Aprendendo

Padrões de Projeto em PHP

William Sanders

Novatec

Editor: Rubens Prates
Tradução: Lúcia A. Kinoshita
Revisão gramatical: Naomi Yokoyama Edelbuttel
Editoração eletrônica: Carolina Kuwabata

ISBN: 978-85-7522-343-7

Histórico de impressões:

Maio/2013 Primeira edição

Novatec Editora Ltda.
Rua Luís Antônio dos Santos 110
02460-000 – São Paulo, SP – Brasil
Tel.: +55 11 2959-6529
Fax: +55 11 2950-8869
E-mail: novatec@novatec.com.br
Site: www.novatec.com.br
Twitter: twitter.com/novateceditora
Facebook: facebook.com/novatec
LinkedIn: linkedin.com/in/novatec

VC20130517

CAPÍTULO 1

PHP e programação orientada a objetos

Nenhuma força no mundo é tão poderosa quanto a de uma ideia cuja hora chegou.

– Victor Hugo

Não reze por tarefas iguais aos seus poderes. Reze por poderes iguais às suas tarefas.

– Phillips Brooks

Um imenso poder será conquistado se, em seus devaneios secretos, você disser a si mesmo que nasceu para controlar as situações.

– Andrew Carnegie

A ignorância é a maldição de Deus; o conhecimento é a asa com que voamos para o céu.

– William Shakespeare

Introdução à programação intermediária e avançada

Quando aprendemos a ler, as histórias, o vocabulário e as palavras tendem a ser pequenos e simples. Lidar com histórias pequenas e simples exige ferramentas pequenas e simples. Contudo, quando estamos em um estágio mais avançado e somos apresentados às obras de William Shakespeare, precisamos de um conjunto de ferramentas mais complexo, maior e mais sofisticado. Se uma professora do Ensino Fundamental entregar *Hamlet* aos seus pequenos, é mais provável que as crianças não o entenderão; porém, se adquirirem um conjunto adicional de ferramentas de leitura ao longo dos anos, quando atingirem o Ensino Médio, elas poderão ler, entender e apreciar *Hamlet*. Este livro é destinado a desenvolvedores que estão prontos para ler a versão em PHP de *Hamlet*.

Para obter o necessário deste livro, será preciso começar por compreensão e experiência com PHP. Outros livros desta série, *Learning PHP 5,* de David Sklar, e *Learning PHP, MySQL, and JavaScript, 2nd Edition,* de Robin Nixon (O'Reilly), constituem bons pontos de partida, caso você não tenha nenhuma experiência com PHP. É claro que você pode ter aprendido PHP a partir de vários outros livros, cursos ou tutoriais online. O que importa é que você saiba programar em PHP. Além do mais, lidaremos com PHP 5 e não com versões anteriores, como a última versão de PHP 4 (PHP 4.4.9). Isso porque praticamente tudo o que precisamos para programação orientada a objetos (POO) não havia sido implementado até o PHP 5.

Por que programação orientada a objetos?

Embora a POO já exista há mais de 40 anos, foi somente nos últimos 15 anos, aproximadamente, que ela se tornou cada vez mais importante. Em grande medida, isso aconteceu em virtude da influência do Java, o qual inclui estruturas de POO embutidas. Linguagens mais novas ligadas à internet, como por exemplo JavaScript, ActionScript 3.0 e PHP 5 também incorporaram POO, seja no estilo ou na estrutura. Em 1998, *JavaScript Objects,* de Alexander Nakhimovsky e Tom Myers (Wrox), dois professores da Colgate University, mostraram que a POO poderia ser incorporada em JavaScript. Sendo assim, POO não é nenhuma novidade, mesmo para aqueles que vêm programando especialmente no domínio das linguagens de internet, e podemos até dizer que é um método "testado e aprovado" de programação para a maioria das linguagens concebidas para fornecer instruções aos computadores.

Investir algum tempo para entender POO é importante porque a compreensão dos padrões de projeto baseia-se na compreensão de POO. Embora você possa ter uma vasta experiência de programação em PHP 5, se não tiver experiência com POO, invista algum tempo na parte I.

Facilitando a resolução de problemas

Programas de computador foram projetados para solucionar problemas humanos. Um processo chamado de *programação dinâmica* consiste em uma técnica para dividir problemas maiores em partes menores. O plano é resolver cada um dos problemas menores e então reunir tudo em uma única solução mais ampla. Considere, por exemplo, o planejamento de uma viagem para Timbuktu (não parece ser um problema complexo, mas veja se você consegue um voo de sua cidade para Timbuktu em um site de viagens). Vamos dividir o problema:

1. Timbuktu (também Tombouctou ou Tombuctu) existe? (Sim/Não) Resposta = Sim.
2. Timbuktu tem um aeroporto? (Sim/Não) Resposta = Sim. Identificador do aeroporto = TOM.
3. Existem voos para TOM? (Sim/Não) Resposta = Talvez. Há voos disponíveis de Bamako e Mopti; porém, rebeldes islâmicos assumiram o controle de Timbuktu desde 1º de julho de 2012 e os voos foram cancelados até nova ordem.
4. Rebeldes hostis estão no controle de Timbuktu no momento? (Sim/Não) Se resposta = Sim → não há voos. Se resposta = Não → poderá haver voos.
5. Se houver voos disponíveis, Timbuktu é seguro para turismo ou negócios? (Sim/Não) Resposta = Não.
6. É possível conseguir visto de meu país para Mali (país em que Timbuktu está localizado)? (Sim/Não) Resposta = Sim.
7. Há exigências quanto a vacinas? (Sim./Não.) Resposta = Sim.

Como você pode perceber, ir e vir de Timbuktu é um problema complexo, mas a lista de perguntas simples pode ser respondida com *sim* ou *não*. Muitas outras perguntas poderiam ser incluídas na lista, mas cada uma delas poderá ser respondida de modo binário. A resposta "talvez" significa que mais perguntas devem ser feitas para obter uma resposta do tipo sim/não.

Modularização

O processo de decomposição de um problema em subproblemas menores corresponde ao processo de *modularização*. Assim como a complexidade de levar você de sua casa até Timbuktu pode ser modularizada em um conjunto de passos do tipo sim/não, qualquer outro problema complexo também pode ser modularizado. A figura 1.1 ilustra esse processo.

Ao olhar para a modularização, você poderá pensar que ela não parece ser tão difícil. Você está absolutamente certo. Quanto mais complexo o problema, mais faz sentido modularizá-lo. Desse modo, o raciocínio inicial na POO, longe de ser complexo, simplifica o que é complexo. Até mesmo o problema de programação mais desanimador pode ser resolvido usando essa estratégia de dividir e conquistar.

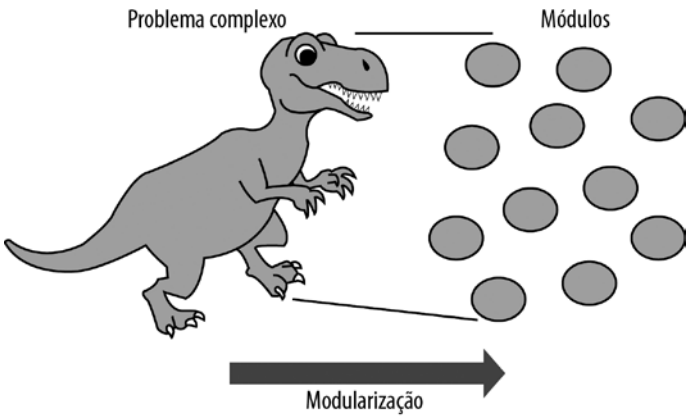

Figura 1.1 – Até mesmo o problema mais complexo pode ser dividido em módulos.

Classes e objetos

Uma vez que o problema for modularizado, o que você fará com os módulos? Conforme visto, dividir um problema complexo pode transformá-lo em vários subproblemas simples, mas é necessária uma maneira de organizar os módulos e trabalhar com eles, uns em relação aos outros, para lidar com o problema maior sendo resolvido. Uma maneira de ver um módulo é encará-lo como uma coleção de funções relacionadas. Em programação, esses módulos são chamados de *classes*.

Uma classe é composta de partes chamadas de propriedades e de métodos. As *propriedades* são diferentes tipos de objetos de dados, como números, strings, nulos e booleanos. Em geral, os dados são armazenados na forma de tipos de dados abstratos, conhecidos como variáveis, constantes e arrays. Por outro lado, *métodos* são funções que operam sobre os dados.

Princípio da responsabilidade única

Uma maneira de pensar em uma classe é imaginá-la como uma coleção de objetos que têm características comuns. Dizer que há algo "comum" nas características não significa que eles sejam iguais, mas que lidam com o problema comum atribuído ao módulo – a classe. Tendo em mente que o propósito de um módulo é resolver algum aspecto de um problema mais complexo, chegamos a um dos primeiros princípios da programação orientada a objetos: o princípio da responsabilidade única, que afirma que uma classe deve ter somente uma única responsabilidade.

Isso não significa que uma classe não possa ter várias responsabilidades, mas não se esqueça de que dividimos um problema complexo em módulos simples para que dessa forma tivéssemos vários problemas fáceis de serem solucionados. Ao limitar uma classe a uma única responsabilidade, não só nos lembramos do motivo pelo qual modularizamos o problema, mas teremos também uma maneira mais fácil de organizar os módulos. Vamos dar uma olhada em uma classe com uma única responsabilidade. Suponha que você esteja criando um site para um cliente, e como o site será visualizado em diferentes dispositivos, que variam de desktops a tablets ou smartphones, você quer ter alguma maneira de determinar o tipo de dispositivo e qual navegador é usado para visualizar sua página na web. Em PHP é fácil escrever uma classe que disponibilize essas informações utilizando o array existente `$_SERVER` e o elemento relacionado, `HTTP_USER_AGENT`. A classe `TellAll` na listagem a seguir mostra uma classe com uma única responsabilidade – prover informações sobre o agente de usuário que está visualizando a página PHP:

```
<?php
//Salvo como TellAll.php

class TellAll {
    private $userAgent;

    public function __construct() {
        $this->userAgent=$_SERVER['HTTP_USER_AGENT'];
        echo $this->userAgent;
    }
}

$tellAll=new TellAll();

?>
```

Carregar essa classe por meio de um navegador Safari em um iMac fará a seguinte informação ser apresentada:

```
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_4) AppleWebKit/ 534.57.2 (KHTML, like Gecko)
Version/5.1.7 Safari/534.57.2
```

Se testada em um iPad usando um navegador Opera Mini, o resultado será diferente:

```
Opera/9.80 (iPad; Opera Mini/7.0.2/28.2051;U;en) Presto/2.8.119 Version/11.10
```

A classe representa um módulo de uma operação mais complexa, da qual a classe é somente uma única parte. Como toda boa classe em POO, ela tem uma única responsabilidade – encontrar informações sobre o agente de usuário.

Funções construtoras em PHP

Uma característica única de classes PHP está no uso da instrução `__construct()` como *função construtora*. A maioria das linguagens de computador utiliza o nome da classe como nome da função construtora; no entanto, o uso da instrução `__construct()` elimina qualquer dúvida acerca do propósito da função.

A função construtora em uma classe é iniciada automaticamente, assim que a classe for instanciada. Na classe `TellAll`, os resultados serão apresentados imediatamente na tela, não importa se você os queira lá ou não. Se o objetivo for fazer uma demonstração, não há problemas, mas enquanto módulo, outros módulos poderão simplesmente querer usar as informações sobre o dispositivo e/ou o navegador. Sendo assim, como você verá, nem todas as classes incluem uma função construtora.

O Client como uma classe solicitante

Na classe `TellAll`, incluí um pequeno gatilho no final para iniciar a classe. Com exceção da classe `Client`, a autoiniciação não é recomendada. Em sua experiência com PHP, é mais provável que você tenha iniciado um programa PHP a partir de HTML usando uma tag `form`, de modo semelhante a:

```
<form action="dataMonster.php" method="post">
```

Portanto, você deve estar familiarizado com a inicialização de um arquivo PHP a partir de uma fonte externa. De modo similar, arquivos PHP contendo classes devem ser usados por outros módulos (classes) e não devem ser autoiniciados.

À medida que avançarmos nos padrões de projeto, você perceberá que uma classe chamada `Client` aparecerá constantemente. O `Client` tem diferentes papéis em projetos maiores, mas o papel principal é fazer solicitações a partir de classes que constituem o padrão de projeto. Nesse caso, o `Client` é mostrado em relação a uma versão revisada da classe `TellAll`. Essa nova classe usada por `Client` é diferente de `TellAll` em vários aspectos, de modo que ela é mais útil para um projeto em geral e é reutilizável em outros projetos. A classe `MobileSniffer` começa com as mesmas informações sobre agente de usuário, mas a classe as disponibiliza de maneiras mais produtivas por meio de suas propriedades e métodos. Ao usar um diagrama UML (Unified Modeling Language), você poderá ver que `Client` instancia (seta tracejada) `MobileSniffer`. A figura 1.2 ilustra um diagrama de classes simples, representando as duas classes.

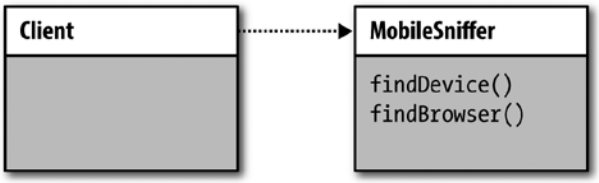

Figura 1.2 – A classe Client instancia a classe MobileSniffer e pode usar suas propriedades e seus métodos.

Se uma classe `Client` tivesse sido instanciada automaticamente, o `Client` teria menos opções para usar as informações de `MobileSniffer`. Dê uma olhada na listagem a seguir para ver como criar a classe:

```
<?php
//Agente de usuário como propriedade do objeto
class MobileSniffer {
    private $userAgent;
    private $device;
    private $browser;
    private $deviceLength;
    private $browserLength;

    public function __construct() {
        $this->userAgent=$_SERVER['HTTP_USER_AGENT'];
        $this->userAgent=strtolower($this->userAgent);

        $this->device=array('iphone','ipad','android','silk','blackberry', 'touch');
        $this->browser= array('firefox','chrome','opera','msie','safari',
            'blackberry','trident');
        $this->deviceLength=count($this->device);
        $this->browserLength=count($this->browser);
    }
    public function findDevice() {
        for($uaSniff=0;$uaSniff < $this->deviceLength;$uaSniff ++) {
            if(strstr($this->userAgent,$this->device[$uaSniff])) {
                return $this->device[$uaSniff];
            }
        }
    }
    public function findBrowser() {
        for($uaSniff=0;$uaSniff < $this->browserLength;$uaSniff ++) {
            if(strstr($this->userAgent,$this->browser[$uaSniff])) {
```

```
                return $this->browser[$uaSniff];
            }
        }
    }
}

?>
```

Habilitando relatório de erros no arquivo php.ini

Trabalho em um ambiente universitário, no qual os administradores do sistema, em geral, são estudantes (com variados níveis de conhecimento e de competência), ainda aprimorando seu ofício. Com frequência, eles se esquecem de configurar o arquivo *php.ini* para informar erros. Como resultado, adquiri o hábito de acrescentar as linhas a seguir no início de meu código:

```
ini_set("display_errors","1");
ERROR_REPORTING(E_ALL);
```

Para algumas pessoas, essas linhas adicionais de código são irritantes, mas eu as incluo na classe `Client` como lembrete da importância de se informar erros no desenvolvimento de aplicações, quando o feedback é essencial. O aprendizado de POO e de padrões de projeto baseia-se fortemente em tais feedbacks.

Para usar `MobileSniffer`, o `Client` instancia essa classe e usa seus métodos, conforme mostrado na listagem a seguir:

```
<?php
ini_set("display_errors","1");
ERROR_REPORTING(E_ALL);
include_once('MobileSniffer.php');
class Client {
    private $mobSniff;
    public function __construct() {
        $this->mobSniff=new MobileSniffer();
        echo "Device = " . $this->mobSniff->findDevice() . "<br/>";
        echo "Browser = " . $this->mobSniff->findBrowser() . "<br/>";
    }
}

$trigger=new Client();
?>
```

O uso da classe `Client` proporciona uma maneira de tornar a classe `MobileSniffer` mais útil. `MobileSniffer` não precisa iniciar a si mesmo, e usando uma instrução de

retorno, qualquer classe que chamar `MobileSniffer` simplesmente obterá os dados. O `Client` poderá usar esses dados da maneira que quiser. Nesse caso, `Client` formata os dados para mostrá-los na tela, como você pode ver na figura 1.3.

Figura 1.3 – O Client usa os dados de MobileSniffer para mostrá-los na tela.

Poderíamos ter formatado os dados na classe `MobileSniffer`, mas nesse caso o resultado não teria sido tão flexível ou útil. Ao permitir que `Client` use os dados da forma mais genérica possível, a classe poderá fazer o que quiser com esses dados. Por exemplo, em vez de formatar os dados para tela, o `Client` poderia ter usado os dados para chamar um arquivo CSS que formatasse para um dispositivo e/ou navegador em particular. Caso os dados tivessem sido pré-formatados na classe `MobileSniffer`, usá-los para identificar um arquivo CSS exigiria a remoção da formatação desnecessária. Lembre-se que uma das características mais importantes dos padrões de projeto é a reutilização dos objetos.

Capturar dispositivos móveis: missão impossível

Na época em que este livro foi escrito, a quantidade e os tipos de dispositivos móveis estavam aumentando constantemente, e qualquer código que você escrevesse em PHP estava fadado a deixar alguns dispositivos e/ou navegadores de fora. Até mesmo capturar o dispositivo pode não ser suficiente porque alguns (como o iPad e o iPad Mini) possuem diferentes resoluções de tela, além de terem tamanhos de tela diferentes. É suficiente dizer que se você planeja criar páginas de web a serem visualizadas em dispositivos diferentes, você deve ter um módulo em seu sistema que possa ser atualizado sem que ocorram erros em seu programa. Portanto, não importa qual seja a técnica mais recente para detectar e responder a múltiplos dispositivos, esteja preparado para mudanças. Você pode se planejar para começar tudo de novo, de modo a incorporar novos dispositivos, como o Surface da Microsoft, ou poderá estar preparado com um módulo que poderá ser incorporado em um aplicativo existente, no qual as alterações não provocarão falhas no sistema.

A essa altura, você poderá estar pensando: "Eu poderia escrever um algoritmo melhor para resolver o problema dos dispositivos e navegadores". Provavelmente poderia, e de fato é provável que você terá de fazê-lo porque, à medida que novos dispositivos e navegadores são introduzidos, eles deverão ser incorporados em um programa que necessite usar informações sobre dispositivos/navegadores. Contudo, se você preservar a estrutura de dois métodos, `findDevice()` e `findBrowser()`, será possível fazer todas as alterações e melhorias que quiser e o programa maior não terá falhas. Você deve pensar em um programa muito maior e muito mais complexo, e pensar em como fazer alterações. Se já teve de revisar um programa maior, você sabe que uma alteração pode se insinuar por todo o programa e provocar erros. E então sua depuração se tornará um pesadelo. Uma das principais funções da POO e dos padrões de projeto está na capacidade de alterar um módulo sem provocar erros em todo o programa.

E a velocidade?

Praticamente todo programador quer que um programa execute com uma velocidade ótima e, para isso, procuramos os melhores algoritmos. Porém, por enquanto, devemos focar nossa atenção em outro tipo de velocidade – na quantidade de tempo necessária para criação e atualização de um programa. Se um programa executa uma operação em ciclos, a qual acontece 100 milhões de vezes, ajustes pequenos em velocidade nessa operação são importantes; no entanto, tentar ganhar alguns microssegundos em uma operação que é executada somente uma vez pode representar um custo alto em relação ao uso de tempo. Da mesma maneira, ter de revisar todo um programa porque algumas linhas de código foram adicionadas, representa, igualmente, um alto custo em relação ao uso de tempo.

Velocidade de desenvolvimento e de alterações

Considere um contrato em que você tenha de atualizar e manter um programa para um cliente. Você negociou um determinado valor para efetuar atualizações constantes e quer, ao mesmo tempo, satisfazer o cliente e gastar uma quantidade de tempo justa, porém limitada, nas atualizações. Por exemplo, suponha que seu cliente faça vendas semanais de diferentes produtos, o que exige atualizações constantes de texto e de imagens. Uma solução poderá ser configurar uma atualização semanal usando a função `time()` e, então, tudo o que você terá de fazer será adicionar a URL da imagem e o texto mais atualizados em um banco de dados. Com efeito, se tivesse o texto e as imagens com antecedência, você poderia sair

de férias e deixar o PHP fazer o trabalho enquanto estivesse ausente. Seria um excelente negócio de manutenção e você poderia manter vários clientes satisfeitos simultaneamente.

Você pensaria em criar um sistema de manutenção em que tivesse de reescrever o programa toda vez que fosse necessário fazer alterações? Provavelmente não. Seria uma maneira muito lenta e cara de realizar tarefas. Portanto, nas situações em que a velocidade da revisão é importante, seu programa deve levar em conta a velocidade tanto da operação quanto do desenvolvimento. Os algoritmos lidam com a velocidade das operações e os padrões de projeto lidam com a velocidade do desenvolvimento.

Velocidade das equipes

Outro problema de velocidade pode ser encontrado no trabalho com equipes. Ao lidar com programas maiores e mais complexos, as equipes devem estar de acordo e compreender que há um plano e um objetivo comuns para que possam criar e manter programas de grande porte de modo efetivo e eficiente. Entre outras vantagens, a POO e os padrões de projeto oferecem uma linguagem comum para agilizar o trabalho em equipe. Referências a "factories", "máquinas de estado" e "observers" significam o mesmo para aqueles que entendem de POO e de padrões de projeto.

Acima de tudo, os padrões de projeto oferecem uma maneira de programar, de modo que uma equipe de programadores possa trabalhar em partes separadas que funcionarão em conjunto. Pense em uma linha de produção de automóveis – cada equipe monta uma parte diferente do carro. Para isso, as pessoas precisam de um padrão de desenvolvimento e devem compreender a relação que existe entre as partes. Dessa maneira, todos podem fazer seu trabalho, sabendo que o trabalho de outra pessoa complementará o seu. Não é necessário conhecer os detalhes do trabalho de outros funcionários. Eles só precisam saber que estão trabalhando de acordo com o mesmo plano.

O que há de errado com programação sequencial e procedural?

"Se não estiver quebrado, não conserte" é uma ideia amplamente preconizada e você poderá concordar imediatamente com ela se uma solução funcionar. No entanto, um pensamento como esse é a antítese do progresso e das melhorias. Afinal de contas, para ir de um local para outro, caminhar funciona muito bem. Porém,

para ir de um lado para outro do país, voar em um jato funciona muito melhor. A POO e os padrões de projeto são melhorias em relação à programação sequencial e procedural, da mesma maneira que voar é uma melhoria em relação a caminhar.

Programação sequencial

A maioria dos programadores começa a fazer programação escrevendo uma instrução após a outra para criar uma série de linhas que executarão um programa. Por exemplo, o código a seguir é um programa PHP sequencial perfeitamente funcional e adequado:

```
<?php

$firstNumber=20;
$secondNumber=40;
$total= $firstNumber + $secondNumber;
echo $total;

?>
```

As variáveis são de tipos de dados abstratos e o operador aritmético de adição (+) combina os valores de duas variáveis em uma terceira variável. A instrução `echo` mostra na tela o total referente aos valores combinados.

A adição de dois números é um problema simples para o PHP e, desde que você lide com problemas simples, poderá usar soluções simples.

Programação procedural

À medida que os programadores começaram a escrever programas cada vez mais longos, com tarefas mais complexas, as sequências começaram a ficar confusas, o que resultava em um código chamado de "macarrônico". Uma instrução `GOTO` permitia aos programadores sequenciais fazer saltos em um programa para completar um procedimento, o que facilitava a criação de um código emaranhado.

Com a programação procedural, surgiram as funções. Uma *função* é um pequeno objeto em que uma operação pode ser chamada para executar uma sequência por meio de uma única instrução. Por exemplo, o código a seguir é uma versão procedural do programa sequencial apresentado na listagem anterior:

```
<?php

function addEmUp($first,$second) {
   $total=$first + $second;
```

```
    echo $total;
}
addEmUp(20,40);

?>
```

As funções (ou *procedimentos*) permitem aos programadores agrupar sequências em módulos que podem ser reutilizados em um programa. Além do mais, pelo fato de ter parâmetros, o programador pode inserir diferentes argumentos em uma função, de modo que ela pode ser usada com diferentes valores concretos.

Assim como a POO, a programação procedural usa modularidade e reuso. No entanto, a programação procedural não oferece classes, nas quais as tarefas de programação podem ser agrupadas em objetos. Objetos de classes (instâncias de classes) podem operar com suas próprias estruturas de dados e isso não pode ser feito apenas pelas funções. Como resultado, a programação procedural exige sequências longas para executar tarefas maiores. Além do mais, trabalhar em equipes é mais difícil com a programação procedural porque diferentes membros da equipe não podem trabalhar facilmente em classes independentes, porém inter--relacionadas, do mesmo modo que pode ser feito com POO.

Pague agora ou pague depois

Há algum tempo, publiquei uma postagem de blog intitulada "No Time for OOP and Design Patterns" ("Sem tempo para POO e padrões de projeto"), em *http://bit.ly/108IFSF*. A postagem foi uma reação a vários desenvolvedores que disseram, com razão, que não tinham tempo para incorporar POO ou padrões de projeto em seus trabalhos, apesar do desejo de fazê-lo. Eles explicaram que um projeto surgia com um prazo final definido claramente e, em um esforço para cumprir o prazo, eles remendavam um programa que estava funcionando, usando programação sequencial e procedural. Talvez incluíssem uma ou duas classes, caso tivessem uma que correspondesse a um objetivo em particular, mas era só.

Ao aprender POO e padrões de projeto para PHP, você deve se lembrar de dois pontos, inicialmente levantados por Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides em *Padrões de projeto: soluções reutilizáveis de software orientado a objetos*:

- fazer projeto de software orientado a objetos é difícil;
- fazer projeto de software orientado a objetos que seja reutilizável é mais difícil ainda.

Em vez de olhar para essas afirmações como motivos para *não* aprender POO e padrões de projeto, elas se apresentam como razões pelas quais a POO e os padrões de projeto são tão valiosos. O conhecimento agrega valor a qualquer conjunto de habilidades. Quanto mais difícil for obter o conhecimento, mais ele será valorizado.

Não espere entender POO e padrões de projeto fácil e rapidamente. Em vez disso, incorpore-os pouco a pouco em sua programação PHP. Em algum momento você perceberá o valor. Ao longo do tempo, você desenvolverá mais habilidades e terá mais compreensão, e se deparará com um projeto no qual poderá reutilizar a maior parte das estruturas de programa de projetos anteriores. Em um projeto recente decidi usar um padrão de projeto Strategy. O padrão incluía uma tabela com 105 campos e o cliente queria um determinado conjunto de funcionalidades. Ao usar um padrão Strategy, cada uma das estratégias era uma classe com um algoritmo para lidar com um problema razoavelmente comum em PHP – filtrar, atualizar e apagar dados em um banco de dados MySQL. Levou um tempo para criá-lo, mas uma vez configurado, alterá-lo era fácil (clientes sempre querem alterações!). Algum tempo depois, fui solicitado a desenvolver um tipo semelhante de projeto usando frontend e backend em PHP, com um banco de dados MySQL. Em vez de começar do zero, simplesmente peguei o padrão Strategy, alterei os algoritmos e ele estava funcionando de imediato. Recebi o mesmo pagamento, mas tendo trabalhado de forma inteligente, meu cliente recebeu um software muito melhor do que receberia, caso eu tivesse trabalhado por mais tempo, de modo menos inteligente.

Em algum momento, temos que acabar com nossos velhos hábitos e melhorar nossas habilidades. Nesse momento, muitos programadores estão melhorando suas habilidades para acomodar dispositivos móveis. Caso não o façam, eles perderão muitas oportunidades – em algum momento, suas habilidades poderão se tornar obsoletas. Ao longo do tempo, sabemos que devemos melhorar nossas habilidades para incorporar os benefícios da última versão de PHP, as novas tecnologias ou a direção tomada pelo mercado.

POO e padrões de projeto contêm conceitos que transcendem todas essas mudanças, nos tornam programadores melhores e permitem que nossos clientes recebam softwares melhores. Tudo começa com um passo inicial. Ao investir tempo agora, você não ficará brigando com prazos para terminar um projeto no futuro. Além disso, você sairá do processo como um programador melhor e somente esse aspecto é motivo suficiente para aprender POO e padrões de projeto.

Acima de tudo, aprender POO e padrões de projeto representa o prazer de fazer algo bem-feito.