Eduardo Ribeiro de Castro (Pós-Graduação)

Anti-Padrão: Abuso de Ferramentas

Raízes do Problema

Os ambientes de desenvolvimento atuais contém várias ferramentas de geração automática de código. Assistentes para geração de classes (*wizards*), editores de interface gráfica e repositórios de funções e trechos de código são alguns dos exemplos mais comuns. Essas ferramentas são fáceis de usar e aumentam muito a produtividade do programador. Com alguns cliques de mouse e um pouco de digitação é possível gerar dezenas ou centenas de linhas de código perfeitamente funcional e sem erros.

A facilidade de uso destas ferramentas, porém, pode deixar os programadores preguiçosos. Eles podem ser tentados a delegar partes cada vez maiores do seu trabalho para as ferramentas, mesmo quando elas não forem a opção mais apropriada. Podem também deixar por conta das ferramentas todas as decisões de implementação, sem se preocupar em analisá-las ou alterá-las. Como o código gerado normalmente é compilável e funciona, parece que a economia de trabalho vale a pena. Como resultado, obtém-se uma velocidade de desenvolvimento muito maior, à custa de um código mal-estruturado e de difícil manutenção.

Anecdotal Evidence

Programador A: "É só arrastar esse ícone pra janela e o IDE gera 500 linhas de código pra mim. Não preciso digitar nada. Minha produtividade aumentou muito".

Programador B, 4 meses depois, dando manutenção: "Pra que serve a função c1_insert (int a)? E os controles Textbox1, Textbox2, ... , Textbox14 ?"

Sintomas e Conseqüências:

Sintomas

1) O código da aplicação está repleto de variáveis e funções com nomes pouco significativos, como Textbox6 ou Command1_Execute.
2) Um mesmo trecho de código, ou variações de um mesmo trecho, são repetidas em diversas partes da aplicação.
3) Muitas classes e trechos de código estão escritos de forma ininteligível.
4) Há classes que seguem uma nomenclatura diferente do resto da aplicação.

Conseqüências

1) A manutenção do sistema fica mais trabalhosa, já que o código é difícil de entender.
2) Com a repetição de código em vários lugares, aumenta a chance de haver erros de programação (*bugs*) também em vários lugares. É uma conseqüência semelhante à do anti-padrão "*Cut-and-Paste Programming*"
3) Se os trechos de código gerados pelas ferramentas forem muito complicados e de difícil compreensão, os programadores não se arriscarão a alterá-los. O abuso de ferramentas, portanto, pode levar ao anti-padrão "*Lava flow*".

Causas Típicas

Preguiça, pressa e falta de disciplina são as principais causas deste anti-padrão. Boa parte das ferramentas gera código inteligível e de fácil compreensão se forem bem usadas. Os editores de interface gráfica atuais, por exemplo, geram um código bem estruturado e comentado. São capazes de interpretar o próprio código gerado, o que facilita muito a manutenção. Mas a facilidade que esses editores proporcionam deixa muitos programadores mal acostumados. É muito comum encontrar formulários (janelas) repletos de controles com nomes como Textbox1, Label2, Button3. Como o ambiente já sugere estes nomes, e o código gerado já funciona com eles, muitos programadores não sentem necessidade de trocá-los por algo mais significativo.

Programadores iniciantes também costumam abusar das ferramentas. Como eles não conhecem bem o arcabouço (*framework*), eles têm dificuldades para executar mesmo as operações mais elementares. Para contornar essa deficiência é comum que eles usem intensivamente assistentes (*wizards*), que criam todo o código que eles não sabem criar. Como resultado, temos o mesmo código repetido em várias partes da aplicação.

A falta de conhecimento sobre a ferramenta, ou o excesso de confiança na mesma também podem levar à geração de lixo. As ferramentas Case, por exemplo, podem ser usadas para gerar código a partir de diagramas UML ou Entidade-Relacionamento. Mas se elas forem mal configuradas, ou se as opções do diagrama não forem bem definidas, o código gerado pode sair bastante diferente do esperado.

Exceções Conhecidas

Quando construímos código descartável, que não precisará de manutenção, é aceitável o uso de ferramentas sem disciplina. Um exemplo típico são os protótipos de telas de interface gráfica. Nesses protótipos o que interessa é a aparência e a disposição dos elementos. O código por trás das telas apenas simula de forma grosseira o funcionamento da aplicação, não tendo muita utilidade prática. Desde que não se aproveite o protótipo para construir a aplicação real, não há nenhum problema em deixar as ferramentas gerarem um código confuso.

Solução Refatorada

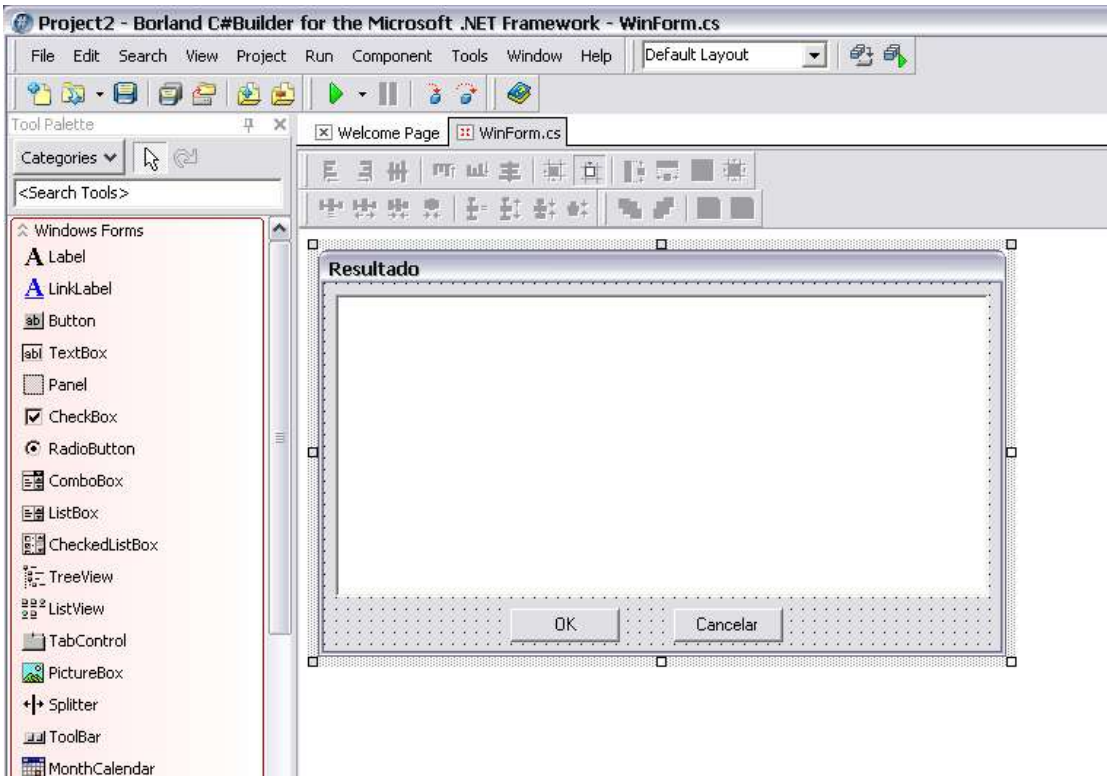
Para evitar a ocorrência deste anti-padrão, basta um pouco de disciplina no uso das ferramentas. Renomear as variáveis geradas usando nomes mais significativos sempre ajuda. Evitar o uso de assistentes, ou restringi-lo a umas poucas classes, evita a repetição do mesmo código por toda a aplicação. E no caso das ferramentas Case, recomenda-se aprender a usá-las bem e testar as opções de geração de código antes de usá-las pra valer.

Consertar o código gerado dentro deste anti-padrão pode ser relativamente simples. Para o caso nos nomes das variáveis, basta usar o comando "Localizar e Substituir" (*Find and Replace*) do editor de texto. No caso de código repetido, ferramentas de refatoração podem ajudar a isolá-lo e movê-lo para uma nova classe. Mas se o código gerado for muito complicado, e for difícil isolá-lo, a melhor saída pode ser jogar tudo fora e reescrever.

Exemplos

1) Editores de Interface Gráfica

Os editores de interface gráfica mais usados atualmente (como os do Borland JBuilder, MS Visual Studio, etc), preenchem os nomes dos controles assim que eles são criados. A figura abaixo mostra um exemplo de uma tela criada com o C#Builder.



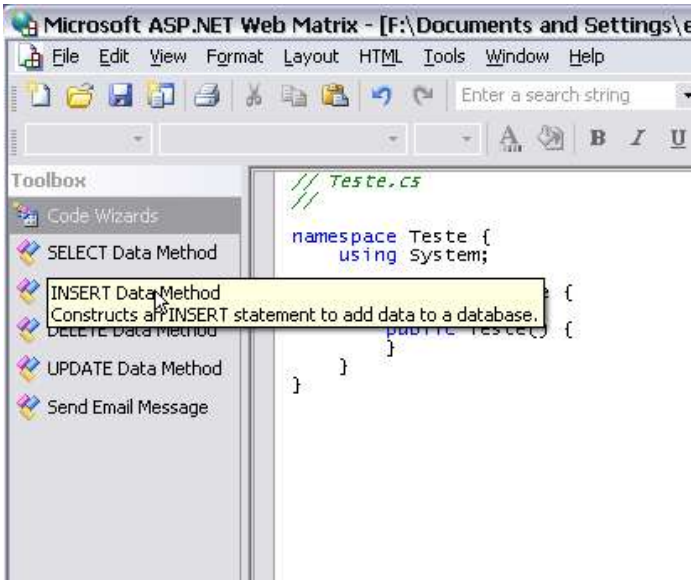
É uma janela contendo somente três controles (uma caixa de texto e 2 botões). O código gerado pelo C#Builder é:

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace Project2
{
      public class WinForm : System.Windows.Forms.Form
      {
         private System.ComponentModel.Container components = null;
         private System.Windows.Forms.Button button1;
         private System.Windows.Forms.Button button2;
         private System.Windows.Forms.TextBox textBox1;

         public WinForm()
         {
            //
            // Required for Windows Form Designer support
            //
            InitializeComponent();

            //
            // TODO: Add any constructor code after InitializeComponent
call
            //
         }
```

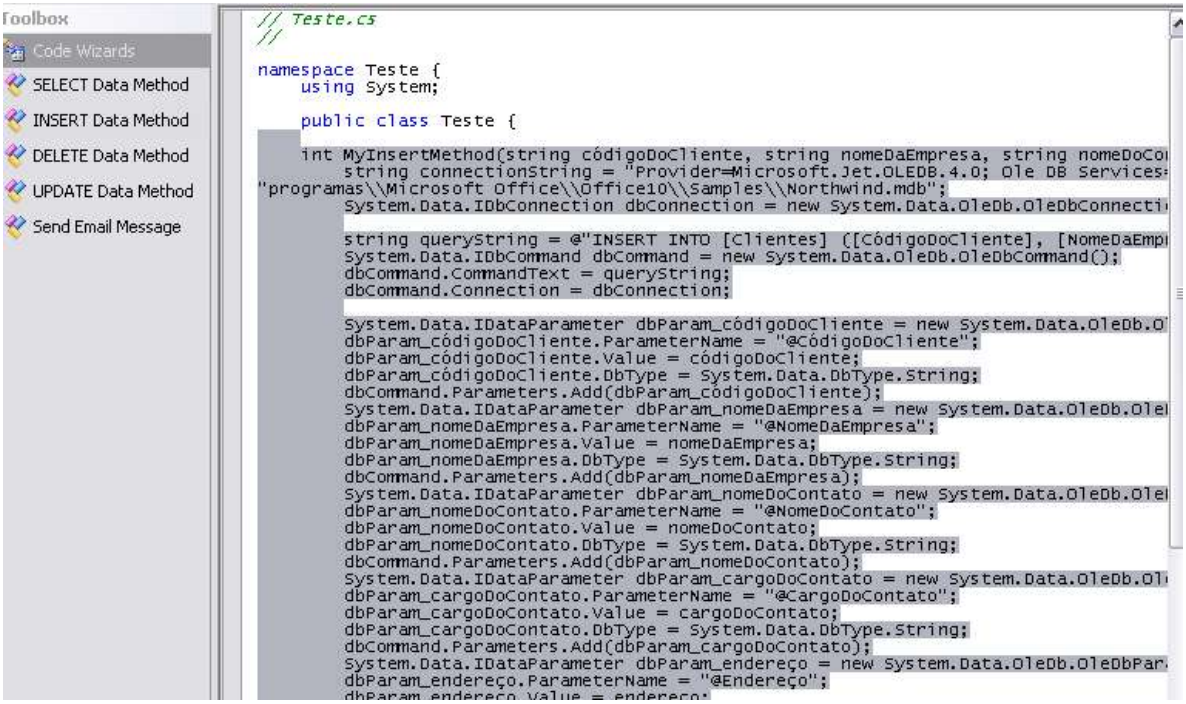
Os dois botões receberam os nomes button1 e button2. Somente inspecionando o código ou usando o editor gráfico é que se descobre qual deles é “OK” e qual é “Cancelar”. Quanto mais controles houver na janela, e quanto mais funções usarem estes controles, maior a confusão.

2) Assistentes (*Wizards*)

As figuras abaixo mostram o uso de um assistente para geração de código de acesso a banco de dados, no MS WebMatrix.



Basta arrastar um ícone e preencher algumas opções apresentadas e o código é gerado.



Em princípio não há nenhum problema com o código gerado. O problema começa quando a facilidade de uso da ferramenta conspira contra o planejamento do sistema. O

programador pode ser tentado a usar o assistente em todas as telas do sistema, ao invés de centralizar o acesso ao banco de dados em algumas classes. Posteriormente, se as colunas de alguma tabela forem alteradas, ele vai ter que “caçar” todas as telas que acessam esta tabela e aplicar em cada uma as alterações necessárias.