TDD

Test-Driven Development

Test-Driven Development (TDD), sem dúvida, tornou-se uma das práticas mais populares entre desenvolvedores de software. A ideia é bem simples: escreva seus testes antes mesmo de escrever o código de produção. Mas por quê a ideia parece tão boa? Ao escrever os testes antes, o desenvolvedor garante que boa parte (ou talvez todo)

do seu sistema tem um teste que garante o seu funcionamento. Além disso, muitos desenvolvedores também afirmam que os testes os guiam no projeto de classes do sistema.

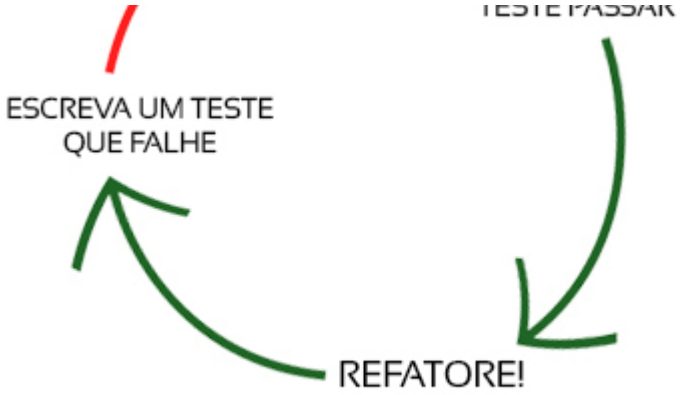
Mesmo com toda a indústria gritando as vantagens para quem queira ouvir, ainda existem mitos em torno da prática. *O desenvolvedor agora vai gastar mais tempo escrevendo testes do que programando? Escrever testes dá trabalho. Testes manuais não são mais produtivos? TDD deve ser feito 100% do tempo?*

Este guia visa responder essas e outras perguntas para você, que é desenvolvedor, gerente, ou está de alguma forma relacionado ao processo de desenvolvimento de software.

TDD

Como funciona?

A mecânica da prática é simples: escreva um teste que falha, faça-o passar da maneira mais simples possível e, por fim, refatore o código. Esse ciclo é conhecido como **Ciclo Vermelho-Verde-Refatora**.



Sempre que um desenvolvedor pega uma funcionalidade para fazer, ele a quebra (muitas vezes mentalmente) em pequenas tarefas. Tarefas essas que exigem a escrita de código. Classes são criadas, outras são modificadas. Todas essas modificações tem um propósito, claro. Todo código escrito tem um objetivo.

Ao praticar TDD, o desenvolvedor antes de começar a fazer essas modificações explicita esses objetivos. Só que ele faz isso por meio de testes automatizados. O teste em código nada mais é do que um trecho de código que deixa claro o que determinado trecho de código deve fazer.

Ao formalizar esse objetivo na forma de um teste automatizado, esse teste falha, claro; afinal, a funcionalidade ainda não foi implementada. O desenvolvedor então trabalha para fazer esse teste passar. Como? Implementando a funcionalidade.

Assim que o teste passar, o desenvolvedor então parte para uma próxima etapa no ciclo, importantíssima para aqueles que tem o sonho de produzir código de qualidade: a hora da refatoração. Refatorar é melhorar o código que já está escrito. A cabeça do desenvolvedor é complicada: quando ele está focado em implementar a

funcionalidade, ele raramente está pensando também em qualidade de código. Não tem jeito, é assim que funcionamos. E justamente por isso que, após a implementação da funcionalidade, o desenvolvedor para e melhora a qualidade do código (que já funciona e atende ao requisito do negócio).

Acabou? Claro que não. Agora é partir para a próxima funcionalidade. E começando por onde? Pelo teste.

Vantagens

O que eu ganho com a prática?

A prática de TDD agrega muitos benefícios ao processo de desenvolvimento. O primeiro deles, e mais claro, são os benefícios na qualidade externa do produto. Todos já sofremos os problemas de uma nova versão do produto que traz novas funcionalidades, mas faz as anteriores pararem de funcionar. A bateria de testes automatizados gerados pela prática dão mais segurança ao desenvolvedor na hora de mudanças.

Os testes automatizados, que rodam em questão de segundos, são executados o tempo todo pelo desenvolvedor. Isso quer dizer que podemos executá-los o dia

todo, muitas vezes por dia. Algo que sabemos ser impossível com testes manuais. Caso algo pare de funcionar, o desenvolvedor é rapidamente notificado, e consegue corrigir o problema antes de mandar a versão para o cliente. E todos nós sabemos o quanto é bom não estressar o usuário final com bugs, não é verdade?

Além disso, muitos autores populares da área afirmam que, caso o desenvolvedor saiba ler o código de testes com atenção, esse mesmo código pode dar informações importantes sobre a qualidade do código que está sendo produzido. Dizemos que a prática de TDD ajuda o desenvolvedor a escrever código de produção de qualidade. É difícil explicar esses efeitos em poucas palavras, mas a ideia geral é que *se está difícil escrever um teste automatizado, é porque provavelmente o código de produção está complicado*. Essa é uma ótima dica para o desenvolvedor.

Perceba então que o uso da prática de TDD ajuda a equipe a garantir que os requisitos funcionam como esperado, e também auxilia o desenvolvedor a perceber problemas de código em suas implementações. Dois coelhos em uma cajadada só.

Devo praticar TDD 100% do tempo?

A resposta para essa pergunta serve para toda e qualquer prática de engenharia de software. **É claro que não.**

Como já discutido anteriormente, TDD faz com que o desenvolvedor teste melhor sua aplicação, bem como pense em um design de classes melhor e mais flexível para aquele problema. Mas não é sempre que precisamos disso. Se estamos, por exemplo, escrevendo a implementação de um DAO (classe que se comunica com o banco de

dados), talvez escrever os testes antes não vá ajudar tanto, afinal não há grandes decisões de design a serem tomadas em classes como essa, e a funcionalidade tende a ser simples. Escrever o teste depois, portanto, não será tão diferente de escrever o teste antes.

O desenvolvedor maduro leva em consideração a sua experiência, e entende bem as vantagens da prática. E, na hora certa, fazer uso dela.

Qual a diferença entre fazer TDD e escrever o teste depois?

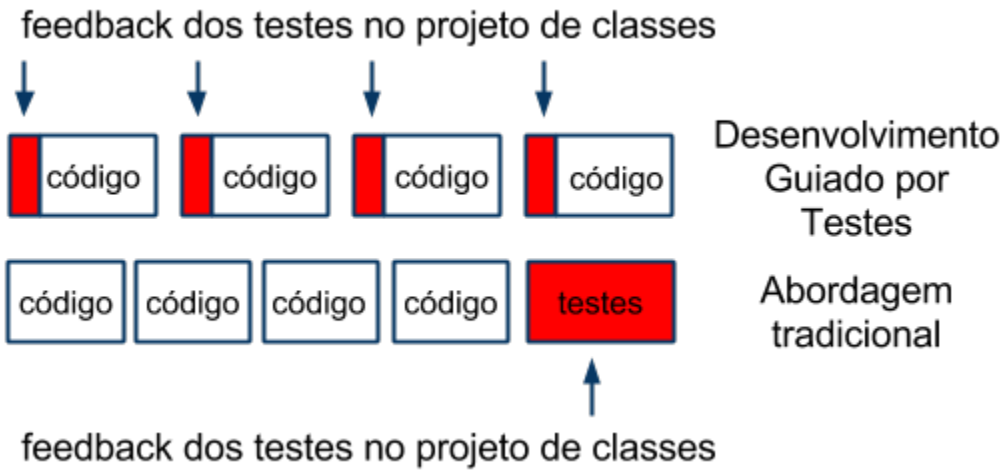
Se pararmos para analisar, o grande responsável pelo aumento da qualidade interna e externa não é o TDD, mas sim o teste automatizado, produzido perante o uso da prática. A pergunta comum é justamente então: *Qual a diferença entre fazer TDD e escrever o teste depois?*

O desenvolvedor obtém feedback do teste. A diferença é justamente na quantidade de feedback. Quando o desenvolvedor escreve os testes somente ao acabar a implementação do código de produção, ele passou muito tempo sem retorno. Afinal, escrever o código de produção leva tempo. Ao praticar TDD, o desenvolvedor divide seu trabalho em pequenas etapas. Ele escreve um pequeno teste, e implementa um pedaço da funcionalidade. E repete. A cada teste escrito, o desenvolvedor ganha feedback.

Quanto mais cedo o desenvolvedor receber feedback, melhor. Quando se tem muito código já escrito, mudanças podem ser trabalhosas e custar caro.Ao contrário, quanto menos código escrito, menor será o custo de mudança. E é justamente isso que acontece com praticantes de TDD: eles recebem feedback no momento em que

mudar ainda é barato.

A figura abaixo exemplifica a diferença entre a quantidade de feedback de um desenvolvedor que pratica TDD e de um desenvolvedor que escreve testes ao final.



Benefícios

Serei mais ou menos produtivo?

Assim como muitas das práticas ágeis, é difícil ver os benefícios quando não se faz uso dela. A primeira reação da maioria das pessoas é: *"Mas agora eu gastarei boa parte do meu tempo escrevendo testes? Isso não pode ser produtivo!"*

A resposta para a pergunta é: **sim, o desenvolvedor gastará boa parte do**

seu tempo escrevendo código de testes. Mas isso não quer dizer que ele seja menos produtivo por causa disso.

Antes de partir para argumentos, é necessário definirmos o que é ser produtivo. Para muitos, produtividade é simplesmente a quantidade de linhas de código de produção que são escritas por dia. Aqui, a definição de produtividade será linhas de código escritas com qualidade, de fácil manutenção, e que dão pouca (ou nada) de re-trabalho.

Para compararmos as vantagens da escrita de testes automatizados, usarei o contra-exemplo: o teste manual. O dia-a-dia do desenvolvedor que faz teste manual é algo parecido com isso: ele programa a funcionalidade (geralmente toda ela de uma vez) e roda a aplicação. Com a aplicação de pé, ele faz o primeiro teste manual, navegando pela aplicação e digitando os diversos dados de entrada necessários para fazer o teste. Muito provavelmente o software não funcionará de acordo.

Ele então é obrigado a procurar pelo problema, lendo as 300 linhas de código que escreveu, ou mesmo debuggando. Debugar é a atividade onde o desenvolvedor executa linha por linha de código e vê o resultado. Ambas as atividades claramente desperdiçam um tempo imenso do desenvolvedor.

Em algum momento, o desenvolvedor encontrará o problema. Ele o corrijirá, e aí repetirá todo o processo: subirá a aplicação e fará o teste manual. Muito provavelmente outro problema aparecerá. Dessa vez em um ponto mais adiante da regra de negócio, claro. Ele então novamente repetirá o processo.

Veja o quanto isso é demorado e caro. O desenvolvedor que faz teste manual repete o mesmo teste várias vezes por dia. O desenvolvedor que o automatiza gasta seu tempo apenas uma vez. Na próxima vez, o teste será executado pela máquina em poucos milissegundos.

Mais ainda, sempre que o desenvolvedor precisar testar novamente no futuro, ele o fará de maneira manual, gastando tempo. Já o desenvolvedor que tem testes automatizados, apenas executará sua bateria de testes. Ela durará pra sempre e poderá ser executada quantas vezes quiser.

Ou seja, o desenvolvedor que escreve testes automatizados gasta tempo com isso. Mas ele gasta tempo de maneira inteligente. Hoje, o desenvolvedor que faz teste manual também gasta muito tempo com testes, mas de maneira errada, improdutiva. A médio prazo, esse desenvolvedor gastará mais tempo testando a mesma funcionalidade do que o que foi esperto e os automatizou desde o começo. É tudo uma questão de pensar a médio prazo.

Estudos científicos

Alguém já fez estudos formais sobre isso?

É difícil acreditar em tudo que foi dito até agora, não? Pois bem, é para isso que servem trabalhos científicos. Para que fatos sejam separados de meros folclores.

Podemos separar estudos sobre TDD em 2 categorias diferentes. Aqueles que olham TDD como uma prática de teste de software, e por consequência avaliam os efeitos dele na qualidade externa do software; e aqueles que olham TDD como uma prática de design e estão preocupados com os efeitos dele na qualidade interna do sistema.

Nos últimos anos, a comunidade acadêmica vem rodando diversos experimentos para tentar mostrar de maneira empírica que TDD realmente ajuda no processo de desenvolvimento de software. Alguns desses estudos são feitos por professores bastante conhecidos na comunidade, como a prof. Laurie Williams (North Carolina State University) e o prof. David Janzen (California Polytechnic State University).

Esses estudos nos mostram que desenvolvedores que praticam TDD gastam menos tempo debugando, escrevem mais testes automatizados para uma funcionalidade, e defeitos são encontrados mais rapidamente, do que por aqueles que não praticam TDD. Em termos de qualidade interna, os estudos mostram que os desenvolvedores tem uma forte percepção de que a prática os ajuda a pensar melhor sobre seu projeto de classes.

Você pode ler muitos desses estudos com mais detalhes em um post do meu blog, chamado TDD Realmente Ajuda?.

Referências

Onde posso ler mais sobre

isso?

Livros sobre TDD, apesar de não serem muitos, são bons. Todos são bastante técnicos, e devem ser lidos apenas por desenvolvedores.

O primeiro livro sobre o assunto, escrito pelo Kent Beck, Test-Driven Development: By Example é um livro para iniciantes. Ao longo dele, o autor desenvolve duas classes do começo ao fim, e explica passo-a-passo como TDD é feito. Os exemplos são bem minimalistas, mas é um excelente primeiro contato com a prática.

Outro livro importante para aqueles que querem se aprofundar é o Growing Object-Oriented Software, Guided by Tests, escritos pelos excelentíssimos autores Steve Freeman e Nat Pryce. Esse é um livro mais pesado; os exemplos são maiores e eles discutem bastante sobre como uma aplicação do zero deve ser criada a partir da prática de TDD. Apesar dos exemplos fazerem uso de Swing, e o leitor encontra por muitas vezes extensas listas de código, é um livro que vale a pena ser lido, caso o desenvolvedor seja mais maduro.

Um livro menos popular, mas também interessante é o Test-Driven Development: A Practical Guide, do Dave Astels. Ele também dá exemplo de uma aplicação do zero, e introduz conceitos interessantes como Mock Objects.

Além disso, o primeiro livro em português brasileiro sobre o assunto, TDD: Teste e Design no Mundo Real, escrito por Maurício Aniche, é uma boa opção para aqueles que querem ver no mesmo livro, exemplos básicos para quem está começando, e exemplos mais avançados sobre a relação entre TDD e design de classes. O livro foi baseado em sua pesquisa de mestrado sobre o assunto.

Existe também muito material informal sobre o assunto. O próprio blog do Aniche, e o blog da Caelum possuem bons textos. Abaixo uma pequena relação desses posts:

Perguntas e Respostas sobre TDD
Bate papo sobre TDD na Caelum
Dependência de cenários em testes de sistema
Quantidade de Asserts no Teste
Testando datas e métodos estáticos
Será que eu preciso de 100% de cobertura de código?
Um pequeno estudo sobre asserções em testes
É TDD, e não DDT
Criando cenários de teste com Fixture Factory
O que a quantidade de asserts em um teste nos diz sobre o código?
Facilitando a manutenção dos testes ao diminuir o acoplamento com o código
TDD e sua influência no acoplamento e coesão
Ganhando ou perdendo tempo com testes de unidade.

Além disso, Maurício Aniche também tem diversas publicações científicas sobre o assunto.

Treinamentos

Como treinar minha equipe?

Muitas vezes a melhor maneira de introduzir uma nova prática de desenvolvimento para a equipe é trazendo alguém com mais experiência

sobre o assunto, para ensinar, discutir e buscar a melhor maneira para introduzi-la no processo atual de desenvolvimento de software.

A Caelum oferece treinamentos online e presencial sobre o assunto. Os cursos oferecidos, bem como as ementas, podem ser vistas na página da trilha de testes do Alura, o portal de ensino a distância da Caelum.

Maurício Aniche também é palestrante sobre o assunto nos mais diversos eventos brasileiros de desenvolvimento de software, como QCON e Agile Brazil. Ele também dá workshops sobre o assunto para empresas privadas e públicas.

Entre em contato.

Versão

Versões do documento

Abril/2014: Versão inicial do documento.

Sobre o Autor

Maurício Aniche é mestre em Ciência da Computação pela Universidade de São Paulo, onde pesquisou sobre os reais efeitos da prática de TDD no

design de classes. Atualmente é aluno de doutorado pelo mesmo instituto. Mauricio opta por ter um pé no mundo da indústria e outro no mundo da academia. Já palestrou em diversos eventos da indústria como QCON, Agile Brazil, .NET Architects, e já publicou em conferências acadêmicas nacionais e internacionais como TDD2010 (Paris), Agile 2011 (EUA), WBMA 2011 (Brasil). O mestrado fez mal a ele, já que ele deixou de acreditar em post de blogs e twitters de aficcionados pelas metodologias ágeis.É também autor do livro "TDD: Teste e Design no Mundo Real", o livro brasileiro mais popular sobre o assunto.

Organizadores

Maurício Aniche 

Conheça também os livros da
Casa do Código