruim

Vistos alguns pontos que dificultam programadores a aderirem ao TDD, listamos agora alguns benefícios que esta prática nos fornece.

Qualidade do código

Um dos principais ensinamentos, senão o principal, do TDD é que se algo não é possível de ser testado então foi desenvolvido de forma errada. Parece um pouco drástico mas não é. Em pouco tempo utilizando testes o programador percebe mudanças relevantes em sua forma de programar. Em suma o uso de TDD ajuda o programador a elaborar um código com cada vez mais qualidade criando objetos concisos e com menos dependências.

Raciocínio

Para que o código torne-se mais conciso, tenha menos acoplamentos e dependências o programador deve forçar seu raciocínio a níveis elevados. É muito difícil criar algo que realmente tenha um bom design. Utilizando TDD o programador praticamente obriga-se a olhar seu código de outra forma normalmente jamais vista antes. Aí é que está a parte legal da coisa toda.

Segurança

Ponto importantíssimo para qualquer software nos dias de hoje. Mas não se engane, não estou falando de segurança da informação e sim de segurança ao desenvolver. Pense em uma situação em que o programador tenha um código que desenvolvera ha cerca de um ano. Como normalmente vivemos em um mundo com inúmeros softwares desenvolvidos ao longo de cada ano, torna-se muito difícil lembrar de tudo a respeito de um que merece nossa atenção em determinado momento. Normalmente deve-se realizar um trabalho bastante cauteloso para nova implementação em um software que encontra-se em produção. Toda e qualquer alteração deve ser minunciosamente testada e garantida que não afetará demais módulos do software. Fazer isto manualmente é realmente complicado pois até então não sabe-se (ou lembra-se) ao certo quem afeta quem no sistema. Com a prática de TDD cada pequeno passo do software está devidamente testado. Ou seja, com este cenário o programador pode realizar qualquer alteração sem medo e sem culpa.
Como cada pequeno passo tomado pelo sistema está testado ao qualquer módulo ou funcionalidade sofrer alteração, com poucos segundos descobre-se se houveram quebras e o melhor de tudo, onde foram essas quebras. Com isso em mãos a correção das quebras torna-se uma tarefa simples sem frustrar o cliente e o usuário.

Trabalho em equipe

Por prover mais segurança o trabalho em equipe torna-se muito mais proveitoso eliminando discussões e dúvidas desnecessárias. Ao entrar no desenvolvimento do projeto o novo desenvolvedor tem apenas o trabalho de entender qual task deve ser realizada e ler os testes das features já desenvolvidas. Ao rodar os testes pela primeira vez o programador descobre se está no caminho de ter um entregável mais rapidamente e com segurança. Existem empresas em que um novo programador tem entregáveis logo no primeiro dia de trabalho. Sem testes normalmente haveria um período de adaptação para prévio entendimento do que há no sistema no momento de seu ingresso ao time de desenvolvimento.

Documentação

Ao criar testes descritivos estes servem como uma excelente documentação para o software. Quando qualquer programador for rodar os testes, basta habilitar o modo *verbose* que uma “história” é contada eliminando o árduo trabalho de documentar um software onde nos meios tradicionais tende a defasar-se. O problema é que a documentação tradicional raramente segue o mesmo ritmo do desenvolvimento. Com os testes unitários a “documentação” é gerada antes mesmo da nova feature ser implementada e permanece fiel a qualquer alteração.

Até aqui temos os pontos negativos e positivos do desenvolvimento orientado a testes. Agora vamos aprender um pouco sobre o que é TDD.

O TDD (Test Driven Development) baseia-se em três passos, vermelho-verde-refatora. O **vermelho** é a escrita do primeiro teste antes mesmo da lógica existir. O **verde** é o ponto em que a lógica para que o teste previamente criado passe. Esta lógica deve ser desenvolvida da forma mais simples possível eliminando complexidades desnecessárias fazendo com que a evolução do código ocorra de forma segura. O **refatora** é a melhoria do código. Neste ponto são removidas duplicações, múltiplas responsabilidades e o código fica cada vez mais próximo de sua versão final.

Para que o processo vermelho-verde-refatora seja de fato implementado, utiliza-se baby steps ou passos de bebê. Esta técnica consiste em realizar um pequeno passo de cada vez, se uma lógica é complexa de ser desenvolvida ela é dividida em muitas pequenas partes que evoluem até sua solução final. Obviamente que esta técnica é aplicável preferencialmente em processos complexos que dependem de muitas variáveis. Para processos simples nem sempre é a mais indicada.

Os testes devem ser unitários. Isto implica em um teste automatizado certificar-se de apenas uma funcionalidade do código utilizando para isso quantos *asserts* forem necessários. Por serem testes automatizados são fácil e rapidamente executados eliminando testes manuais que encarecem o software em vários aspectos. Devido à dinâmica de tais testes é possível vincular o software com ferramentas de integração contínua como *Jenkins* reduzindo ainda mais o custo (tempo de desenvolvimento) do mesmo.

Concluindo

Como ficou claro a prática do desenvolvimento orientado a testes pode ser penosa para quem está começando mas muito satisfatória para quem já a tem como parte de seu dia a dia. Ao utilizar TDD o programador “perde” em linhas de código por hora mas ganha horas, dias, quem sabe até semanas na prevenção de novos erros e correção dos que ocorrem durante a implementação de uma nova feature.
Detalhe que o método TDD parte do princípio que o teste sempre deve ser implementado antes do código de produção.

Em suma tudo vai depender to porte do projeto em que se está trabalhando, da cultura da empresa e da vontade de programadores mas sabe-se que quem utiliza TDD reconhece suas vantagens.

Agora que você já tem uma noção do que é TDD, suas vantagens e desvantagens, sugiro a ler sobre *xUnit*. O xUnit é um framework de testes disponível para várias linguagens de programação. Em PHP temos o PHPUnit, em java o jUnit e demais linguagens possuem seus frameworks que tem a mesma finalidade, criar um ecossistema para que o programador desenvolva orientando-se a testes.

Palavras-chave sugeridas

TDD, testes automatizados, testes unitários, refatoração, xUnit

Referências

TDD → http://pt.wikipedia.org/wiki/Test_Driven_Development
xUnit → http://www.martinfowler.com/bliki/Xunit.html
PHPUnit →http://phpunit.de/manual/3.7/pt_br/automating-tests.html
jUnit → http://junit.org/

Andre Cardoso