# Boas e Más Práticas no Desenvolvimento Web com MVC: Resultados de Um Questionário com Profissionais

**Maurício F. Aniche**[1], **Marco A. Gerosa**[1]

[1]Universidade de São Paulo (USP)
Departamento de Ciência da Computação

{aniche, gerosa}@ime.usp.br

***Abstract.*** *Web applications are commonly built upon different technologies and architectures on both the server and client side. Among them, many applications rely on the MVC architectural pattern. As each layer is different from the other, understand their specific best practices is fundamental. In this study, we applied a questionnaire to identify and understand good and bad web software development practices according to professional developers. Based on 41 participants' answers in a questionnaire with 50 open and closed questions, we catalogued and ranked different best practices. We noticed that developers have a set of good practices that go beyond the existent MVC's architectural restrictions.*

***Resumo.*** *Aplicações web são comumente construídas sobre diferentes tecnologias e arquiteturas, tanto no servidor e do lado do cliente. Dentre elas, muitas aplicações fazem uso do padrão arquitetural MVC e, uma camada é diferente da outra, entender as boas práticas específicas de cada camada é fundamental. Neste estudo, aplicamos um questionário para identificar e compreender boas e más práticas de desenvolvimento de software web na opinião de desenvolvedores profissionais. Baseado em 41 respostas dos participantes a um questionário com 50 perguntas abertas e fechadas, percebemos que os desenvolvedores possuem um conjunto de boas práticas que vão além das restrições arquiteturais existentes no MVC.*

## 1. Introdução

Não há mais a necessidade de se argumentar a importância do desenvolvimento para web. De acordo com o *Internet Live Stats* [ils 2015], em junho de 2015, o número de sites existentes em toda a rede mundial de computadores é perto de 1 bilhão. Desenvolver aplicações web é desafiador [Ginige and Murugesan 2001]. Afinal, elas são comumente compostas por diferentes tipos de tecnologias [Ward and Kroll 1999], tanto do lado do servidor, que faz todo o processamento e persistência de dados, quanto do lado do cliente, que apresenta e interage com o usuário.

Cada uma dessas tecnologias contém seu próprio conjunto de boas práticas. Para exemplificar, há diversas boas práticas consideradas pelo mercado e academia para aplicações Java, que fazem uso de Spring MVC como arcabouço MVC, Hibernate para persistência dos dados em um banco de dados relacional e HTML, JSP e JSTL para a camada de visualização. Em um projeto como esse, o desenvolvedor possivelmente pergunta à si

mesmo: *"Qual a melhor maneira de modelar essa classe de domínio?"*, ou *"Como escrever essa consulta SQL?"*, ou mesmo *"Essa regra pertence a essa classe controladora ou aquela classe de negócio?"*.

A busca por boas práticas é fundamental. No entanto, afirmar com certeza que uma prática é realmente eficaz é difícil. Afinal, boas e más práticas são naturalmente empíricas. Na própria definição de Booch [Booch 1998], em seu conhecido trabalho *"The Rational Unified Process: an introduction"* (conhecido também por RUP), "boas práticas" são abordagens que se mostraram válidas na indústria e que, quando combinadas, atacam as raízes de problemas de desenvolvimento de software; elas são boas práticas, não tanto porque elas conseguem quantificar seu valor, mas sim porque elas são comumente usadas na indústria em projetos de sucesso. Portanto, perguntamos:

> ***Q:*** *Quais são as boas e más práticas em desenvolvimento de software web, de acordo com profissionais da indústria?*

Neste trabalho, descrevemos uma análise exploratória das opiniões de desenvolvedores sobre as melhores práticas em desenvolvimento de aplicações Web que fazem uso do padrão arquitetural MVC [Krasner et al. 1988] – que é bastante comum nos arcabouços mais populares do mercado, como Spring MVC, Asp.Net MVC ou Rails.

## 2. Trabalhos Relacionados

Na indústria, a maior parte dos trabalhos preocupa-se com boas práticas em um nível mais abstrato, discutindo boas práticas de codificação, orientação a objetos e arquitetura. Na academia, os trabalhos focam na camada de visualização, com estudos sobre HTML, CSS e Javascript. Aqui, listamos alguns deles. No entanto, até onde conhecemos, não há estudos focados nas boas práticas específicas de cada camada.

A comunidade Java possui um grande catálogo de boas práticas. O mais popular deles, conhecido por *J2EE Patterns* [Alur et al. 2003], contém padrões que aparecem frequentemente em sistemas web. Fowler [Fowler 2002] também possui um catálogo de boas práticas, dentre os quais destacamos *Domain Model, Active Record, Data Transfer Object, Front Controller e Repository*.

A própria arquitetura MVC contém um conjunto de restrições para garantir a qualidade de código produzido. Todas regras de negócio são responsabilidade da camada de Modelo. Controladores não devem contê-las; eles devem apenas controlar o fluxo entre a camada de Modelo e Visualização. A camada de Visualização é responsável apenas por exibir e interagir com o usuário final.

Na academia, em sua dissertação de mestrado, Gharachorlu [Gharachorlu 2014] avaliou algumas más práticas em código CSS em projetos de código aberto. As más práticas foram levantadas após uma análise em alguns sites e blogs famosos na área. Segundo o autor, as mais frequentes são valores fixos no código, estilos que desfazem estilos anteriores e o uso de IDs nos seletores. Esse tipo de análise pode ser feita por ferramentas que fazem a análise automática das regras de um arquivo CSS [Mesbah and Mirshokraie 2012]. Badros *et al.* [Badros et al. 1999] chegaram inclusive a trabalhar em um conjunto de restrições, de maneira a diminuir os possíveis problemas que os desenvolvedores cometem. Já no trabalho de Nederlof *et al.* [Nederlof et al. 2014], os autores investigaram como

o código HTML de 4 mil aplicações web se comportaram. Eles encontraram diversos problemas, dentre eles, mensagens de erros e avisos da W3C, IDs não únicos na página, layout quebrado e problemas de acessibilidade.

## 3. Questionário

Conduzimos um questionário com objetivo de entender e descobrir quais são as boas e as más práticas do desenvolvimento web do ponto de vista dos profissionais. O questionário, entre abertas e fechadas, contém 51 perguntas. Todas as questões visam possíveis boas e más práticas e foram agrupadas em 3 seções, uma para cada camada do MVC. A estrutura de cada seção é similar: elas começam com perguntas fechadas e terminam com perguntas abertas. As perguntas fechadas são baseadas nas más práticas conhecidas pela indústria, nas quais os participantes devem avaliar sua severidade, em uma escala de 1 a 10. Todas elas foram derivadas de buscas por boas práticas em desenvolvimento web na comunidade, bem como na experiência dos autores deste trabalho. Nas perguntas abertas, os participantes podem escrever sobre suas boas e más práticas particulares, que não apareceram nas perguntas fechadas. Ao final, o participante também tem espaço para mencionar qualquer outra ideia que ele tenha.

Enviamos o questionário para a comunidade brasileira *.NET Architects*, que tem, até a data deste artigo, por volta de 2.000 usuários, e também postamos no perfil do Twitter dos autores, que tem 2.500 seguidores. O *tweet* foi passado adiante por mais 12 pessoas. Ao final, obtivemos 41 respostas de desenvolvedores de diferentes comunidades brasileiras. Todos os 41 participantes responderam às perguntas fechadas, e desses, 21 também responderam às perguntas abertas.

Analisamos o questionário em duas etapas. Para as perguntas fechadas, descrevemos os dados por meio de estatística descritiva. Já nas perguntas abertas, usamos processo de análise qualitativa e codificação, similar ao sugerido pela Teoria Fundamentada nos Dados [Strauss and Corbin 1997]. O protocolo de análise seguiu os seguintes passos: (i) leitura cada resposta, (ii) relacionar cada sentença a uma boa ou má prática a um código, (iii) categorizar, contabilizar e agrupar os códigos levantados. Ao final, obtivemos 61 códigos (boas e más práticas) diferentes[1].

## 4. Perfil dos Participantes

De maneira geral, os participantes possuem experiência relevante em desenvolvimento de software e de aplicações web. 38 dos 41 participantes tem mais de 3 anos de experiência em desenvolvimento de software, e 13 tem mais de 10 anos. Apenas um trabalha há menos de um ano como desenvolvedor de software.

Em relação à experiência em desenvolvimento web, os números, apesar de menores, também mostram que os participantes são experientes. A maioria deles desenvolve aplicações web entre 3 e 10 anos; apenas 8 participantes tem menos de 3 anos de experiência. Além disso, a maioria deles já colocou entre 3 e 10 aplicações em produção, e apenas 4 participantes só colocaram uma única. Em uma auto avaliação sobre conhecimentos em boas práticas, em uma escala Likert de 1 a 10, a média foi 7. No entanto, 8 deles deram nota menor ou igual a 4 para si mesmos.

[1]Todo o questionário, bem como as 41 respostas fechadas, 21 respostas abertas e os 61 códigos finais podem ser encontrados em http://www.aniche.com.br/cbsoft-vem-2015/.

Java é a linguagem mais popular entre os participantes; 25 deles a usam no dia a dia. Em segundo lugar, C#. Poucos participantes usam PHP e Ruby. Os arcabouços mais populares são JSF, e ASP.NET MVC, seguidos de Spring MVC e VRaptor. Acreditamos que a variedade de tecnologias beneficia nosso estudo, pois dessa forma evitamos práticas que são específicas de uma tecnologia.

## 5. Resultados

Ao longo do questionário, os participantes relataram diferentes boas e más práticas. Algumas delas, inclusive, foram relatadas por mais de um participante. Curiosamente, alguns participantes preferiram citar a má prática, enquanto outros preferiram destacar a boa.

Na Tabela 1, apresentamos as avaliações dadas por cada participante nas boas e más práticas das perguntas fechadas do questionário. Ela está em ordem de severidade (a mais severa primeiro). É possível notar que algumas más práticas já conhecidas na literatura são realmente problemáticas do ponto de vista do desenvolvedor, como a existência de regras de negócio fora da camada de modelo e a mistura de infraestrutura com código de domínio, enquanto outras são menos graves, como o caso da falta de injeção de dependência nos controladores, ou o uso de Javascript puro, sem arcabouços.

Tabela 1. Más práticas e seu nível de criticidade, de acordo com os participantes.

| **Camada** | **Má prática** | **Mediana** | **8 a 10** | **6 a 8** | **<= 6** | **<= 2** |
|---|---|---|---|---|---|---|
| **Modelo** | Regras de negócio em DAOs | 10.0 | 65% | 15% | 19% | 2% |
| | Mistura de Regras de Negócio e Infraestrutura | 9.0 | 50% | 30% | 19% | 4% |
| | Complexidade | 8.0 | 51% | 34% | 19% | 7% |
| | Complexidade em DAOs | 8.0 | 43% | 29% | 26% | 2% |
| | Método ambíguo em DAOs | 7.0 | 14% | 48% | 36% | 4% |
| **Controlador** | Regras de negócio | 9.0 | 53% | 19% | 26% | 2% |
| | Complexidade | 8.0 | 41% | 29% | 29% | 4% |
| | Muitas rotas | 6.0 | 17% | 29% | 53% | 21% |
| | Falta de arcabouço de DI | 5.0 | 19% | 19% | 6% | 21% |
| **Visualização** | Regras de negócio | 10.0 | 68% | 17% | 14% | 2% |
| | CSS *inline* | 8.0 | 26% | 43% | 29% | 4% |
| | JavaScript arquivo único | 8.0 | 46% | 17% | 36% | 9% |
| | CSS em arquivo único | 6.0 | 24% | 19% | 56% | 31% |
| | Regras de visualização em scriptlets | 5.0 | 17% | 14% | 65% | 17% |
| | Longa Hierarquia em CSS | 5.0 | 12% | 24% | 53% | 17% |
| | JS sem *prototype* | 5.0 | 7% | 19% | 7% | 19% |
| | JS sem arcabouços | 4.0 | 9% | 12% | 78% | 29% |

Nas subseções abaixo, apresentamos a análise do estudo qualitativo, feito em cima das 21 respostas abertas que obtivemos. Nelas, os participantes nos disseram suas particularidades nas suas boas e más práticas.

### 5.1. M - Modelo

A mistura de código de infraestrutura com código de domínio foi a má prática mais comentada entre os participantes. 10 deles explicitamente disseram acreditar que isso é um grave problema. Um dos participantes disse, traduzido do inglês: *"Em minha experiência, o grande problema são quando as classes domínio estão muito acopladas com infraestruturas externas, como ORMs. Isso faz com o que o modelo reflita os requisitos estabelecidos por essas dependências e, ao final, acabamos com um projeto de classes pobre. De vez em quando, isso é uma troca que os desenvolvedores fazem para ganhar produtividade e diminuir custos, mas normalmente isso causa muitos problemas no futuro."*. Um participante em particular disse que, do seu ponto de vista, uma boa prática e solução para o problema seria fazer com que as classes de modelo façam uso de um conjunto de abstrações que representem a infraestrutura.

Qualidade de código também apareceu como uma preocupação importante nas respostas do questionário. 3 participantes mencionaram o problema do alto acoplamento e baixa coesão; métodos longos e a falta de encapsulamento também foram mencionados. Em relação a boas práticas, três eles afiram que "o bom uso de POO" é fundamental; dois deles mencionaram o Princípio da Responsabilidade Única [Martin 2003]. O uso de abstrações, assim como imutabilidade em classes de domínio, injeção de dependências via construtor e *tiny types* [Hobbs 2007] também foram considerados boas práticas.

Em relação aos DAOs, classes que acessam dados em outras fontes, como bancos de dados, o problema mais reportado foi a existência de regras de negócios misturadas em seus códigos; 8 participantes comentaram. No entanto, opiniões diferentes também apareceram. Um deles disse que se uma regra de negócios específica melhorar a performance do sistema só por estar dentro do DAO, então isso seria válido. Dois outros desenvolvedores mencionaram que se a regra de negócios for simples, então ela também pode não ser um problema por estar dentro de um DAO. Outro disse que regras de validação de dados também são aceitáveis.

Além disso, o uso do padrão Repositório [Evans 2004], o uso de um arcabouço ou mini arcabouço para acessar os dados e a escrita de uma única consulta SQL por método também foram comentadas como boas práticas. Como más práticas, os participantes mencionaram o uso de carregamento preguiçoso (que é feito automaticamente por muitos arcabouços, como *Hibernate*) e o uso de uma "classe pai" que contém códigos genéricos para todos os DAOs do sistema.

**Resumo. Desacoplar infraestrutura de regras de negócio, usar boas práticas de orientação a objetos nas classes de domínio. Ter apenas uma única consulta por método nos DAOs, usar o padrão Repositório. Evitar DAOs genéricos e carregamento preguiçoso (*lazy loading*).**

### 5.2. C - Controlador

A má prática mais popular em controladores é também a existência de regras de negócios em seus códigos. 10 participantes escreveram sobre isso. Um deles foi bem claro sobre as responsabilidades de um controlador: *"Lógica de negócio deve ficar no modelo. Controller tem que ser o mais "burro" possível"*. No entanto, um participante mencionou que regras de negócio, como "checar se uma string é nula" ou "número é maior que X"

podem ser aceitáveis, assim como qualquer tratamento de exceção. Além disso, outro participante disse que regras para instanciar uma entidade são também válidas.

Muitos participantes mencionaram sobre controladores com mais responsabilidades do que deveriam. Dois deles afirmaram que muitas rotas em um único controlador pode ser um indicador disso. Um participante falou sobre a complexidade dos controladores, afirmando que muitas linhas de código são problemáticas. Interessantemente, outro participante disse não ligar para essa complexidade, contanto que elas não sejam regras de negócio: *"Eu sei que alguns podem discordar, mas eu tendo a colocar mais complexidade em meus controladores"*.

**Resumo. Não ter regras de negócio dentro de controladores, com exceção de tratamento de exceções, validação e regras de criação de objetos. Evitar controladores com muitas responsabilidades e rotas.**

### 5.3. V - Visualização

A existência de *scriptlets* (programação dentro de arquivos de visualização, como JSPs) com regras de negócio é a má prática mais popular na camada de visualização. Um participante mencionou que o uso de *scriptlets* são permitidos apenas para regras de visualização.

A falta de padrão e código macarrônico também apareceram como más práticas. Em termos de boas práticas, participantes mencionaram a criação de componentes de visualização para possibilitar reuso, validação do lado do cliente e um arcabouço para prover segurança de tipos entre as variáveis declaradas nesses arquivos.

Em relação a CSS, participantes deram muitas pequenas sugestões. No entanto, nenhuma delas foi dita por mais de um participante. Maus nomes, grandes hierarquias, CSS *inline*, unidades de medida absolutas e uso de ID e seletores complicados foram considerados más práticas. Como boas práticas, eles sugerem o uso de classes com nomes descritivos, evitar conflitos de especificidade de classes, não sobrescrever propriedades, usar ferramentas como LESS ou SASS, ter um CSS por JSP e separar CSS em componentes.

Em Javascript, a má prática mais popular foi o uso de variáveis globais. O uso de bibliotecas pesadas, como jQuery também foi mencionado como uma má ideia. Como boas práticas, participantes mencionaram o uso de um carregador de módulos, a separação entre Javascript e CSS e alta testabilidade. Ter um único código fonte com todo Javascript parece problemático. Participantes também acreditam que o não uso de protótipos, a maneira de simular orientação a objetos em códigos Javascript não é um problema.

**Resumo. *Scriptlets* são permitidos apenas para regras de visualização. Ter um arquivo CSS por JSP, sem longas hierarquias e sem propriedades sobrescritas. Usar um carregador de módulos Javascript e pensar em sua testabilidade.**

## 6. Discussão

Muitas das boas e más práticas levantadas pelos participantes já eram conhecidas. Afinal, várias delas dizem respeito às responsabilidades de cada camada do MVC. No entanto, acreditamos que os "casos especiais" mencionados pelos participantes são bastante valiosos. Por exemplo, controladores não podem conter regras de negócio, mas sim apenas

fluxo, o que já era sabido. No entanto, regras de validação, criação de objetos e tratamento de exceções são válidas. Além disso, eles devem ser pequenos e conter apenas poucas responsabilidades. Além disso, apesar de alguns arcabouços como ASP.NET MVC, VRaptor e Spring MVC sugerirem o uso de injeção de dependências, e outros, como Ruby on Rails e Django, não, os participantes não veem diferença de qualidade entre usar ou não.

Modelos devem conter regras de negócio, como também é dito no padrão MVC. No entanto, desenvolvedores sugerem forte uso de boas práticas de orientação a objetos e desacoplamento da infraestrutura. Já DAOs devem conter uma única consulta por método e fazer uso do padrão Repositório. DAOs devem evitar o uso de carregamento preguiçoso e a existência de regras de negócio.

Visualizações também devem evitar regras de negócio. *Scriptlets* são permitidos apenas para regras de visualização. CSS deve ser claros, conter nomes descritivos, hierarquias pequenas, e não devem se conectar com elementos pelos seus IDs. Javascript devem ser testáveis e fazer uso de um carregador de módulo. Além disso, ter um único CSS e Javascript por página é considerado uma boa prática.

## 7. Ameaças à Validade

*Poucos participantes.* Tivemos apenas 41 participantes, dos quais só 21 responderam às perguntas fechadas, o que pode ser considerado um número baixo para um estudo qualitativo. No entanto, acreditamos que a amostra selecionada aborda um perfil diversificado, e a análise qualitativa foi bastante rica. E, portanto, acreditamos que os achados são relevantes.

*Localização*. O questionário foi compartilhado apenas com a comunidade brasileira de desenvolvimento de software, e por isso podem haver algumas particularidades regionais. No entanto, como um desenvolvedor de software comumente lê conteúdo de diferentes programadores ao redor do mundo, acreditamos que o sentimento do desenvolvedor é também baseado nessas opiniões, diminuindo o possível problema da regionalização.

*Peso das opiniões*. Não demos pesos diferentes para opiniões que vieram de desenvolvedores mais experientes. Todas práticas mencionadas por qualquer participante apareceram neste estudo.

*Anedotais*. Não fizemos um experimento controlado para dizer se a prática é realmente boa ou ruim; elas refletem a opinião dos desenvolvedores. No entanto, como já discutido, boas práticas tendem a emergir de experiências anteriores desses desenvolvedores, e por isso, acreditamos elas venham de sua vivência.

## 8. Conclusão

Neste estudo, coletamos opiniões de 41 diferentes desenvolvedores sobre boas e más práticas em desenvolvimento web. Como pudemos ver, as boas práticas vão além das sugeridas pelo padrão arquitetural MVC. Cada camada tem o seu conjunto de boas e más práticas específicas.

Acreditamos que desenvolvedores web devam conhecer esse conjunto de boas e más práticas, pois elas podem ajudá-los a melhorar suas aplicações e evitar a criação de código de baixa qualidade. Apesar delas refletirem apenas o ponto de vista dos partici-

pantes, é importante para um desenvolvedor conhecer esses diferentes pontos de vista. Eles podem, junto com a equipe, decidir se as aplicarão ou não.

No entanto, ainda há trabalho a fazer. Perguntas que ainda temos em aberto: Quais más práticas são mais graves? Quais não são tão importantes? Como podemos detectar a presença dessas más práticas em aplicações web, de maneira automática?

## Referências


(2015). Internet live stats. http://www.internetlivestats.com. Acessado em 14 de março de 2015.

Alur, D., Malks, D., Crupi, J., Booch, G., and Fowler, M. (2003). *Core J2EE Patterns (Core Design Series): Best Practices and Design Strategies*. Sun Microsystems, Inc.

Badros, G. J., Borning, A., Marriott, K., and Stuckey, P. (1999). Constraint cascading style sheets for the web. In *Proceedings of the 12th annual ACM symposium on User interface software and technology*, pages 73–82. ACM.

Booch, G. (1998). Software development best practices. *The Rational Unified Process: an introduction*, pages 3–16.

Evans, E. (2004). *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional.

Fowler, M. (2002). *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co., Inc.

Gharachorlu, G. (2014). Code smells in cascading style sheets: an empirical study and a predictive model.

Ginige, A. and Murugesan, S. (2001). Web engineering: A methodology for developing scalable, maintainable web applications. *Cutter IT Journal*, 14(7):24–35.

Hobbs, D. (2007). Tiny types. http://darrenhobbs.com/2007/04/11/tiny-types/. Acessado em 14 de março de 2015.

Krasner, G. E., Pope, S. T., et al. (1988). A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming*, 1(3):26–49.

Martin, R. C. (2003). *Agile software development: principles, patterns, and practices*. Prentice Hall PTR.

Mesbah, A. and Mirshokraie, S. (2012). Automated analysis of css rules to support style maintenance. In *Software Engineering (ICSE), 2012 34th International Conference on*, pages 408–418. IEEE.

Nederlof, A., Mesbah, A., and Deursen, A. v. (2014). Software engineering for the web: the state of the practice. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 4–13. ACM.

Strauss, A. and Corbin, J. M. (1997). *Grounded theory in practice*. Sage.

Ward, S. and Kroll, P. (1999). Building web solutions with the rational unified process: Unifying the creative design process and the software engineering process. *Rational Software Corporation*.