BOAS PRÁTICAS DE PROGRAMAÇÃO

Sumário

Codifique em inglês 3
Nomeação verbo-substantivo (*Verb-Noun Naming*) 3
Indentação consistente 3
Evite comentários óbvios 5
Agrupamento de código 5
Esquema consistente de nomeação 6
Princípio DRY 6
Minimize o aninhamento 6
Limite o tamanho da linha 7
Organização de arquivos e pastas 8
Leia e estude código *open source* 8
Referências 8

Codifique em inglês

Quando estamos codificando, devemos sempre ter em mente que estamos escrevendo para os outros. Atualmente, com o mundo *open source* amplamente difundido e as comunidades de desenvolvedores sempre disponíveis para ajudar, estamos escrevendo código não apenas para nossos colegas de projeto ou de empresa, mas para todos os desenvolvedores do mundo.

Tendo isso em vista, eu recomendo fortemente que você escreva seu código em **inglês**. E se você tem dificuldade com este idioma, corra atrás de aprender, senão sugiro mudar de área, pois computação não é para você.

Nomeação verbo-substantivo (*Verb-Noun Naming*)

Nomear é um daqueles problemas que assombram desenvolvedores desde os anos 1990's. Porém, existem algumas dicas que ajudam os desenvolvedores a mitigar o problema de nomeação. Um que tem funcionado bem para muitos desenvolvedores é a convenção "verbo-substantivo".

Quando for nomear um método, descreva a ação que ele executa com um verbo (get, set, transform, render). Em seguida adicione a descrição do valor que o método irá retornar ou atualizar, como getInvestmentTotal(...args), renderHeader(...args) ou saveUserAvatar(...args).

De forma análoga, todas as variáveis, estruturas e classes que você criar devem ser nomeadas através da descrição do que eles representam. Uma vez que esses elementos são em geral representações do mundo real ("coisas") que serão manipuladas pelos métodos e funções existentes, nada mais lógico do que usarmos substantivos para descrevê-las. Logo, uma classe que possui um método "render" pode ser nomeada como "Renderer", e uma variável que aponta para um objeto da classe "Renderer" pode ser nomeada como "renderer".

Essa convenção verbo-substantivo ajudará outros desenvolvedores a melhor entender o código que você escreveu.

Indentação consistente

Assumo que você já sabe que tem que indentar o seu código. Porém, é importante que você o faça de forma consistente. Existem diversos estilos de indentação, como os três apresentados abaixo.

Style 1:

```
function foo() {
    if ($maybe) {
        do_it_now();
        again();
    } else {
        abort_mission();
    }
    finalize();
}
```

Style 2:

```
function foo()
{
    if ($maybe)
    {
        do_it_now();
        again();
    }
    else
    {
        abort_mission();
    }
    finalize();
}
```

Style 3:

```
function foo()
{   if ($maybe)
    {   do_it_now();
        again();
    }
    else
    {   abort_mission();
    }
    finalize();
}
```

Programadores nas linguagens Java, C e C++ costumam utilizar o estilo 1. Já programadores C# costumam utilizar o estilo 2. Alguns programadores gostam de indentar utilizando 4 espaços, outros utilizam 8 espaços, outros apenas 2 espaços, outros ainda preferem utilizar o tab. Não há um estilo melhor que os demais e que deveria ser seguido por todos os programadores. Na realidade, o melhor estilo é um estilo consistente.

Existem algumas linguagens onde a indentação indica blocos de código, como python, e para essas linguagens a escolha do estilo de indentação pode facilitar a interpretação pelos demais. Nessas linguagens, o uso de espaços costuma ser preferido em detrimento ao uso de tab.

Eu particularmente gosto de utilizar 4 espaços como indentação. Meu racional é que 8 espaços "rouba" muito espaço na linha a cada bloco aninhado, e com apenas 2 espaços, o início do bloco aninhado fica muito próximo do início do bloco que o encapsula. Por fim, o uso dos espaços evita surpresas, uma vez que você pode configurar sua IDE para converter um tab em espaços.

Se você está trabalhando em um projeto individual, escolha o estilo que mais lhe agrade e utilize-o em **todo o seu código**. Caso você esteja trabalhando com outras pessoas ou esteja contribuindo com um projeto, você deve seguir o estilo já em uso no projeto. Caso ainda não haja um, escolham um estilo que agrade a todos (ou a maioria) e exija que ele seja utilizado em todo o código produzido.

Ressalto que você não precisa utilizar um estilo pré-existente, você pode misturar estilos e definir o seu, desde que seja consistente. Mas, ao usar um estilo pré-existente, você terá uma quantidade maior de desenvolvedores acostumados ao estilo.

Aqui está um link da wikipedia sobre estilos de identação.

Evite comentários óbvios

Comentar código é uma excelente prática de programação, mas apenas se o comentário adicionar informação. Se você utilizar bem as práticas mencionadas nesse documento, seu código já será auto-explicativo, e uma boa parte dos comentários se tornará desnecessária. Veja esse exemplo:

```php
// get the country code
$country_code = get_country_code($_SERVER['REMOTE_ADDR']);

// if country code is US
if ($country_code == 'US') {

    // display the form input for state
    echo form_input_state();
}
```

Como o texto está óbvio, não é produtivo repeti-lo em comentários. Se ainda assim quiser comentar este código, você pode colocá-lo em uma única linha acima do bloco:

```php
// display state selection for US users
$country_code = get_country_code($_SERVER['REMOTE_ADDR']);
if ($country_code == 'US') {
    echo form_input_state();
}
```

Agrupamento de código

Frequentemente algumas tarefas vão demandar várias linhas de código. É uma boa prática agrupar essas tarefas em blocos de código com alguns espaços entre eles. Mas quantos espaços? Quantos você achar necessário, desde que você seja consistente.

```php
// get list of forums
$forums = array();
$r = mysql_query("SELECT id, name, description FROM forums");
while ($d = mysql_fetch_assoc($r)) {
    $forums []= $d;
}

// load the templates
load_template('header');
load_template('forum_list',$forums);
load_template('footer');
```

Adicionar um comentário no início de cada bloco de código ajuda a enfatizar a separação visual.

Esquema consistente de nomeação

Todos os nomes devem ter fronteiras. Existem duas opções populares:

- camelCase: Primeira letra de cada palavra em caixa alta, exceto na primeira palavra.
- underscores: Uso de underscore entre palavras, como: mysql_real_escape_string().

Novamente, não existe uma opção melhor que a outra, você deve escolher a que melhor lhe agrada, mas... Se o projeto já segue uma convenção, você deve seguir esta convenção. Adicionalmente, algumas linguagens tendem a utilizar determinados esquemas. Por exemplo, em Java é comum uso de camelCase, enquanto em PHP a maioria usa underscores.

Estes estilos também podem ser misturados. Alguns desenvolvedores preferem o uso de underscores para variáveis, mas camelCase para nomear métodos e classes. Novamente, o importante é ser consistente.

Porém, independente do padrão de nomeação de variáveis e métodos, existe uma convenção de nomeação praticamente unânime na programação, que é a nomeação de constantes. Praticamente todos os padrões de codificação, de quase a totalidade das linguagens de programação modernas, define que constantes devem ser nomeadas sempre em **CAIXA ALTA**.

Princípio DRY

DRY significa "*Don't Repeat Yourself*" (Não repita a si mesmo, em português). Também conhecido como "DIE: *Duplication is Evil*".

O princípio afirma que "cada porção de conhecimento em um sistema deve possuir uma representação única, de autoridade e livre de ambiguidades em todo o sistema".

O propósito da maioria das aplicações é automatizar tarefas repetitivas. O princípio deve ser mantido em todo o código, até em aplicações web. Um mesmo trecho de código não deve ser repetido várias vezes.

Minimize o aninhamento

Quando temos vários níveis de aninhamento, pioramos a legibilidade de nosso código, tornando-o mais difícil de ler e analisar.

```php
function do_stuff() {

// ...

    if (is_writable($folder)) {

        if ($fp = fopen($file_path,'w')) {

            if ($stuff = get_some_stuff()) {

                if (fwrite($fp,$stuff)) {

                    // ...

                } else {
                    return false;
                }
            } else {
                return false;
            }
        } else {
            return false;
        }
    } else {
        return false;
    }
}
```

Para fins de usabilidade, é sempre possível fazer mudanças no código para reduzir a quantidade de aninhamentos:

```php
function do_stuff() {

// ...

    if (!is_writable($folder)) {
        return false;
    }

    if (!$fp = fopen($file_path,'w')) {
        return false;
    }

    if (!$stuff = get_some_stuff()) {
        return false;
    }

    if (fwrite($fp,$stuff)) {
        // ...
    } else {
        return false;
    }
}
```

Limite o tamanho da linha

Uma boa prática de programação é evitar escrever linhas de código muito longas, pois não é confortável para nossos olhos. Além disso, se alguém tenta ler o código numa janela de terminal, é uma boa ideia reduzir o tamanho da linha a 80 caracteres. De forma geral, não são estabelecidos padrões contendo linhas com mais de 100 caracteres, e a maioria das IDEs permite que você sinalize um limite de linha.

Organização de arquivos e pastas

Tecnicamente você pode escrever uma aplicação inteira em um único arquivo. Mas tenha certeza de que ela se transformará em um pesadelo para ler e manter.

Praticamente toda linguagem de programação possui mecanismos para incluir / importar outros arquivos em sua aplicação. Há linguagens, como Java, que exigem uma estrutura de pastas e arquivos que replica a estrutura lógica da aplicação, por exemplo, toda classe tem que estar dentro de um arquivo de mesmo nome, e todo pacote deve estar em um diretório de mesmo nome, e como toda classe deve pertencer a um pacote, os arquivos que representam as classes ficam organizados dentro do diretório que representa o pacote.

Então estude os mecanismos que a sua linguagem de programação disponibiliza e organize sua aplicação em arquivos e pastas de acordo com os padrões típicos da linguagem.

Leia e estude código *open source*

Projetos *open source* são construídos por diversos desenvolvedores. Estes projetos têm que manter um alto nível de legibilidade em seus códigos de forma que as equipes possam trabalhar juntas de forma eficiente. Portanto, é sempre uma boa ideia navegar pelo código fonte desses projetos e observar o que os desenvolvedores estão fazendo tanto em termos de padrões de codificação quanto em idiomas da linguagem específica.

Referências

1. https://code.tutsplus.com/tutorials/top-15-best-practices-for-writing-super-readable-code--net-8118
2. https://x-team.com/blog/programming-best-practices/
3. https://www.topcoder.com/coding-best-practices/
4. https://en.wikipedia.org/wiki/Indentation_style