TRANSFORMAÇÃO DE UM PROJETO DE SOFTWARE ATRAVÉS DE REFATORAÇÕES E PADRÕES DE PROJETO

Adriano Lourenço da Conceição, Ismar Frango Silveira, Juliano Schimiguel, Luis Naito Mendes Bezerra

Pós-Graduação Lato Sensu em Engenharia de Websites
Universidade Cruzeiro do Sul (UNICSUL)
Av. Dr. Ussiel Cirilo, 225 – 08060-070 – São Paulo – SP – Brasil
adrianolc@gmail.com, {ismar.silveira,juliano.schimiguel,luis.naito}@unicsul.br

Resumo- A manutenção pode ser considerada um dos pontos mais críticos no desenvolvimento de software, já que o custo de um projeto pode dobrar se sua manutenção for complexa. Este trabalho visa apresentar o tema refatoração, que tem como objetivo facilitar a expansão e manutenção de projetos de softwares orientados a objetos. As refatorações auxiliam na atualização do software sem alterar seu comportamento observável. Pretende-se apresentar a aplicação das principais técnicas de refatoração baseadas em padrões de projeto tendo como base um estudo de caso real desenvolvido para um software de gerenciamento acadêmico.
Palavras-chave: Padrões de Projeto, Refatoração, Engenharia de Software
Área do Conhecimento: Ciência da Computação

Introdução

Na área de desenvolvimento de software, é comum o desenvolvimento focado em cronogramas que deixam a qualidade do projeto em segundo plano. Quando uma empresa qualquer produz um software, a equipe que o produz, na maioria dos casos, com o passar do tempo, é desfeita. Se a empresa foi precavida em documentar o software desenvolvido, ela com certeza terá menos problemas na manutenção ou expansão do software em questão, já que os novos contratados terão um guia formal para efetuar possíveis atualizações no software – o que não significa que não terão problemas, já que programadores pensam e escrevem códigos para um mesmo objetivo de formas diferentes.

Outro aspecto importante é o primeiro contato com o código de um software desconhecido, uma vez que com o passar do tempo, o que foi projetado e padronizado poderá ser modificado, perdendo assim sua estrutura. Nesses casos o conceito de Refatoração surge com grande valia.

Refatoração para o desenvolvimento de software significa brevemente melhorar o projeto após ele ter sido escrito, ou seja, modificar a estrutura interna de um software para torná-lo mais fácil de ser entendido e menos custoso de ser modificado sem alterar seu comportamento observável (FOWLER, 2004).

Ao tentar fazer a manutenção ou expansão de um software, programadores podem encontrar duas situações problemáticas: Projetos padronizados, mas extremamente complexos, com uma forte tendência futura à despadronização, assim como projetos sem nenhum tipo de padronização, com uma forte tendência a extinção. Isto posto, surge a questão: o que leva um software a ser extinto? Existem várias respostas para esta pergunta, mas a principal delas é o custo de sua manutenção ou expansão. Assim, uma outra pergunta surge: como fazer a manutenção ou expansão de um software sem ter prejuízos?

A resposta para esta pergunta são os “dois chapéus” (FOWLER, 2004; ASTELS et al., 2002), expressão usada por Kent Beck – um dos criadores do conceito Programação Extrema (*Extreme Programming*). Segundo Beck (FOWLER, 2004) os dois chapéus servem para dividir o tempo de desenvolvimento entre duas atividades distintas: adicionar funcionalidades e refatorar. Será usado um dos chapéus para refatorar um bloco de código mal estruturado: uma vez que este bloco de código estiver legível e estruturado, troca-se de chapéu para adicionar a nova funcionalidade.

Este artigo propõe a refatoração das principais rotinas de um software de gestão acadêmica chamado SGA (Sistema de Gerenciamento Acadêmico), atualmente utilizado na Fundação Instituto de Pesquisas Contábeis, Atuariais e Financeiras (FIPECAFI), São Paulo, SP, desde 2004. Além disso, irá exemplificar a aplicabilidade de algumas refatorações importantes para reformulação de um projeto mal concebido, com a introdução de padrões de projetos (*Design Patterns*) (GAMMA et al., 1995). A UML (*Unified Modeling Language*) será utilizada para a visualização dos artefatos complexos do software.

Refatoração e Padrões de Projeto

Segundo Ron Jeffries (2001) refatoração é um processo formal (mas não complexo) que permite atualizar códigos sem mudar suas estruturas, onde cada etapa de alteração pode ser reversível caso o resultado não seja o esperado. Já Fowler (2004) diz que refatoração para o desenvolvimento de software significa basicamente melhorar o projeto após ele ter sido escrito, ou seja, modificar a estrutura interna de um software para torná-lo mais fácil de ser entendido e menos custoso de ser modificado sem alterar seu comportamento observável. Ao ganhar prática na aplicação das principais refatorações, pode surgir a necessidade de se catalogar as experiências para usos futuros. Para cada nova entrada no catálogo de refatorações é indicada por Fowler (2004) a seguinte estrutura: nome, resumo, motivação, mecânica (passo-a-passo) e exemplos de situações onde foi aplicada a refatoração.

Já em relação a Padrões de Projeto, Christopher Alexander et al. (1977) foram os primeiros a definir padrão como uma entidade que descreve um problema que ocorre repetidamente em um ambiente. Ao descrever a essência da solução para este problema, pode-se usar essa solução várias vezes de modos diferentes. Gamma et al. (1995) referenciam os padrões de projeto como sendo descrições de objetos que se comunicam e classes que são customizadas para resolver um problema de projeto genérico em um contexto específico. Larman (2001) afirma que desenvolvedores experientes construíram um repertório de princípios gerais e de soluções idiomáticas que os guiam na criação de softwares. Esses princípios e idiomas, codificados de forma estruturada, que descrevem o problema e a solução e são identificados por um nome, podem ser chamados de padrões.

Os padrões de projetos expressam soluções que já foram utilizadas em inúmeros projetos por projetistas experientes. Assim, um padrão, além de apresentar uma solução para um determinado problema, explica porque a solução é bem sucedida. O termo “Padrões de Projeto” refere-se a padrões que visam solucionar problemas na fase de projeto de software. Entretanto, existem padrões para várias áreas da indústria de software (TEIXEIRA, 2006).

Os padrões GoF (Gang of Four) (Gamma et al., 1995) constituem-se certamente no catálogo mais conhecido de Padrões de Projeto, relacionados com determinados tipos de problemas de projeto orientado a objetos. O padrão descreve quando é possível, e os custos e conseqüências resultantes de sua aplicação. Tais padrões foram categorizados entre padrões de criação, estruturais e comportamentais. Cada categoria expressa as intenções de seus padrões para o desenvolvimento de cada fase do sistema.

O relacionamento entre refatorações e Padrões de Projeto já era previsto em Gamma et al. (1995). Nesse contexto, é digno de nota o trabalho de Kerievsky (2004), que apresenta um conjunto de Refatorações envolvendo padrões de projeto, assim como trabalhos que tentam automatizar este processo, como exposto em Cinnéide (2000) ou Rajesh e Janakiram (2004).

Metodologia e Estudo de Caso

O sistema SGA (Sistema de Gerenciamento Acadêmico) utilizado no trabalho foi escrito na linguagem Visual Basic.Net 2005 com seus dados sendo armazenados em SQL Server 2000. O software é composto por rotinas que indicam se um usuário tem permissão ou não para acessar um determinado recurso, o acesso aos recursos são monitorados através de registros de log gerados em cada evento que acessa os dados armazenados. As principais rotinas deste software estão ligadas ao controle de acesso, registros de log e o acesso aos dados.

Não há uma divisão bem definida das camadas do sistema, as classes que são responsáveis pelas regras do negócio também são responsáveis pela persistência dos dados (Operações de CRUD – *Create, Read, Update* e *Delete*). As classes não possuem características de projetos orientados a objetos, já que elas possuem vários métodos procedurais que na maioria recebem listas de parâmetros muito extensas, e são responsáveis por boa parte do processamento das informações do sistema.

A figura 1 apresenta uma visão geral das classes do sistema, com ênfase para a classe “Curso”, que possui muitas responsabilidades mal delegadas, resultando em uma coesão muito baixa, já que é responsável por inúmeros acessos ao banco de dados e também por varias regras do negócio. A maioria das classes do sistema possui as mesmas imperfeições.

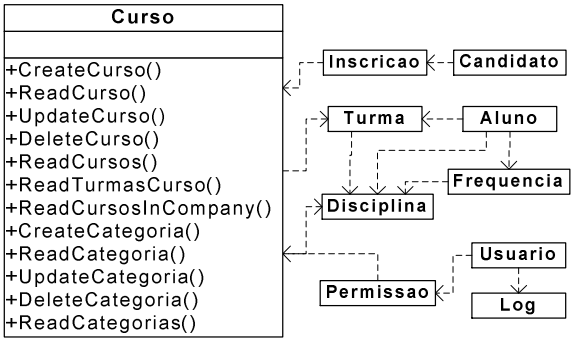

Figura 1. Diagrama de classes inicial

Como apresentado na figura 1, os relacionamentos entre as classes são de pura dependência. Não existem heranças, composições, agregações e outros tipos de relacionamentos, todas as classes de forma independente executam tarefas para um objetivo não muito bem estabelecido.

O SGA, apesar de utilizar uma linguagem orientada a objetos (Visual Basic.Net 2005), é composto por classes de natureza procedural. Estas classes são compostas por métodos longos de acesso. O SGA precisa ser refatorado para se tornar um autêntico projeto de software orientado a objetos. É desejável isolar as classes procedurais para possibilitar a transformação dos registros de dados em objetos, o comportamento de cada classe procedural pode ser dividido entre os objetos (FOWLER, 2004).

Como o SGA persiste seus dados em um banco de dados relacional (SQL Server) cada tabela do banco de dados pode ser representada no sistema como uma classe de dados, de forma a que o código procedural possa ser movido para uma única classe. Neste exemplo a nova classe será nomeada como *GerenciadorDe<Classe>* (Figura 2).

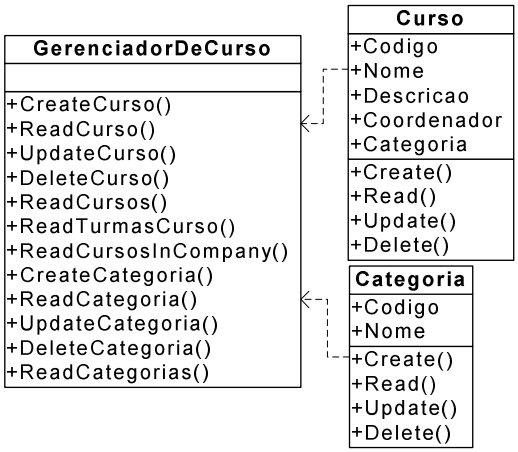

Figura 2. Códigos procedurais encapsulados em uma única classe

À medida que os procedimentos forem sendo decompostos, pode-se aplicar a refatoração "Mover Método" (FOWLER, 2004) para mover cada procedimento resultante para a classe burra de dados apropriada. Esta refatoração depende de várias outras que serão exibidas no decorrer deste trabalho, de forma que todo o comportamento de caráter procedural das classes originais deve ser removido da aplicação nas próximas refatorações.

Uma vez que os códigos procedurais foram isolados em uma única classe, é possível observar o "mau cheiro" mais comum em códigos procedurais: o código duplicado (FOWLER, 2004). As responsabilidades de se conectar ao banco de dados e realizar as operações de CRUD podem ser delegadas a uma nova classe, pois a classe original esta sendo responsável pelo trabalho de duas.

Se forem observados os métodos que acessam dados, todos são muito semelhantes diferenciados somente pelos parâmetros de chamada das procedures armazenadas no banco de dados. Além disso, todos são dependentes da API que acessa o banco de dados SQL Server (SqlClient). Se for necessário um dia trocar de fornecedor de banco de dados, seria necessário atualizar todas as classes de negócio simultaneamente, o que praticamente seria refazer todo o software.

Existem várias formas de resolver este problema; uma delas é tirar do cliente (camada de negócio) a responsabilidade de persistir os dados. A figura 3 exemplifica o uso de padrões através da classe "Acessador", uma interface unificada de acesso aos dados que representa o padrão *Façade* (GAMMA et al., 1995).

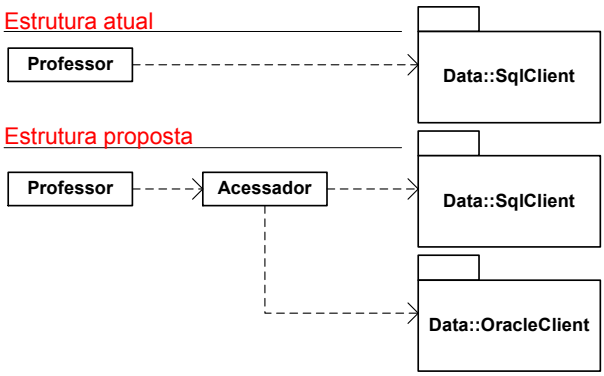

Figura 3. Diagrama de classes da estrutura de acesso aos dados

O SGA registra logs textuais de cada ação do sistema dificultando o rastreamento de possíveis infrações ou falhas. A cada ação de persistência de dados o SGA registra um log contendo o usuário executor da ação, a data e hora, além de uma breve descrição.

É desejável refatorar a estrutura atual para obter um melhor aproveitamento dos registros de log gerados pelo sistema. O objetivo é monitorar os dados de entrada e saída de uma tela (Serviço) qualquer. Além disso, o controle de acesso aos recursos se demonstra deficiente. Uma vez que sempre é necessário reescrever o trecho de código responsável pelo controle gerando duplicidades, o pior dos maus cheiros (FOWLER, 2004). Os logs são salvos em um banco de dados relacional. A estrutura de acesso aos dados gerada nos exemplos anteriores será usada para persistir o objeto log e seus parâmetros.

Da mesma maneira, deve-se gerar uma nova estrutura para controlar os perfis dos usuários, onde grupos representem o perfil de cada usuário. A figura 4 apresenta a estrutura proposta para o controle de usuários e o registro de logs através de um diagrama de classes.

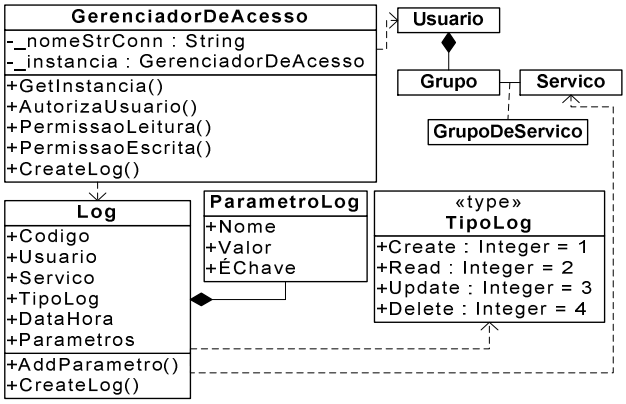

Figura 4. Diagrama de classe – Fachada – "GerenciadorDeAcesso"

Resultados e Conclusões

Este artigo apresentou as principais técnicas de desenvolvimento de software baseado em padrões de projetos elaborados por especialistas no tema. Além disso, ilustrou com exemplos práticos a aplicação de algumas refatorações em um sistema de gerenciamento acadêmico (SGA) que apesar de utilizar uma linguagem orientada a objetos tinha sua estrutura procedural.

Um desenvolvedor habituado com o paradigma estruturado/procedural de desenvolvimento de software tem a tendência de ignorar certos conceitos ao iniciar seus trabalhos com orientação a objetos. O resultado na maioria dos casos é catastrófico. O SGA apresentado no estudo de caso tinha cem por cento de suas rotinas no modelo procedural.

As refatorações aplicadas possibilitaram a transformação do modelo procedural em um modelo orientado a objetos sem afetar o comportamento observável do sistema. Além disso, algumas refatorações sugerem a aplicação de padrões para projetos de software orientados a objetos. As vantagens na utilização destes padrões são inúmeras. Eles garantem a confiabilidade do projeto, pois são soluções aprovadas e testadas que já foram usadas em diversas situações de domínios diferentes. Entretanto, trabalhos recentes como o de Strogyllos e Spinellis (2007), por exemplo, refutam essa tese ao demonstrar um exemplo de Refatoração que causou uma piora nas métricas. Contudo, os autores não são conclusivos a respeito do que realmente causou a piora nas métricas: se foram as refatorações em si ou seu emprego inadequado. Trabalhos que enfocam especificamente métricas na aplicação de padrões GoF para Refatoração são esclarecedores, como o de Muraki e Saeki (2001). Assim, conclui-se que mais pesquisas devem ser realizadas para que se possa afirmar com segurança se a aplicação de Refatoração seria sempre benéfica a um sistema.

Referências

ASTELS, D., MILLER, G. e NOVAK, M. (2002). Extreme Programming: Guia Pratico. trad. Kátia Roque. Rio de Janeiro: Campus.

ALEXANDER, C., ISHIKAWA, S., SILVERSTEIN, M., JACOBSON, M., FIKSDAHL, I. e ANGEL, K.S. (1977). A Pattern Language. New York: Oxford University Press.

CINNÉIDE, M. Ó. (2000). Automated refactoring to introduce design patterns. Proceedings of the 22nd international conference on Software engineering – ICSE'00. Limerick, Irlanda.

FOWLER, M. (2004). Refatoração: Aperfeiçoando o projeto de código existente. Porto Alegre: Bookman.

GAMMA, E., VLISSIDES, J., JOHNSON, R. e HELM, R. (1995). Design Patterns: Elements of Reusable Object Oriented Design. New Jersey,. USA: Addison-Wesley.

JEFFRIES, R., ANDERSON, A. e HENDRICKSON, C. (2001). Extreme Programming Installed. EUA: Addison Wesley- Longman.

KERIEVSKY, J. (2004) Refactoring to Patterns. New Jersey,. USA: Addison-Wesley.

LARMAN, C. (2001). Applying UML and Patterns. 2 ed. New Jersey, USA: Prentice Hall.

MURAKI, T. e SAEKI, M. (2001). "Metrics for Applying GoF Design Patterns in Refactoring Processes". Proceedings do 4th International Workshop on Principles of Software Evolution. Viena, Áustria.

TEIXEIRA, J. (2006). "Introdução a Padrões de Software". Disponível na Internet: http://www.lia.ufc.br/~eti2005/menu/modulos/PS/Parte1.pdf. Acesso em 10/06/2008

RAJESH, J. e JANAKIRAM, D. (2004). Proceedings of the 6th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming. Verona, Itália.

STROGYLLOS, K. e SPINELLI, D. (2007). "Refactoring – Does it Improve Software Quality?". Proceedings do 5th International Workshop on Software Quality – IEEE WoSQ'2007. Austin, TX, EUA.