

MAC5715 - Tópicos de POO

Anti-padrão: “Vamo que vamo”

Rodrigo Baroni Ivan Neto

{baroni, ivanneto}@ime.usp.br

Nome do anti-padrão: “Vamo que vamo”

Também conhecido por: “Just do it”; “Programação baseada em exemplos”; “Hey, ho, let’s go!”

Escala mais freqüente: Aplicação

Tipo da resolução refatorada: Software

Raízes do problema: Preguiça, acomodação, falta de tempo e documentação

1 Evidência Anedótica

- Pronto. Acabei a função. Dá uma olhada.
- Uau! 500 linhas de código enquanto eu fui tomar um café! .. Opa, mas não entendi essas invocações de métodos desses objetos..
- Nem eu!
- Como você não sabe como funcionam estes trechos? Foi você que escreveu!
- Encontrei em uns exemplos em um site. Está funcionando!
- Mas você não sabe o por quê?!
- Por quê você insiste em ficar reinventando a roda?!

2 Contexto

Atualmente a Ciência da Computação constitui uma das áreas mais dinâmicas do conhecimento científico. Conseqüentemente, um profissional desta

área precisa conhecer uma grande variedade de tecnologias para prover soluções satisfatórias a problemas que aparecem freqüentemente.

3 Forma Geral

Desenvolvedores de software constantemente precisam elaborar código no desenvolvimento de sistemas para lidar com situações específicas. Não raramente, tais situações exigem que o programador tenha um conhecimento profundo sobre as tecnologias empregadas. Porém, problemas como prazos de entrega, complexidade das tecnologias envolvidas e dependências entre estas tecnologias levam o programador a práticas perniciosas. Nesse contexto, é comum que um programador adicione código obtido de alguma fonte externa (uma página na Internet, por exemplo) a um sistema sem compreender exatamente o seu funcionamento. Tais práticas acarretam em trechos de código com finalidade conhecida, porém com peculiaridades e implicações desconhecidas pelo próprio autor.

4 Sintomas e Consequências

- Trechos do sistema que ninguém (nem mesmo o autor) consegue explicar como funciona (embora o propósito seja conhecido).
- A tecnologia utilizada quase sempre não é empregada da melhor maneira possível, implicando em uma performance inferior.
- Conforme o sistema cresce e novas funcionalidades são adicionadas, o conjunto de trechos de código sem explicação cresce da mesma forma, levando a potenciais chances de falhas e bugs, e grandes dificuldades na manutenção.
- Destino certo de “Lava Flow”.
- Impossibilidade de documentação de certos trechos (pois ninguém entende o que ocorre nestes trechos).
- Grande dificuldade de manutenção.

5 Causas Típicas

Preguiça e falta de motivação são as principais causas deste anti-padrão. Programadores inexperientes ou desmotivados geralmente procuram soluções prontas para seus problemas na Internet, e muitas vezes não compreendem e não têm absoluta certeza de que a solução encontrada é válida. Profissionais não qualificados são os principais responsáveis pelo uso destas soluções incertas.

A falta de tempo é outro fator importantíssimo que incentiva a ocorrência do “Vamo que vamo”, pois nesse caso o programador (que está sob pressão de tempo) tenta encontrar soluções “mágicas” para seus problemas, e não dispõe de tempo para analisar satisfatoriamente a solução encontrada. Infelizmente, a adoção de prazos extremamente curtos (e muitas vezes inviáveis) é muito difundida hoje na indústria, o que torna a ocorrência desse anti-padrão muito comum.

Podemos mencionar também a grande quantidade de informações e a complexidade das tecnologias sendo empregadas como causas do anti-padrão em questão. Muitas vezes é mais fácil pegar um exemplo pronto na Internet do que estudar as tecnologias envolvidas para então desenvolver a solução. Isto, porém, traz uma série de conseqüências negativas.

Por fim, a falta de documentação das tecnologias empregadas no projeto também pode estimular o uso do “Vamo que vamo”. Isso porque, neste caso, talvez o único meio de aprender as tecnologias em uso seja através de exemplos prontos. Entretanto, caso isso ocorra, a chance de versões modificadas destes exemplos se tornarem código em produção aumenta consideravelmente.

6 Exceções Conhecidas

Em pequenas soluções encontradas como exemplo em que é possível perceber o correto funcionamento do código, onde os detalhes podem ser revistos posteriormente e cuja futura modificações podem ser feitas facilmente sem implicar no funcionamento geral do sistema.

Outra exceção seria a prototipação de sistemas, onde é necessária uma versão simplificada do sistema em funcionamento dentro de um curto prazo de tempo. Nesse caso, grande parte do código do protótipo será reescrito ou mesmo descartado. Portanto, o principal requisito neste caso é que o

protótipo funcione.

Outro uso comum e válido deste anti-padrão é na validação e teste de tecnologias destinadas a um mesmo propósito. Testes simples e sucintos podem auxiliar na escolha da tecnologia mais adequada.

Por fim, podemos concluir que trechos de código que serão descartados ou reescritos futuramente podem se utilizar deste anti-padrão visando o ganho de agilidade em concluir e validar partes relevantes do sistema.

7 Solução Refatorada

Para evitar que a prática deste anti-padrão seja comum deve-se investir em profissionais qualificados e motivados. Além disso é importante estabelecer prazos compatíveis com o volume e a complexidade do trabalho requerido.

O emprego de um método ágil de desenvolvimento, como por exemplo *eXtreme Programming*, pode evitar a proliferação deste anti-padrão. Características como programação pareada, propriedade coletiva de código e refatoração constante, comuns a muitos dos métodos ágeis, asseguram que uma ocorrência desse anti-padrão não persistirá por muito tempo.

8 Exemplos

O “Vamo-que-vamo” pode ocorrer em praticamente qualquer tipo de sistema, desde os simples até os mais complexos. Alguns exemplos simples e muito comuns de serem encontrados é por exemplo em programação em shell-script, onde o programador encontra facilmente linhas de código que realizam a funcionalidade desejada e imediatamente insere-a em seu código, sem compreender exatamente o que está ocorrendo:

#Funcao para arrumar os nomes dos arquivos

```
zzarrumanome(){ zzzz -z $1 zzarrumanome && return
local A A1 A2 D i f_R=0 f_D=0;          [ "$1" = '-d' ] && { f_D=1; shift; }
[ "$1" = '-r' ] && { f_R=1; shift; }; [ "$1" = '-d' ] && { f_D=1; shift; }
[ "$1" ] || { echo 'uso: zzarrumanome [-d] [-r] arquivo(s)'; return; }
for A in "$@"; do [ "$A" != / ] && A=${A%/}
  [ -f "$A" -o -d "$A" ] || continue; [ -d "$A" ] && {
    [ "$f_R" -eq 1 ] && zzarrumanome -r ${f_D:+-d} "$A"/*
    [ "$f_D" -eq 0 ] && continue; }
  A1="${A##*/}"; D='.'; [ "${A%/*}" != "$A" ] && D="${A%/*}";
  A2='echo $A1 | sed "s/[\\"']//g"'
  y/ABCDEFGHIJKLMNOPQRSTUVWXYZ/abcdefghijklmnopqrstuvwxyz;/s/^-/_/
  y/.../aaaaaaaaaaaaaaaaeeeeeeeeiiiiiiiccnn/
  y/.../oooooooooooouuuuuuuuubcdloosuyyy123/
  s/[^a-z0-9._-]/_/g;s/_*/_/g;s/_\([.-]\)/\1/g;s/\([.-]\)_/\1/g'
  [ "$A1" = "$A2" ] && continue ; [ -f "$D/$A2" -o -d "$D/$A2" ] && {
    i=1 ; while [ -f "$D/$A2.$i" -o -d "$D/$A2.$i" ]; do i=$((i+1)); done
    A2="$A2.$i"; }; mv -v -- "$A" "$D/$A2"; done
}
```

Ou em APIs muito complexas, em que uma sequência de invocações de métodos é realizada, apenas porque foi obtida de um exemplo em alguma fonte e o resultado desejado foi obtido. O próprio programador não tem conhecimento do porquê daquele código:

-- Exemplo utilizando uma API hipotética, com caso típico da ocorrência do padrão:

```
SpecificClass X *
MyClass::transformation( MyClass & myClass )
{
    /* Just do it */

    ClassA *objA = new ClassA( 0, 0, 0 );
    ClassB *objB = new ClassB( objA, 1, 1, 1, 0 );
    ClassC *objC = new ClassC( objB );
    ClassD *objD = new ClassD( objC, objA->getOneOfTheSubclassInstance( ) );

    objD->setParametroJ( );
    objD->setParametroK( );
    objD->setParametroL( );
    objD->setParametroM( );

    objD->doTransformation( myClass );

    return objD->getResult( );
}
```