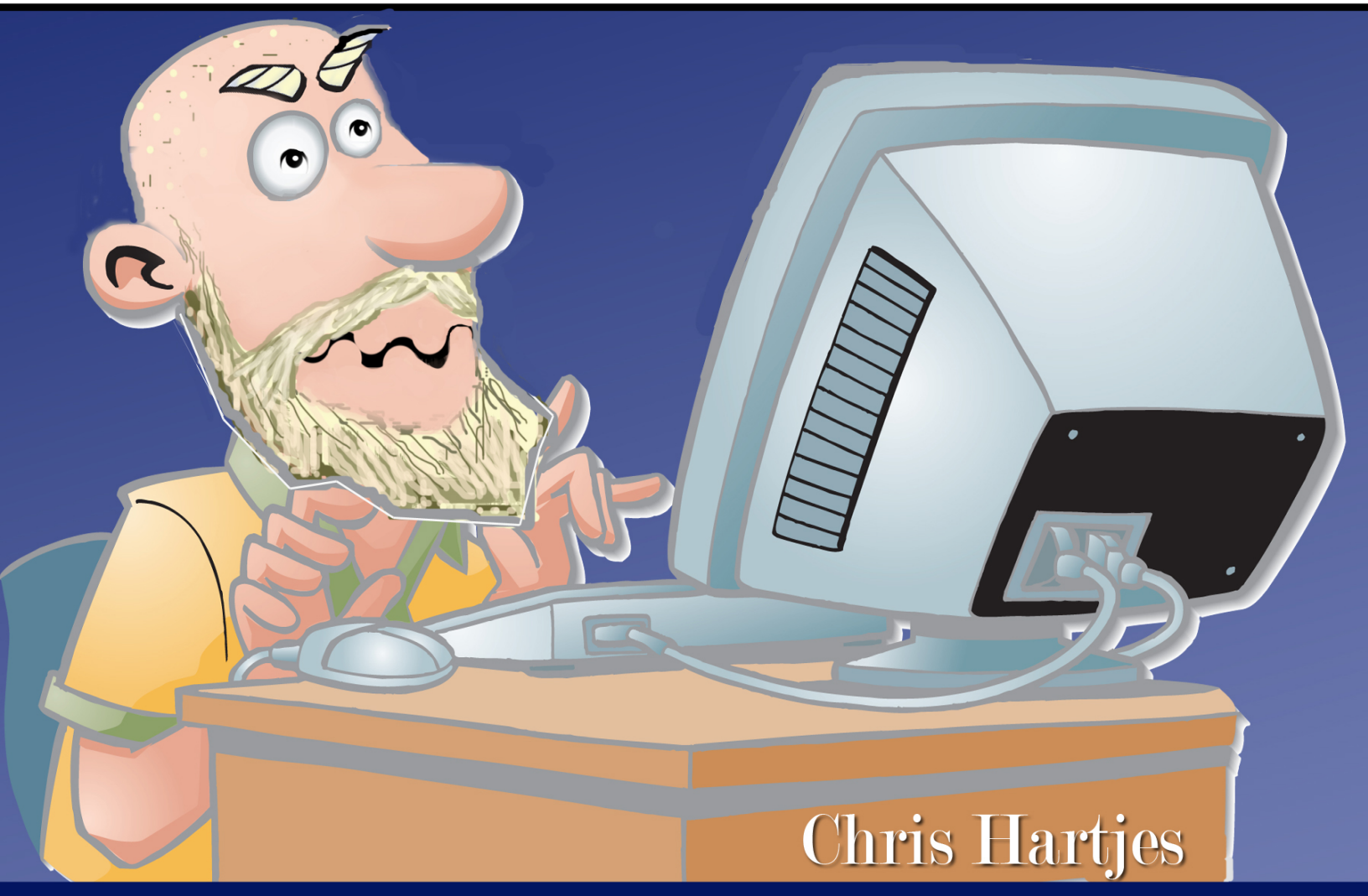
The Grumpy Programmer's

Guia para criação de aplicações testáveis em PHP



Chris Hartjes

The Grumpy Programmer's Guia para criação de aplicações testáveis em PHP

Chris Hartjes e Nanderson Castro

Esse livro está à venda em http://leanpub.com/grumpy-testing-br-pt

Essa versão foi publicada em 2017-01-17

Conteúdo

Agradecimentos **1**

Contruir aplicações testáveis em PHP! Uma tarefa difícil **3**

Agradecimentos

Em primeiro lugar, eu gostaria de agradecer meus revisores técnicos. Eles conseguiram pegar todos os detalhes que passaram desapercebidos por mim e fizeram deste guia o que ele é. Obrigado também (sem ordem especifica) a Peter Meth, Joel Perras, Matthew Turland e Remi Woler pela ajuda de todos. Bons guias iniciam a partir da ajuda de todos.

Em segundo lugar, eu gostaria de agradecer a todos da comunidade PHP que estão sempre levando adiante, em alto e bom som, a importância de se seguir as melhores práticas e contribuem para o crescimento da mesma. Aqueles que sempre me inspiram a continuar escrevendo e falando de assuntos que me interessam e espero que lhe interesse também. Eu posso me sentir, muitas vezes, fora de sincronia com a comunidade "mainstream" PHP mas se o meu trabalho online e delirante inspira outros programadores em seus trabalhos, então sou muito grato pelo impacto que alcancei.

Finalmente, eu agradeço a minha esposa Claire por estar sempre comigo nessa minha viagem de um programador recém-formado a um bem preparado, extremamente mal-humorado que sou hoje. Obrigado pelas longas horas de aturação na qual um programador passa olhando para telas ao invez de olhar para seus lindos olhos. Minha esposa é minha parceira em tudo que faço.

Como sempre, seus comentários são muito bem vindos neste guia, você deve ter inúmeras dúvidas ou apenas deseja obter meus pensamentos sobre o vasto mundo que é a programação. E baseball também.

Este guia foi produzido com a ajuda de incrivéis pessoas na Leanpub[1]. Minha decisão inicial ao fazer o guia como texto com algumas marcações AsciiDoc[2] tornou as coisas fáceis ao converte-las para o Markdown[3] como o Leanpub requer. Eu também usei Pandoc[4] para ajuste dos três bônus que incluir neste guia.

Infelizmente este guia não visa o programador iniciante, escrever aplicações testáveis é um coisa bem difícil e requer uma base bem sólida em programação. Este guia é destinado aos programadores intermediários que tiveram algum contato com a escrita de testes e desejam melhorar sua habilidades, passando para um próximo nível.

Se você está procurar por um guia introdutório para escrita de testes unitários eu recomendo fortemente o PHPUnit documentation[5]. Ele irá cobris o básico necessário.

Você pode me encontrar via email chartjes@littlehart.net e também no Twitter @grmpyprogrammer. Eu mantenho um blog em http://www.littlehart.net/atthekeyboard. Obrigado pelo tempo gasto

[1] http://leanpub.com

[2] http://www.methods.co.nz/asciidoc/

[3] https://en.wikipedia.org/wiki/Markdown

[4] http://johnmacfarlane.net/pandoc/

[5] http://www.phpunit.de/manual/3.6/en/phpunit-book.html

ao analisar o guia, espero que você possa encontrar as dicas necessárias para melhorar sua capacidade ao criar aplicações testáveis com PHP.

Contruir aplicações testáveis em PHP! Uma tarefa difícil

Se você é desenvolvedor PHP a muito tempo você muito provavelmente já encontrou alguém que fale sobre os benefícios dos testes em seus códigos. Test-Driven Development (TDD)[6], Behaviour-Driven Development (BDD)[7] e Functional Testing[8] são apenas algumas das estratégias que você pode usar. Porém deixar de testar seu código não deve ser uma opção.

Eu já passei por isso também: Eu não acreditava no valor de escrever testes para meu código. Estou familiarizado com todas as desculpas que as pessoas podem dar:

- "Irei me atrasar ao escrever testes"
- "Por que não podemos simplesmente testar manualmente a aplicação no browser?"
- "Eu já teria implementado as novidades se eu não tivesse que escrever os testes"

Realmente, elas são só desculpas. Dadas as ferramentas disponíveis para os desenvolvedores PHP essas não são razões para não escrever testes para seu aplicativo. Até mesmo aplicativos que resistem aos seus esforços para refatorar componentes individuais e envolver testes de unidade em torno deles, o temido "o aplicativo não testável", pode ser testado em um ambiente automatizado.

As razões pelas quais você deve investir em testes automatizados são óbvias: qualquer bug capturado antes de sua aplicação ir para produção custa menos em termos de recursos(tempo de desenvolvimento, dinheiro) para corrigir, do que o fazer em produção. Em um estudo que a Microsoft e IBM conduziram sobre TDD[9] eles encontraram algumas informações interessantes. Um projeto usando TDD levaria de 20 a 40% mais tempo do que um que não use TDD. Porém, as aplicações resultantes continham de 40 a 90% menos bugs, que apenas foram descobertos em produção. Se você ajustar as estatísticas apenas para a direita, como pode 20% de tempo extra e 90% com menos bugs? Como alguém pode dizer não a isso?

PHPUnit[10], a bala de prata para testes em códigos PHP, é facilmente integrado a qualquer projeto. Instale via Composer[11], crie um diretório para seus testes em sua aplicação e inicie as escritas. Porém, nem sempre isso será tão simples. Nem toda aplicação estar pronta ou configurada para testes.

Você usa um sistema de autenticação de usuários de terceiros? Você terá que descobrir como simular as respostas deste sistema. Você têm valores embutidos para conversar com serviços da web? Você

[6] https://en.wikipedia.org/wiki/Test-driven_development
[7] https://en.wikipedia.org/wiki/Behaviour_driven_development
[8] https://en.wikipedia.org/wiki/Functional_testing
[9] http://www.infoq.com/news/2009/03/TDD-Improves-Quality
[10] https://github.com/sebastianbergmann/phpunit/
[11] https://getcomposer.org

terá que extrair tais coisas e coloca-las em um arquivo de configuração. A verdade é que diversas aplicações precisam ser refatoradas antes sequer de você poder testá-las.

Neste guia irei mostrar como você pode construir uma aplicação que é facilmente testável. É claro, nós estamos começando do zero então isto será fácil mas também irei mostrar alternativas para refatoração de código que não serão tão amigáveis assim!

Em cada capitulo nós iremos abordar diferentes tópicos que irão incrementar suas habilidades na criação de aplicações que você poderá facilmente testar. Alguns dos problemas que você vai encontrar são bastante fáceis de se resolver enquanto outros podem exigir algumas mudanças intrusivas para que isso aconteça.

Para ilustrar algumas, das que considero, boas práticas para criação de aplicações facilmente testáveis, irei criar uma aplicação exemplo que utiliza muitas das dicas que eu passarei aqui. Você pode encontrar o código da aplicação em meu Github[12].

Nossa aplicação companheira é baseada em um website que simula uma liga de baseball. Estou na Internet Baseball League[13] desde 1997 e criei esta versão do site usando uma versão muito desatualizada do CakePHP[14]. Eu certamente pego o trabalho feito mas tenho vontade de limpar as coisas que estão acontecendo por baixo do capô e tirar proveito de algumas características do PHP 5.3+

Ao invés de duplicar toda a aplicação, eu extrair as partes que geram a página principal do site. Esta provavelmente é a parte mais complicada do site voltada para o público. É incorporado o uso de múltiplos models e é bastante complicado o uso de templates.Um excelente candidato para realçar o valor dos testes para verificar suas funcionalidades.

Apesar de ter adquirido uma reputação ao logo dos anos como sendo um Mister Framework, nós não usaremos nenhum framework aqui. Eu achei importante mostrar a você que, apesar dos maiores frameworks do 'mundo dos frameworks php' serem mais fáceis de se escrever testes para aplicações usando os mesmos, você pode facilmente cobrir qualquer aplicação com testes.

Estou usando uma combinação de características e padrões de projeto. Embora eu não seja viciado em padrões, eu acredito nas soluções certas para os problemas certos. Vamos dar uma olhada no que estamos usando:

- PHP 5.3.8 instalado no OS-X 10.7.2 via Homebrew[15]
- A extensão Xdebug[16] para code-coverage reports
- A extensão APC[17] para impulsionar a performance PHP
- Twig[18] para templates

[12] https://github.com/chartjes/building-testable-applications
[13] http://www.ibl.org
[14] http://www.cakephp.org
[15] http://notfornoone.com/2010/07/install-php53-homebrew-snow-leopard/
[16] http://www.xdebug.org
[17] http://www.php.net/apc
[18] http://twig.sensiolabs.org

- Postgres[19] como banco de dados
- Pimple[20], um container de injeção de dependências que é usado como um registro de objetos e valores necessários para outras partes da aplicação

Eu também testei esta aplicação em um Ubuntu 10.04 (Lucid) usando uma máquina virtual em meu MacBook. Mais informações sobre o papel das máquinas virtuais em testes, abordaremos em um capitulo posterior.

Para a arquitetura da aplicação, eu decidi usar uma estrutura de Page Controller para cada sessão da aplicação. Se você está usando um framework provavelmente você estará usando um Front Controller. Isto significa que todas as requisições irão passar por um único arquivo e em seguida o framework irá interpretar os pedidos e determinar qual par de controller e ação precisarão ser executados.

Um padrão Data Mapper como descrito no incrível livro do Jason Sweat's sobre Design Patterns[21] é usado para prover o acesso a base de dados. Se você precisar mudar seu provedor de dados, se ele estiver crescendo para uma solução em cluster ou a mudança será para algum dos NoSQL disponíveis no mercado, um Data Mapper irá ajudar você nesta mudança tornando esse trabalho mais fácil.

Neste guia eu apenas dei uma olhada nos testes que acompanham o nosso artigo companheiro. Por favor, baixe uma cópia do código e brinque com seus próprios testes. O código pode ser encontrado em https://github.com/chartjes/building-testable-applications[22], o código em mãos tornará mais fácil a compreensão dos conceitos descritos neste guia.

[19] http://www.postgresql.org
[20] http://pimple.sensiolabs.org
[21] http://www.phparch.com/books/phparchitects-guide-to-php-design-patterns/
[22] https://github.com/chartjes/building-testable-applications