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*

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?

Raízes do Problema

A adesão ao uso de Padrões de Projeto de Software ganhou força nos últimos dez anos, em especial após lançamento de [2]. No entanto, alguns desenvolvedores, que fazem uso de padrões, por vezes demonstram não possuir o preparo necessário para tal.

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

Sintomas

Os sintomas da ocorrência do anti-padrão "Será um padrão?" são detectados durante a manutenção de um código (Figura 1).

Quando é preciso fazer manutenção no código, existe um esforço muito grande para se entender o que uma suposta implementação de padrão faz realmente. Somente o dono daquele código pode explicar sua finalidade.

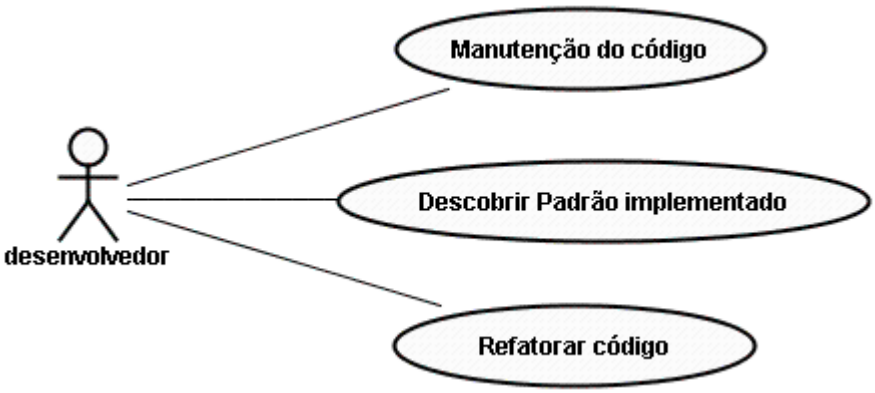

Figura 1: diagrama UML de Caso de Uso do cenário de manutenção de código

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

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

Categorias e Conseqüências

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

Tipo 1. 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.

Tipo 2. 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].

Tipo 3. 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

Por inexperiência ou por insegurança, o programador quer demonstrar para os outros que domina a utilização de algum padrão. Em alguns casos, o programador super-valoriza o uso de padrões e acaba por forçar a aplicação destes, ainda que em situações inconvenientes (optando por soluções complicadas, em vez de seguir caminhos mais simples).

Existe também falta de iniciativa para compreender corretamente a especificação de um padrão de projeto. Há momentos em que o desenvolvedor copia parte de soluções ou trechos de código de outros projetos bem sucedidos, sem entender por completo as implicações dessa "reutilização". Muitas vezes o fator tempo, dentro do contexto de

cronograma de tarefas, é o motivo principal pelo qual um código seja mal escrito e não refatorado.

Exceções Conhecidas

Quando o anti-padrão é detectado em um bloco código que possui simultaneamente as três características-chave abaixo:

1) o bloco forma um módulo coerente, com entrada e saída bem determinadas;
2) tem-se certeza do correto funcionamento desse módulo;
3) e não se deseja refatorar o módulo, devido à sua complexidade;

pode-se criar uma classe que encapsula todo o módulo, mantendo-o como um subsistema legado.

Solução Refatorada

Nos casos mais simples, o anti-padrão “Será um padrão?” ocorre simplesmente por causa de nomeações inadequadas de classes ou padrões, bastando renomeá-los ou reescrever a documentação associada.

Nas demais situações, faz-se necessária uma refatoração. A seguir, seguem passos gerais para essa refatoração.

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. Caso o pressuposto padrão não resolva adequadamente o problema, encontrar nova solução, desvinculada do padrão não aplicável.

O diagrama de caso de uso descreve este cenário, ilustrado pela Figura 2.

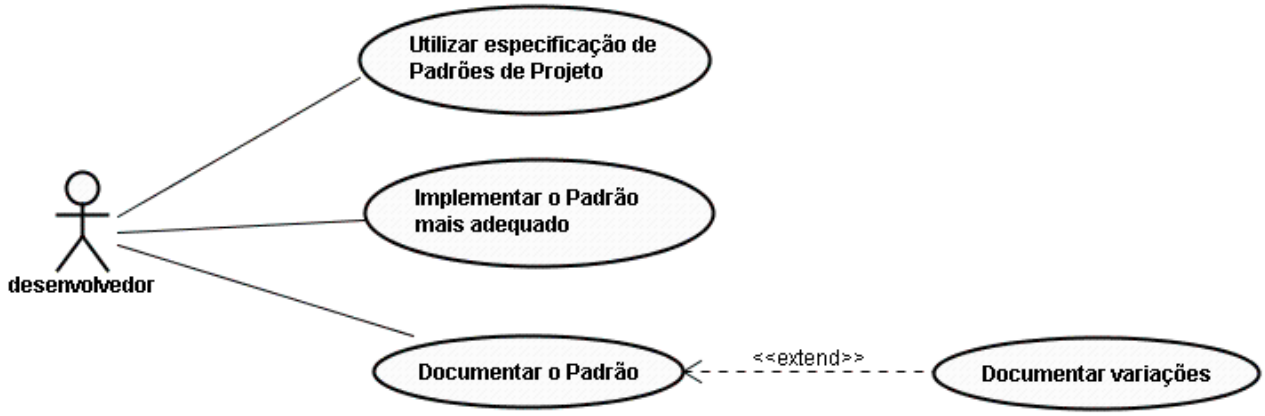

Figura 2: diagrama UML de Caso de Uso do cenário de implementação de Padrão de Projeto

Quando for identificado o anti-padrão “Será um padrão?” Tipo 1, é necessário separar as funcionalidades extras em outras classes. Se for mais conveniente, crie um novo padrão estendido, porém deixe claro qual era o padrão original e documente corretamente as extensões implementadas.

Se forem identificados o Tipo 2 ou Tipo 3, juntamente com desenvolvedores mais

experientes, modele (ou remodele) a aplicação. Se você for o mais experiente e mesmo assim se não conseguir identificar um padrão adequado ao modelo, faça-o de maneira simples mesmo.

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

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

O diagrama UML de atividades abaixo, ilustrado pela Figura 3, descreve os passos para uma possível manutenção (refatoração) do código.

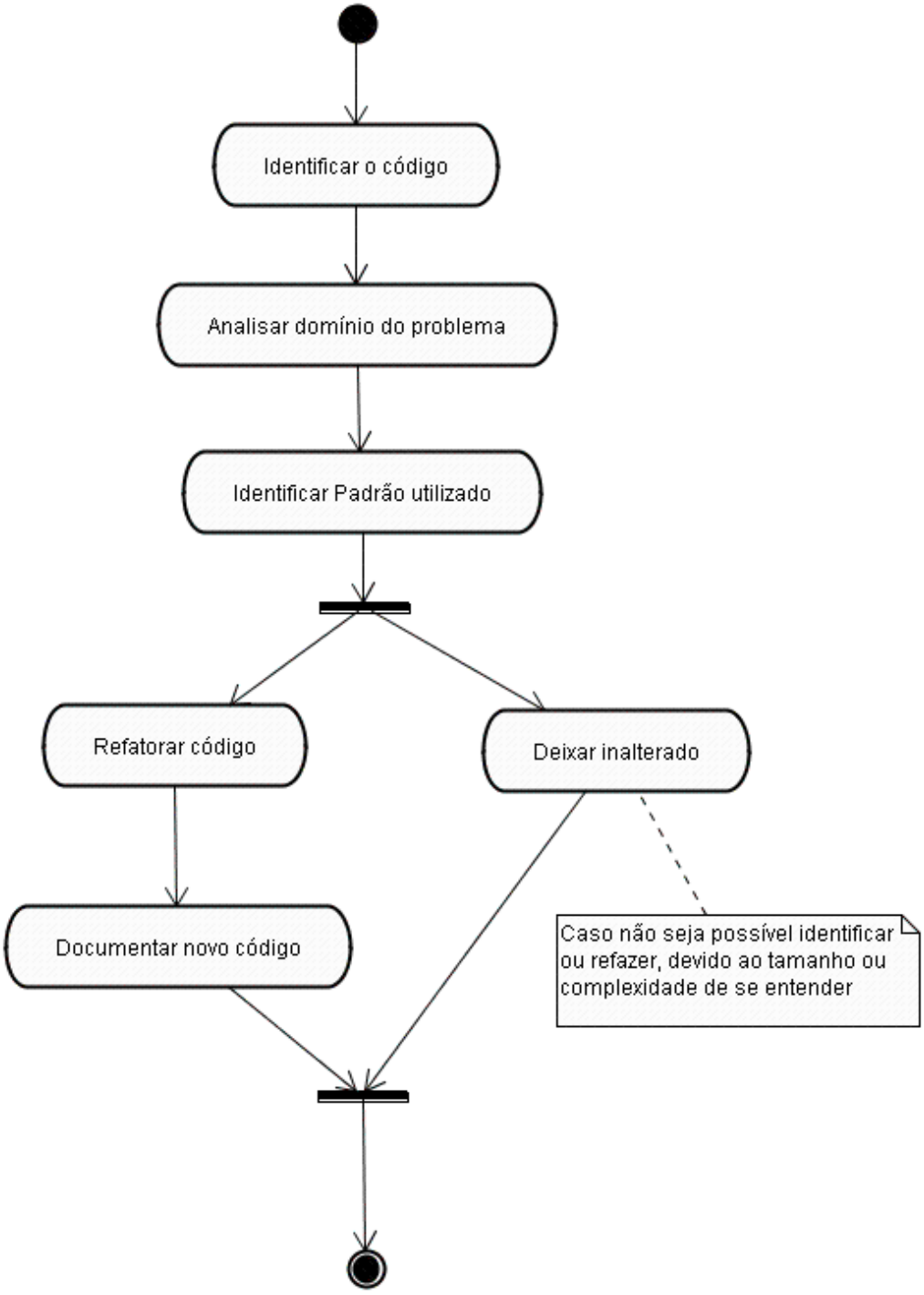

Figura 3: diagrama UML de Atividades do cenário de Refatoração do Código

Práticas como programação em pares, de técnicas de desenvolvimento ágil como Extreme Programming [3], podem ajudar na prevenção e identificação da ocorrência do anti-padrão, pois quando várias pessoas trabalham num mesmo código, aumentam-se as chances de que más modelagens ou implementações equivocadas de padrões sejam rapidamente detectadas e refeitas.

Reescrever a documentação, ou até mesmo criá-la pela primeira vez já é um passo significativo na tentativa de amenizar os constantes problemas.

Soluções para correção de um sistema com o anti-padrão “Será um Padrão?” existem. Basta o desenvolvedor ter capacidade para identificá-lo e de tomar decisões acertadas na remodelagem e recodificação.

Exemplo 1

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 diagrama ilustrado na Figura 4, onde se identificam dois supostos padrões de projeto: *Composite* [2] e *Factory* [2].

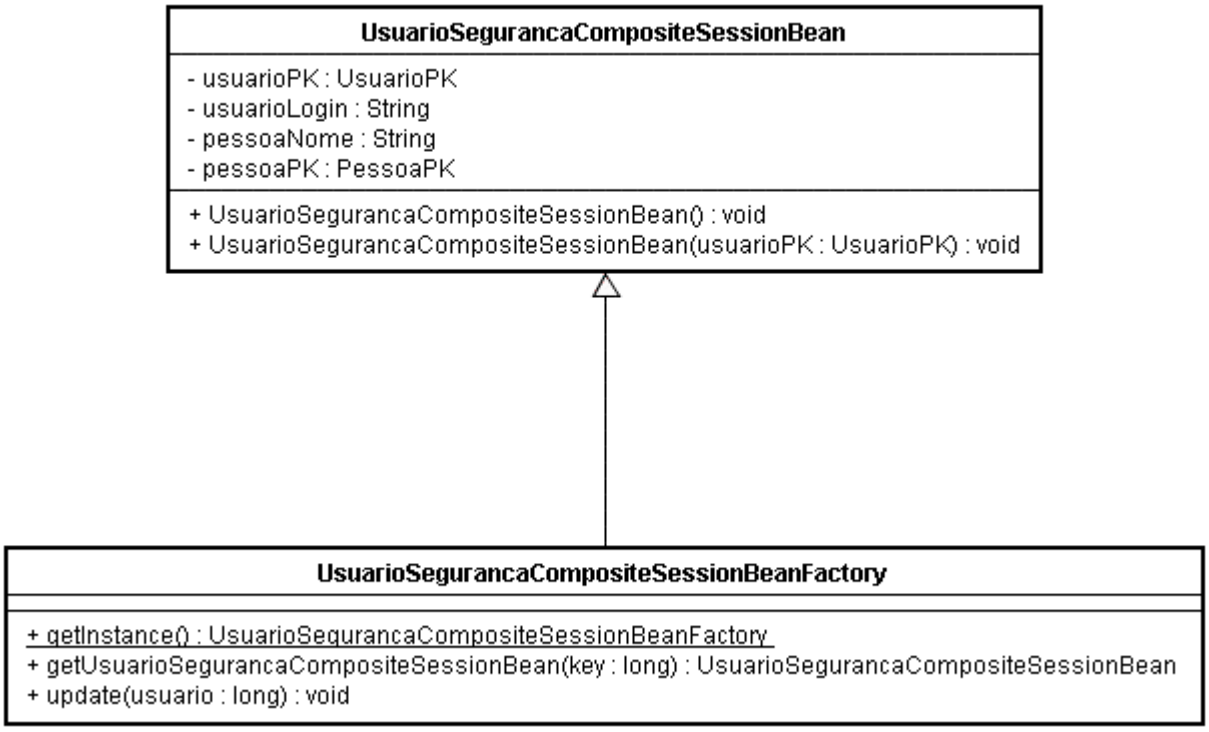

Figura 4: diagrama de classe UML do suposto Padrão *Composite* e do Padrão *Factory*

Resumidamente, um *Factory* possui como única e exclusiva finalidade a criação de objetos de tipos específicos. Já um *Composite* permite construir estruturas de objetos na forma de árvores (para representar hierarquias todo-parte), contendo tanto composições de objetos como objetos individuais atuando como nós.

Através do diagrama nota-se que no momento do desenvolvimento algo ocorreu: ou o desenvolvedor não sabia o que estava fazendo ou copiou alguma solução parecida de algum trecho de código.

Fica claro que a classe *UsuarioSegurancaCompositeSessionBean* não possui uma real composição (talvez o desenvolvedor quisesse mencionar o fato de que estava "compondo" o padrão *Composite* com um *Factory*). Observa-se ainda que esta classe não possui uma hierarquia de classes do mesmo tipo.

UsuarioSegurancaCompositeSessionBean ainda serve como base para a classe *UsuarioSegurancaCompositeSessionBeanFactory*. Pela definição de um *Factory*, constata-se que a classe responsável por criar objetos não deve herdar características de nenhuma outra classe. Como apresentado no diagrama, tem-se um alto acoplamento devido à herança e não há separação de responsabilidades, o que seria uma vantagem utilizando um objeto criado por um *Factory*.

Percebe-se que erros após erros foram cometidos nesta modelagem. A classe *UsuarioSegurancaCompositeSessionBeanFactory* deveria ter como única finalidade criar; entretanto, faz além: também atualiza dados do objeto, conforme se vê no método update(usuário: long).

"Será um Padrão?" 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, é possível encontrar ocorrências dos três Tipos citados de "Será um Padrão?".

Exemplo 2

Um outro exemplo de implementação duvidosa de um padrão está demonstrado no diagrama da Figura 5.

Nota-se que as classes *RecebeVaga* e *RecebeCandidato* são subclasses da classe abstrata *AbstractMessageCommand*. Em sua documentação, 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 execute é o método *run*( ). Nota-se que essa especificação nos leva ao padrão de projeto *Template Method* [2], no qual o comportamento de *execute* é implementado através das subclasses, podendo variar na implementação de acordo com seu tipo.

Mais uma falha de modelagem ocorreu com este último diagrama. Levando em consideração que os dois exemplos dados foram implementados pelo mesmo desenvolvedor, conclui-se que o mesmo realmente não conhece a especificação correta de um padrão de projeto. Não haveria problema se esse desenvolvedor criasse novos padrões e os renomeasse de forma conveniente; porém como ele fez uso de nomes de padrões já conhecidos, geram-se dúvidas e perda de tempo no momento de se fazer manutenção no código.

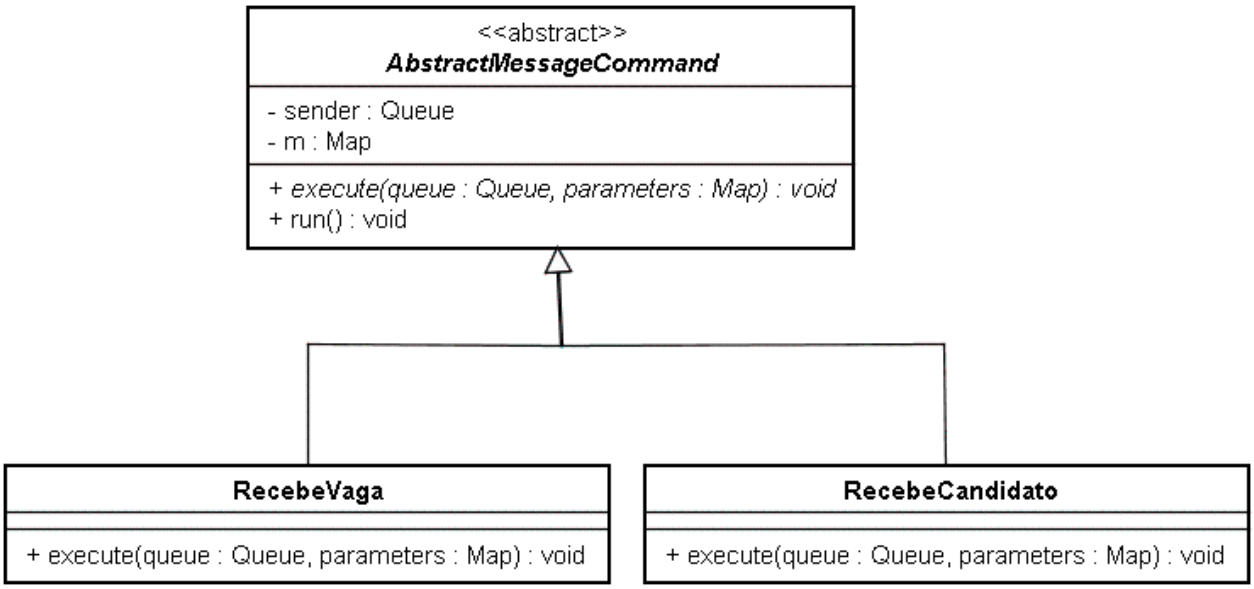

Figura 5: diagrama de classe UML do suposto Padrão *Command*

Anti-Padrões Relacionados

- Swiss Army Knife [1]. O anti-padrão "Será um padrão?" do Tipo 1 pode originar um Swiss Army Knife, se o padrão original for amplamente expandido em funcionalidades.
- Lava Flow [1]. O anti-padrão "Será um padrão?" do Tipo 2 às vezes descaracteriza o padrão original de tal forma que os responsáveis pela manutenção do código não são capazes de compreender o que ocorre e optam por não modificar nada; carregam o obscuro código versões após versões, dando origem a um Lava Flow. O mesmo pode ocorrer em implementações do anti-padrão "Será um padrão?" do Tipo 3.

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.