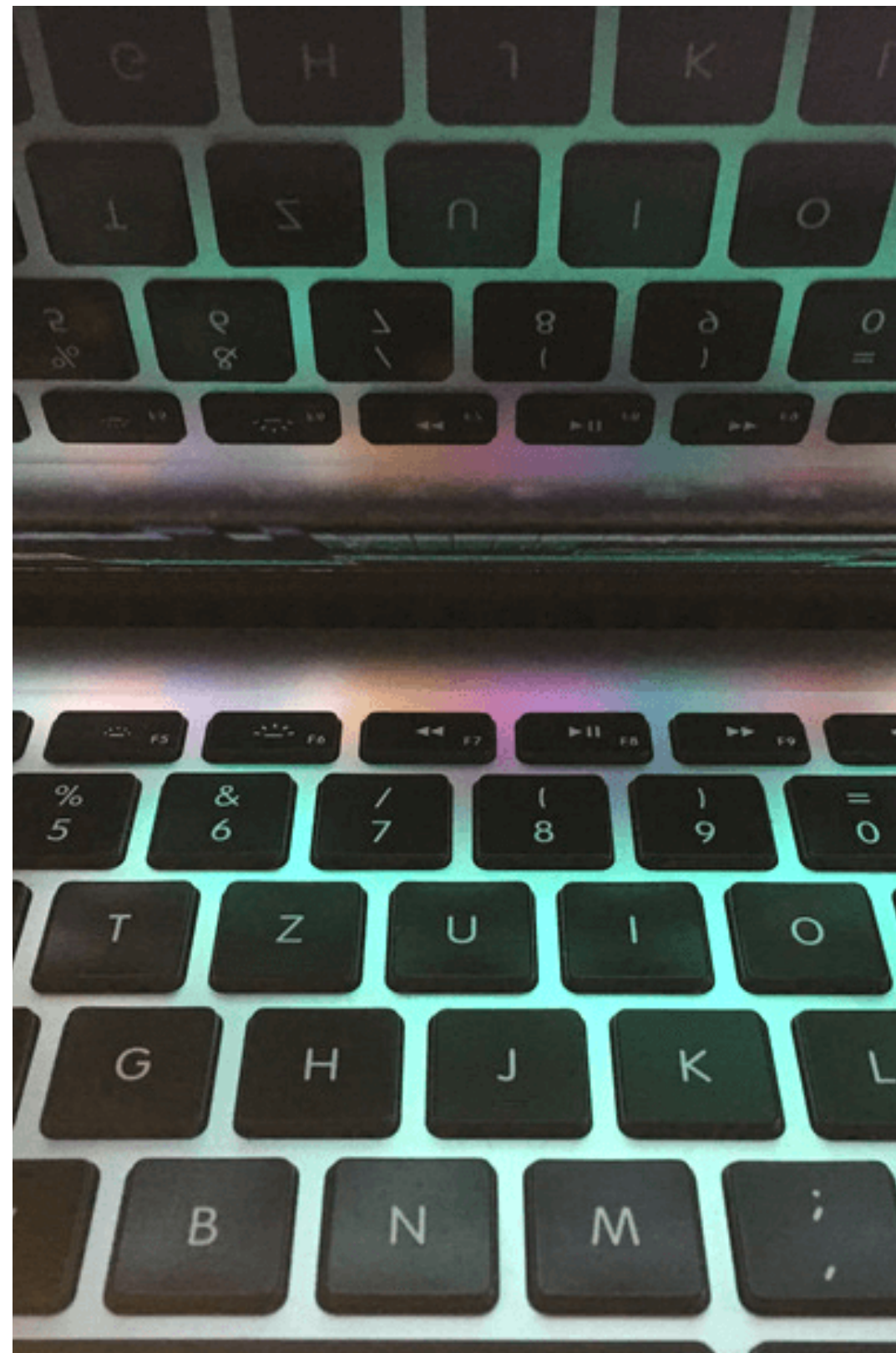


7 hábitos para uma carreira sólida em PHP

Introdução

**Você quer
construir uma
carreira na qual
os outros vejam
em você uma
referência?**



Em desenvolvimento de software, assim como em qualquer outra profissão, você pode escolher ser somente mais na multidão, ou ser uma referência.

Quando você escolhe ser referência, você assume um compromisso consigo mesmo de uma busca incessante por conhecimento, você sai da zona de conforto.

Com o passar do tempo, os profissionais ao seu redor o terão como uma referência técnica, e seu futuro profissional tende a ser bem mais sólido. Vamos discutir alguns dos principais hábitos para construir uma carreira sólida como desenvolvedor PHP?

1. ACOMPANHE A EVOLUÇÃO DA LINGUAGEM

Para quem acompanhou a evolução do PHP desde as primeiras versões, sabe que a linguagem passou por uma transformação profunda e o que conhecemos hoje pouco faz lembrar aquelas primeiras versões.

Enquanto que o PHP4 tinha uma implementação de Orientação a Objetos simplória, o PHP5 revolucionou, abrindo as portas para implementarmos aplicações corporativas complexas baseadas em Patterns.

Projetos que eram procedurais passaram a ser Orientadas a Objetos. A comunidade passou a discutir mais sobre Design Patterns, Arquiteturas de Software, e Frameworks floresceram. O PHP7 trouxe uma melhora de performance incrível e ao mesmo tempo, a maturidade da comunidade continuou em evolução, discutindo cada vez mais sobre como construir aplicações desacoplados.

Não somente a linguagem evolui, mas a comunidade em torno dela. E quais seriam os caminhos para nos mantermos em constante atualização?

Para acompanharmos a evolução da linguagem, não deixe de consultar sempre as releases e as modificações. Este é o caminho para o ChangeLog da versão 7.0 (<https://www.php.net/ChangeLog-7.php>).



1. ACOMPANHE A EVOLUÇÃO DA LINGUAGEM

Sempre que precisar migrar de uma versão para outra, consulte os manuais de migration. Eles contêm as mudanças necessárias para você trazer seu código para as últimas versões (https://www.php.net/manual/pt_BR/migration73.php).

Outra fonte riquíssima sobre certificação são os grupos de discussão. No Brasil, um dos principais é o Rumo à Certificação (<http://www.rumoacertificacaophp.com/>), acompanhe as dicas preciosas que esses grupos fornecem.

Mas para acompanhar a evolução das discussões da comunidade em torno de assuntos como Arquitetura de software, não há melhor forma do que acompanhar os eventos de PHP e publicações de profissionais reconhecidos na comunidade.

Não deixe de frequentar:

PHP Conference (<http://www.phpconference.com.br>)

PHP Experience (<https://eventos.imasters.com.br/phpexperience/>)

PHPRSCConf (<https://conference.phprs.com.br/>)

PHPSCConf (<https://conf.phpsc.com.br/>)

dentre outros.

2. Saiba fazer as coisas na mão, sem Frameworks

HOJE EM DIA É COMUM ENCONTRARMOS CADA VEZ MAIS OS "PROGRAMADORES FRAMEWORK", MAS VOCÊ SABE O QUE É ISSO?

São profissionais que apoiam-se em Frameworks de desenvolvimento, mas evitam estudar os fundamentos de sua linguagem de programação e também de Arquitetura de Software. Um "Programador Framework" não se intitula um "Programador PHP", e muitas vezes se o Framework não oferece tal recurso, ele não sabe como fazer.

Frameworks são facilitadores no Desenvolvimento de Software, mas não devemos limitar nossas escolhas simplesmente ao que eles oferecem. Antes de aprender o Framework, domine a linguagem de programação (Ex: Seja um "Programador PHP" raíz), e conheça tudo que a linguagem lhe oferece de maneira nativa (classes, funções).



O próprio PHP, por exemplo, possui dezenas de funções para manipulação de vetor (https://www.php.net/manual/pt_BR/ref.array.php) que muitos desconhecem e acabam fazendo na mão o que poderia ser resolvido com uma linha. O PHP também possui ótimas classes nativas, como a DateTime (https://www.php.net/manual/pt_BR/class.datetime.php), para manipulação de datas. Estude o manual do PHP. Conheça todo arsenal que você tem disponível para utilizar.

Busque conhecer a Orientação a Objetos. O PHP é incrível e possui recursos fantástico como métodos mágicos, uma API para reflection, classes incríveis para manipulação de XML (DOM, SimpleXML), traits, namespaces, dentre muitos outros. Além disso possui uma biblioteca (SPL) completa para manipulação de arquivos, filas, pilhas, diretórios, e arrays. E para finalizar o composer, o gestor mágico de dependências que permite você instalar milhares de pacotes prontos.

Busque compreender os padrões de arquitetura de software. Mesmo que você utilize um Framework que traga as coisas prontas para você, entenda que é possível fazer muito mais do que o seu Framework oferece. Mas para dar esse salto de conhecimento, busque estudar assuntos relacionados com arquitetura de software: Padrões de projeto, Inversão de controle, DDD, Clean Architecture, Microservices.

2. Saiba fazer as coisas na mão, sem Frameworks

3. Conheça e aplique Design Patterns

Você sabia que a maioria dos Frameworks aplica Design Patterns? Mas o que são Design Patterns? São elegantes soluções abstratas que fazem uso da Orientação a Objetos para resolver problemas comuns e que podem ser reproduzidos.

Um Design Pattern não é uma solução concreta (como uma biblioteca que possa ser baixada), mas é uma solução abstrata que resolve uma grande gama de problemas, e para alguns desses pode envolver exemplos concretos de utilização.

Qual a vantagem de aprender Design Patterns? Em primeiro lugar, você evolui seu vocabulário, você passa a se comunicar dentro da comunidade de Software utilizando termos mais refinados, que carregam um significado maior.

Mas essa seria a única vantagem? Comunicação? Na verdade não. A principal vantagem é encontrar um catálogo com um verdadeiro arsenal de soluções elegantes para resolver problemas pelos quais você certamente passará. Imagine que você precise utilizar uma biblioteca externa. Sabia que você não deve utilizar essa classe diretamente? Pois ao utilizar ela diretamente, você cria um vínculo forte e dificulta a substituição futura.



3. Conheça e aplique Design Patterns

Mas quais Patterns podem ajudar neste desafio? Você pode por exemplo utilizar um Facade e criar uma fachada para "esconder" essa biblioteca de sua aplicação. Assim, terá apenas um ponto de contato para manutenção.

Você possui uma classe que está ficando muito grande e que possui diferentes algoritmos que você troca em tempo de execução? Separe esses algoritmos e conecte-os em tempo de execução utilizando o Strategy Pattern.

Você possui uma classe que você precisa "aumentar" as funcionalidades, adicionando comportamento, mas não deseja inflar ela em tamanho? Talvez você possa utilizar o Pattern Decorator, criando uma "camada" sobre a classe original.

Padrões de projeto oferecem soluções elegantes e criativas para problemas que todos programadores já passaram ou ainda vão passar? É impressionante como desenvolvedores reconhecem os padrões ao estudá-los.

Mas como eu me especializo em Patterns? Existem 2 livros fundamentais: "Design Patterns: Elements of Reusable Object-Oriented Software" (Erich Gamma), e "Patterns of Enterprise Application Architecture" (Martin Fowler).

No início do desenvolvimento Web, poucos se preocupavam com Arquitetura de Software, a maioria só desejava "conseguir" criar uma solução para a Web. Nos anos 2000, conceitos como a arquitetura MVC, que surgiu no Desktop migrou para a Web, implementada em Frameworks famosos como o Ruby on Rails.

Frameworks para PHP eclodiram nos anos 2000. A maioria dos desenvolvedores utilizou extensamente esse modelo arquitetural, sendo esta a única escolha disponível. Atualmente, tem-se uma visão mais abrangente de desenvolvimento de software, e nem todos problemas se enquadram no modelo MVC.

Padrões de arquitetura novos foram propostos como Clean Architecture, Domain Driven Design, Microservices, e outros. A maioria desses padrões privilegia a criação de pequenas aplicações construídas sobre um modelo de domínio enxuto e puramente orientado a objetos.

Sobre este modelo de domínio são agregadas camadas com o fluxo de controle da aplicação, além de detalhes técnicos (bibliotecas). DDD privilegia que os modelos (muitas vezes conhecidos como módulos) não se comuniquem diretamente, mas somente por contratos muito bem definidos (ex. REST API).

Microservices privilegia que os domínios sejam isolados e possam inclusive rodar em nodos diferentes (RAM, CPU), para que cada um possa escalar diferentemente. Boas sugestões de leitura são "Domain Driven Design" (Eric Evans) e "Clean Architecture: A Craftsman's Guide to Software Structure and Design" (Robert C. Martin).

4. Compreenda os tipos de arquitetura de software

5. Estude infraestrutura em desenvolvimento

Antigamente, quando um desenvolvedor começava a trabalhar em uma empresa, alguém saía correndo para instalar o "ambiente" de desenvolvimento: Apache, PHP, PostgreSQL, Mysql e outros.

Realizar esse processo manualmente é sujeito a muitos problemas. Quem instalou pode ter configurado diretivas de configuração diferentes do ambiente de produção.

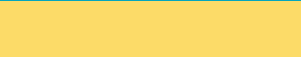
Mas o que isso causa? O desenvolvedor pode perder horas e horas procurando um problema pensando que era uma falha de programação, quando na verdade era uma diferença de configuração.

Mas como podemos minimizar este tipo de problemas? Hoje em dia existem ambientes de virtualização, correto? Então inicialmente você pode ter uma máquina virtual padronizada na empresa.

Virtual Box, VMWare são exemplos de ferramentas de virtualização. Para que você não precise manipular a máquina virtual, existe o Vagrant, que permite utilizar a VM de maneira transparente.

O Vagrant disponibiliza uma interface em linha de comandos e outras funcionalidades interessantes como redirecionamento de portas entre a máquina "convidada" e o "hospedeiro".

Quando você utilizar o Vagrant, comece a estudar sobre provisionamento, que permite você criar "receitas" genéricas de instalação de aplicações e configurações.




Ao plugar receitas no Vagrant, que podem ser escritas em Shell, Ansible, ou Chef, dentre outros, você define um roteiro de instalação e preparação para o ambiente multi-plataforma.

Agora que você está com o ambiente formado, qual o próximo passo? Quando sua aplicação estiver pronta para ser testada e colocada em produção, como você fará?

Nossa dica é definir um processo de build e deploy. O GitLab tem o maravilhoso GitLab-CI, ambiente de integração contínua, no qual você pode escrever receitas de build para testar e analisar a qualidade de seu código.

Mas e quando o build estiver pronto e você tiver que atualizar o software em 200 clientes, como fará? Conheça o Deployer, uma ferramenta de deploy em PHP, que atualizará sua aplicação em vários Hosts!



5. Estude infraestrutura em desenvolvimento

6. Conheça boas práticas e também as ruins

Quando começamos a desenvolver software, nossas expectativas são pequenas. Só desejamos que o programa funcione, não mais do que isso. Com o passar do tempo, começamos a programar com cada vez menos código-fonte, e já não conseguimos olhar com simpatia para o código antigo.

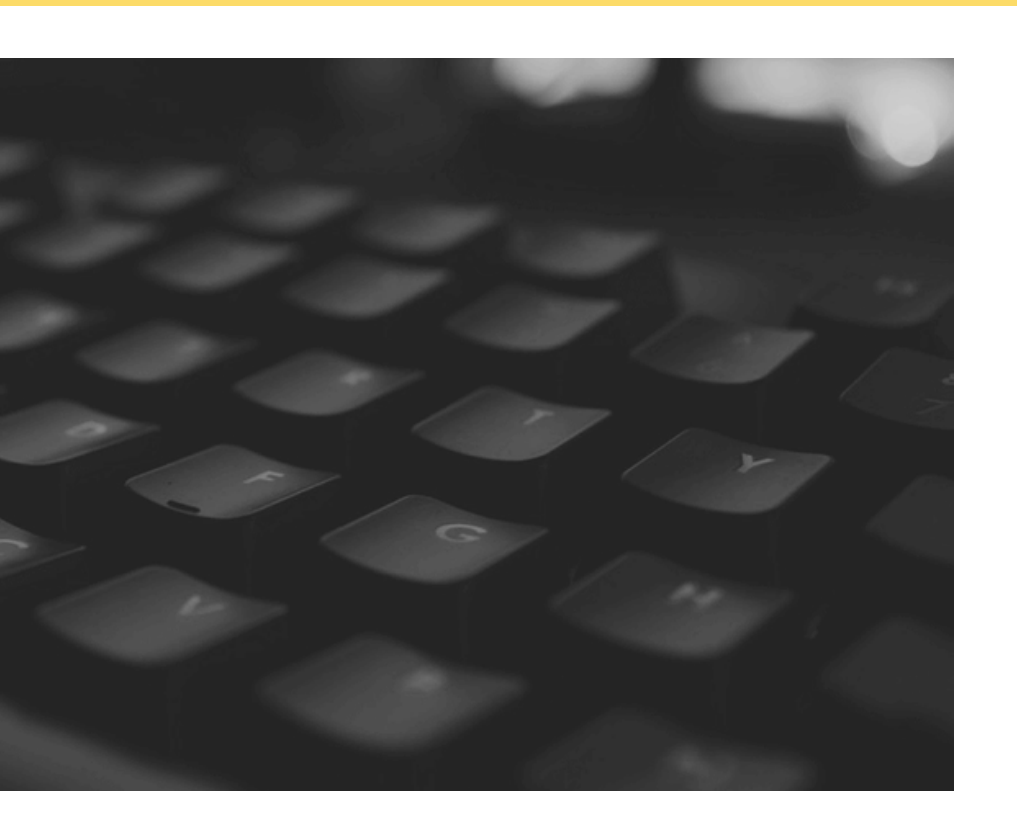
Isso é natural, essa vergonha de nossa versão passada, pois nossas técnicas vão ficando apuradas. Mas tem uma forma de acelerar esse processo? A resposta é sim. Basta identificarmos quais traços de código-fonte são considerados ruins, e por que. E para isto existem certos padrões.

Existem coisas que são consideradas ruins em código-fonte. Assim, devemos evitar fazê-las a todo custo. Quer exemplos? Vamos começar com exemplos ruins. Procure na internet sobre Programação Orientada a Gambiarras (https://desciclopedia.org/wiki/Gambi_Design_Patterns).

Estudando sobre POG, você identificará padrões ruins. O próximo passo é evitar repetir esses padrões durante o desenvolvimento do dia a dia.



6. Conheça boas práticas e também as ruins



Agora, vamos aos exemplos bons. Primeiramente, vou indicar para você uma leitura imperdível, do respeitado Robert Martin (Uncle Bob).

A obra "Clean Code: A Handbook of Agile Software Craftsmanship" apresenta uma coletânea de boas práticas para você escrever código limpo. Um bom código começa com bons nomes para classes e métodos. Nomes de classes devem transmitir apenas uma única responsabilidade, não mais.

Métodos devem fazer alguma coisa, ou retornar, jamais ambos. A quantidade de linhas de um método e de uma classe deve ser reduzido. É preferível quebrar um problema em mais classes do que ter uma classe que faz muita coisa. Estas são apenas amostras de boas práticas.

Lembre-se de nunca se repetir. Código duplicado é ruim pelo simples fato de existir, pois maior tempo será necessário para mantê-lo. Entenda que código bom é código-fonte legível, e testável. Porém para ser testável, ele precisa ter uma interface bem definida e poucas responsabilidades.

E o mais importante. Aquele código-fonte obscuro que ninguém entende, deve ser refatorado o mais rápido possível para não gerar dores de cabeça futuras.

7.

Desenvolva um projeto open source.

Você como desenvolvedor pode trabalhar sozinho, criando um produto para sua clientela. Neste caso, somente você será responsável por conhecer o código-fonte e manter sua limpeza, para o bem de sua sanidade.

Se você trabalhar em uma equipe, também criando produtos para os clientes de sua empresa, compartilhará o código com seus colegas. Quando criar um código ruim, poderá ser alvo de piadas, mas vão te entender.

Agora se você criar um projeto open source, o nível subirá e muito. Isto por que terá ao seu lado a comunidade open source, que possui um nível de exigência muito superior. Qual a vantagem de criar um projeto Open?

A primeira é ajudar os outros. Quando você cria algo que é bom para você, lembre que isto pode ser bom para outras pessoas que também possuem o mesmo perfil que você. Então a primeira virtude é fazer um bem para o próximo.

A outra vantagem de criar um projeto Open é seguir padrões. Você exigirá muito de si antes de publicar este projeto. Com certeza, você tentará fazer ao máximo que ele seja limpo, organizado. Afinal, você não quer passar vergonha.

Por fim, ao criar um projeto Open, você sairá de sua zona de conforto e dará a cara a tapa. Com isso, estará aberto à sugestões e críticas, que farão não somente o projeto evoluir, mas também você como profissional.

Nada é mais desafiador do que desenvolver para os outros. Isso nos faz escrever o software de uma maneira que os outros compreendam e possam utilizá-lo de variadas maneiras. Desenvolver open source nos faz não apenas desenvolver melhor, mas também documentar e interagir. Nos faz enxergar nossos problemas de um ângulo diferente, aprender, evoluir e atingir um estágio de maturidade superior.

php.com.br



TWITTER



FACEBOOK