MAC 5715 - Tópicos de P.O.O.

Anti-Padrão de Desenvolvimento: "Será um padrão?"

Diego Tarábola *tarabola@gmail.com*
Denise Goya *dhgoya@ime.usp.br*

Raízes do Problema

Projetos de desenvolvimento de software mal elaborados costumeiramente levam a soluções ineficientes. Muitas vezes, a simplicidade é deixada de lado e são adotados modelos inadequados na resolução do problema.

Na fase de manutenção desses sistemas mal projetados, descobrem-se estranhos padrões de projeto, que fazem de um tudo um pouco ou, às vezes, sequer são implementados em conformidade com a especificação do padrão.

É comum programadores que se iniciaram sob o modelo de programação estruturada ou em linguagens não orientadas a objeto não conseguirem absorver corretamente os conhecimentos básicos de OO e acabam por disseminar conceitos errados por todas as fases do projeto e desenvolvimento.

Historinha-Evidência

Esta implementação de *Singleton* [2] faz mais do que deveria! Estranho hein! Ele possui métodos para criar e salvar...

A especificação do Padrão inclui características como essas? Será um novo Padrão?

Pessoal, encontrei mais um Padrão "torto". Tem um *Composite* [2] aqui que não é composto por objetos do mesmo tipo. Será que o desenvolvedor errou o nome do Padrão? Ué, cadê a composição dos objetos deste Padrão?

Sintomas e Conseqüências

Muitas vezes, quando é preciso fazer manutenção no código, existe um esforço muito grande para se entender o que aquele suposto padrão faz realmente. Somente o dono daquele código pode explicar sua finalidade.

Na tentativa de se compreender seu funcionamento, as dúvidas são constantes. É preciso recorrer a desenvolvedores mais experientes para desvendá-lo.

A documentação (quando existe) é precária e às vezes relata a implementação de determinados padrões de projeto, cuja especificação não corresponde ao código.

Podemos identificar três tipos básicos de padrões "descaracterizados" quando implementados:

- O padrão original é consideravelmente expandido, para agregar novas funcionalidades. Cria-se uma espécie de "*padrão-faz-tudo*". Em alguns casos, fica caracterizada a ocorrência do anti-padrão *Swiss Army Knife* [1], em que a interface

é grande demais, possivelmente antecipando todos os possíveis usos futuros.

- Parte da especificação do padrão original parece ser implementada corretamente; outra parte, nem tanto. Paira a dúvida para quem lê o código: isso funciona? Quando se opta por deixar tudo como está (pois não se tem certeza do que acontecerá se o suposto padrão for corrigido), origina-se um *Lava Flow* [1].
- A implementação do padrão é completamente equivocada. Acredita-se que quem o codificou (ou gerenciou o projeto) não tinha a menor idéia do que estava fazendo. E o padrão original fica apenas na intenção.

Causas Típicas

Uma das causas da ocorrência do "*Será um padrão?*" é a falta de conhecimento do desenvolvedor, para aplicar algum tipo de padrão (arquitetural, de projeto ou de análise) ao projeto. Por muitas vezes, esse tipo de desenvolvedor acaba por copiar e colar trechos de código de outros projetos bem sucedidos, ou trechos da própria aplicação, recaindo no anti-padrão *Cut-and-Paste Programming* [1].

Por inexperiência ou por insegurança, o programador quer demonstrar para os outros que domina a utilização de algum padrão.

Existe também falta de iniciativa para entender realmente a especificação de um padrão.

Exceções Conhecidas

Se o anti-padrão ocorre em grande escala num projeto, deseja-se apenas usar a funcionalidade do sistema e não se quer modificá-lo, pode-se criar uma única classe que o encapsula como um sistema legado.

Solução Refatorada

Abrir algum livro clássico sobre Padrões de Projeto e seguir as especificações. Identificar se o pretenso padrão realmente resolveria o problema; caso sim, corrigir a implementação. Separar as eventuais funcionalidades extras em outras classes. Caso o pressuposto padrão não resolva adequadamente o problema, encontrar nova solução, desvinculada do padrão não aplicável.

Nem sempre o modelo mais complexo é o mais indicado. Por isso, sempre refatore seu modelo simplificando-o ao máximo.

Juntamente com desenvolvedores mais experientes, modele sua aplicação. Se você for o mais experiente e mesmo assim não conseguir identificar um padrão adequado ao modelo, faça-o de maneira simples mesmo.

É preciso documentar o código implementado mesmo que o padrão seja de conhecimento de todos. Deixar em evidência se ocorrer uma "variação" do padrão.

Práticas como programação em pares, de técnicas de desenvolvimento ágil como Extreme Programming [3], podem ajudar na prevenção da ocorrência do anti-padrão.

Exemplo

Muitas vezes, durante o processo de manutenção de um projeto, deparamos-nos

com Padrões que deveriam ser Padrões, ou que ao menos seguissem sua especificação.

Tomemos como exemplo o Padrão *Composite* [2] e uma pretensa implementação desse padrão, ilustrada na Figura 1.

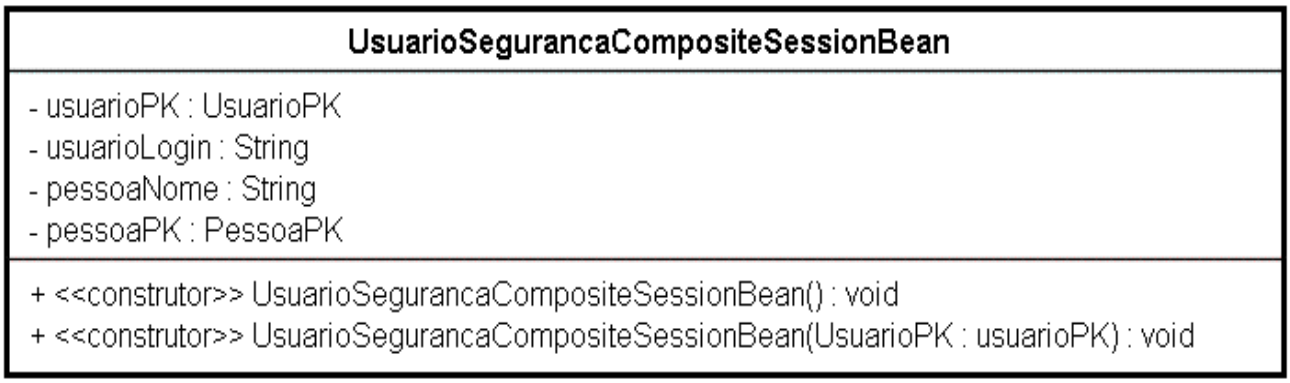

Figura 1: diagrama de classe UML do suposto Padrão *Composite*

A intenção do desenvolvedor foi utilizar outras entidades do tipo *SessionBean* para compor um objeto, mas observa-se que esta classe possui um construtor que recebe outro tipo de entidade além de não ter uma hierarquia de classes do mesmo tipo.

O "Padrão-faz-tudo" ocorre com freqüência em uma empresa onde trabalha o autor; a área-fim não é tecnologia, mas a empresa possui equipe de desenvolvedores de *software*. Lá, existe padrão que, por exemplo, cria a conexão com banco de dados e, ao mesmo tempo, cuida de validar qual usuário pode entrar no sistema.

É possível encontrar também padrões sem padrão, ou seja, padrões que não condizem com sua especificação.

Como exemplo, na Figura 2, notamos que as classes RecebeVaga e RecebeCandidato são subclasses da classe abstrata AbstractMessageCommand. Em sua especificação, descreve o emprego do padrão de projeto *Command* [2].

Recorrendo a [2], percebe-se que houve uma falha na nomeação da classe ou realmente o desenvolvedor não sabia diferenciar os padrões de projeto.

A classe AbstractMessageCommand possui um método abstrato execute(queue: Queue, parameters: Map) e outro método concreto run( );

As subclasses é que especializam (implementam) o método execute de acordo com suas necessidades. E quem invoca este método é o método run( ). Nota-se que essa especificação nos leva ao padrão de projeto *Template Method* [2].

Percebe-se que este modelo implementa o comportamento *execute* através das subclasses, podendo variar de acordo com seu tipo.

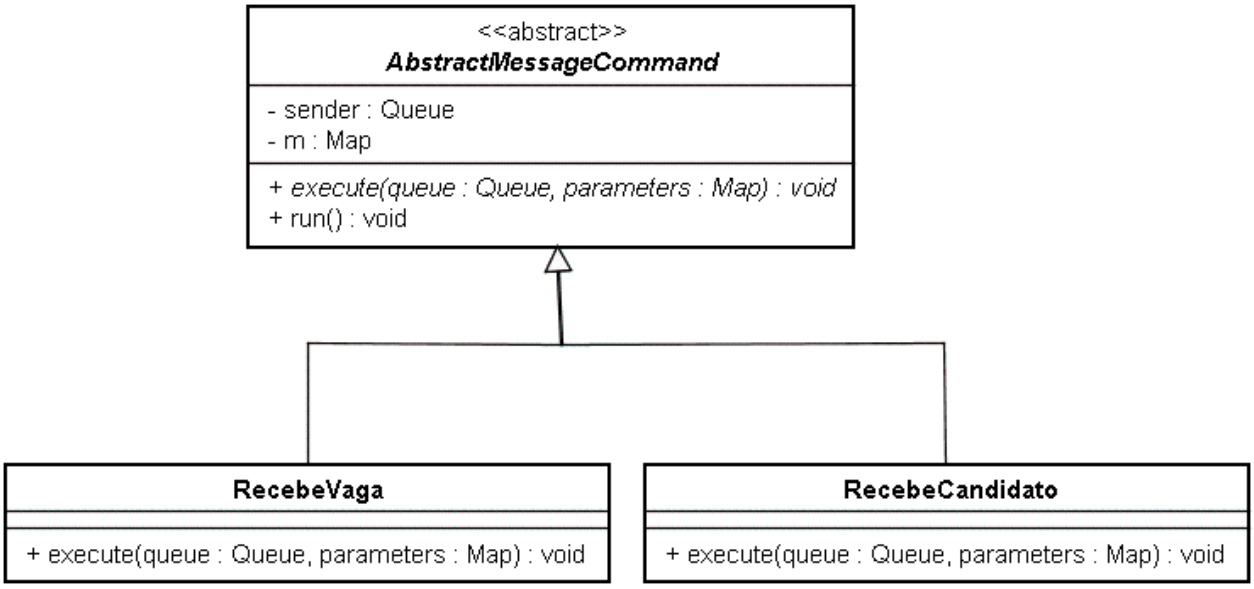

Figura 2: diagrama de classe UML do suposto Padrão *Command*

Anti-Padrões Relacionados

Cut-and-Paste Programming [1].

Lava Flow [1]

Swiss Army Knife [1]

Referências

[1] William J. Brown, Raphael C. Malveau, III Hays W. McCormick, and Thomas J. Mowbray. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. John Wiley & Sons, Inc. 1998.

[2] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.

[3] Kent Beck, Martin Fowler. *Planning Extreme Programming*. Addison Wesley, 2001.