Розглянуто можливості використання сегментації для проектування модуля інформаційної системи у структурі бази даних. Запропоновано використання технологій рефакторингу для поліпшення та оптимізації програмних модулей

Ключові слова: рефакторинг, база даних, фрагмент, компонент

Рассмотрена возможность использования сегментации кода для проектирования модуля информационной системы в структуре базы данных. Предложено использование технологий рефакторинга для улучшения и оптимизации программных модулей

Ключевые слова: рефакторинг, база данных, фрагмент, компонент

The possibility of using segmentation of the code for the design of the module information system in the database structure. Proposed use of technology refactoringto improve and optimize the software modules

Keywords: refactoring, database, code, components

УДК 004.05

ОПТИМИЗАЦИЯ ПРОЦЕССА СТРУКТУРИЗАЦИИ КОДА

С.Г. Чёрный
Кандидат технических наук, доцент
Кафедра электрооборудования судов и автоматизации производства
Керченский государственный морской технологический университет
ул. Орджоникидзе, 82, г. Керчь, АР Крым, 98309
Контактный тел.: 050-590-77-08
E-mail: black@kerch.net

Постановка задачи в общем виде и ее актуальность

Современные приложения имеют сложную и многослойную архитектуру. Трудно представить сегодня серьезное коммерческое приложение, не взаимодействующее с базой данных (БД), что подразумевает уровень архитектуры, обеспечивающий взаимодействие с БД. Рефакторинг этого уровня - довольно трудоемкий процесс. По определению, рефакторинг (англ. refactoring) - процесс изменения внутренней структуры программы, не затрагивающий её внешнего поведения и имеющий целью облегчить понимание её работы [1]. В основе рефакторинга лежит последовательность небольших эквивалентных (то есть сохраняющих поведение) преобразований. Иногда понятия рефакторинга заключается в определении регламентированного способа реструктуризации кода с применением небольших итерационных шагов. Прежде всего рефакторинг обеспечивает постепенное развитие кода во времени, в результате чего реализуется эволюционный подход к программированию. Особенностью рефакторинга является то, что он сохраняет функциональную семантику базовости кода.

Многие разработчики, организующие свою работу на принципах адаптивного программирования, и, в частности, специалисты по экстремальному программированию (Extreme Programmer - XPer), полагают, что рефакторинг является ведущим подходом к разработке. На практике можно столь же часто встретить примеры применения рефакторинга небольших фрагментов кода, как и введения операторов if или циклов.

К наиболее часто употребимым методам рефакторинга можно отнести:

- изменение сигнатуры метода (Change Method Signature);
- инкапсуляция поля (Encapsulate Field);
- выделение класса (Extract Class);
- выделение интерфейса (Extract Interface);
- выделение локальной переменной (Extract Local Variable);
- выделение метода (Extract Method);
- генерализация типа (Generalize Type);
- встраивание (Inline);
- введение фабрики (Introduce Factory);
- введение параметра (Introduce Parameter);
- подъём поля/метода (Pull Up);
- спуск поля/метода (Push Down);
- замена условного оператора полиморфизмом (Replace Conditional with Polymorphism).

Рефакторинг кода должен осуществляться до полного исчерпания его возможностей, поскольку наибольшая производительность может быть достигнута только в условиях работы с исходным кодом максимально высокого качества. При возникновении необходимости добавить к коду новые возможности следует оценить реплицируемого кода в аспектах данного проекта, что позволит успешно реализовать требуемые средства. Если ответ на этот вопрос является положительным, то можно приступать к добавлению новых функциональных средств. При отрицательном решении данного аспектного внедрения код вначале должен быть подвергнут рефакторингу, для того чтобы он имел оптимальный возможный проект, и только после

этого возможно осуществление процесса добавления новых функций. Применение указанного подхода приводит к значительному увеличению объема работы, но практика показывает, что если доработка начинается с высококачественного исходного кода, после чего постоянно осуществляется рефакторинг этого кода для поддержки его в том же состоянии, то все новые замыслы реализуются чрезвычайно показательна и эффективна.

Рефакторинг нужно применять постоянно при разработке кода. Основными стимулами его проведения являются е задачи:

- необходимо добавить новую функцию, которая не достаточно укладывается в принятое архитектурное решение программного модуля;
- необходимо исправить ошибку, причины возникновения которой не выделены четко структурированной базовой внешней формой;
- проблематика в командной разработке, которая обусловлена сложностью логики программного продукта.

Операции рефакторинга БД концептуально являются более сложными, чем операции рефакторинга кода, поскольку при проведении операций рефакторинга кода необходимо заботиться лишь о сохранении функциональной семантики структуры фрагментации, а при осуществлении операций рефакторинга БД возникает процесс необходимости и сохранение информационной семантики. Достаточно важным фактором является то, что усложнение операций рефакторинга БД может быть обусловлено наличием большого количества связей, поддерживаемых архитектурой БД [2]. Рефакторинг кода может быть связан не только с изначальной архитектурной проблематикой. Процесс необходимо осуществлять после комплексного анализа структуры и выявления «фрагментарных участков необходимых для оптимизации со стороны команды разработчиков». В этом случае проводится рефакторинг «фрагментарного» участка программы. Профилировка поможет его определить.

При осуществлении процесса рефакторинга необходимо четко представлять какой из методов более лучший и оптимальный для данной структуры, ведь нелогичное использование методов приведет к трудностям роботы программы. Опытные разработчики четко представляют и делят процесс оптимизации кода и процесс рефакторинга. При процессе рефакторинга разработчик старается сделать так, чтобы код стал удобнее для понимания и его поддержки, а при оптимизации кода приходится делать процедуры, которые приводят к обратному эффекту, что приводит к проблемам читаемости кода, но за счет этого возрастает скорость его выполнения.

Изложение основного материала

Многие разработчики программного обеспечения считают, что при рефакторинге лучше полагаться на интуицию, основанную на опыте, но можно выделить наиболее очевидные причины, когда код нужно подвергнуть процессу «рефакторинг»:

- дублирование фрагментов кода;
- длинный метод реализации;
- классы данных;
- использование большого класса;
- envious function;
- использование достаточно длинного списка параметров;
- примение и использование избыточных временные переменных;
- несгруппированные данные.

В общем аспекте проблемы, возникающие при проведении рефакторинга делятся на две категории:

- проблемы, связанные с БД;
- проблемы изменения интерфейсов.

Процесс переноса фрагментов кода можно разделить на обобщенные этапы:

- в оригинальном коде постепенно заменяется все, что использует специфические возможности исходного языка, на более простое, но эквивалентное по функционалу, что может привести медленной работе кода и к не корректному внешнему виду;
- редактируемый код приводится к виду, который сможет собрать новый компилятор.
- переносятся тесты, и код на новом языке доводится до совпадения по функционалу с оригинальным кодом.

В настоящее время существует много ORM-прослоек, позволяющих общаться с БД в терминах сущностей. Обычно сущности соответствуют таблицам БД. Например, есть БД, со следующей структурой (рис. 1):

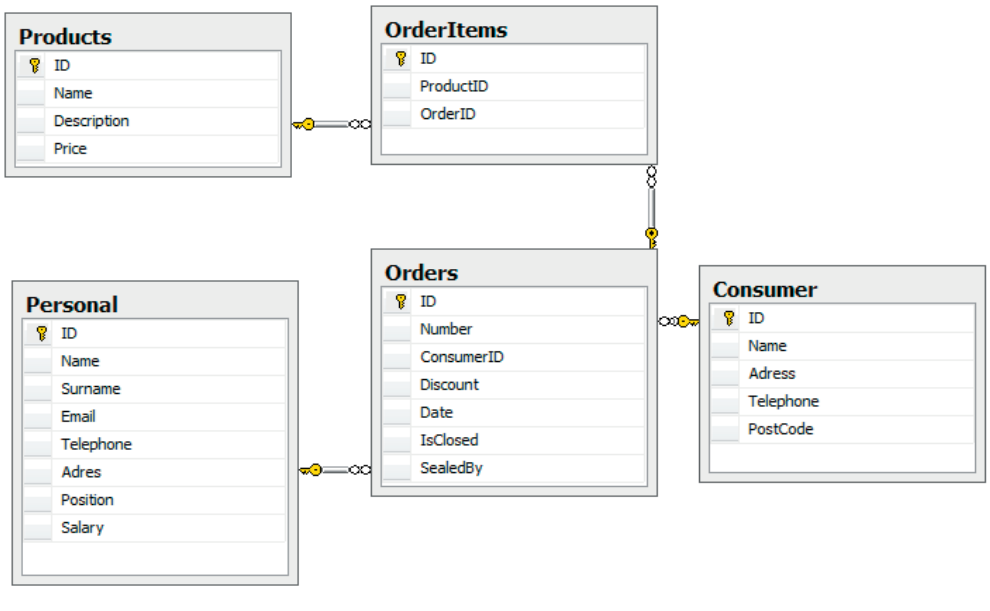

Рис. 1. Структура семантики связей БД

Инструменты рефакторинга могут быть реализованы с DMS для различных языков включают в себя [2-3]:

- поиск и удаление повторяющегося кода;
- форматирование кода для удобства просмотра;
- переименование идентификаторов;
- замена spaghetti-tangles gotos со структурированным кодом;
- использование CASE фрагментов вместо вложенных IF на той же переменной;
- включение функции/метода документации автоматизированного извлечения фактов;
- стиль проверки отладки;
- преобразование блоков кода в функции/подпрограмм/методов с соответствующими параметрами;
- удаление «мертвого кода;
- реструктуризация APIs-интерфейсы для поддержки различных OS.

Все коммуникации с БД приложение осуществляет через прослойку DataAccessLayer, которая предоставляет собой сервисы для доступа к данным. В результате этого данные возвращаются в виде сущностей, каждая из которых имеет структуру соответствующей таблицы БД. То есть имеется набор сущностей Products, Personal, Orders, Consumer, OrderItems (рис. 2). DMS может быть настроен для выполнения произвольного рефакторинга для выбранного языка.

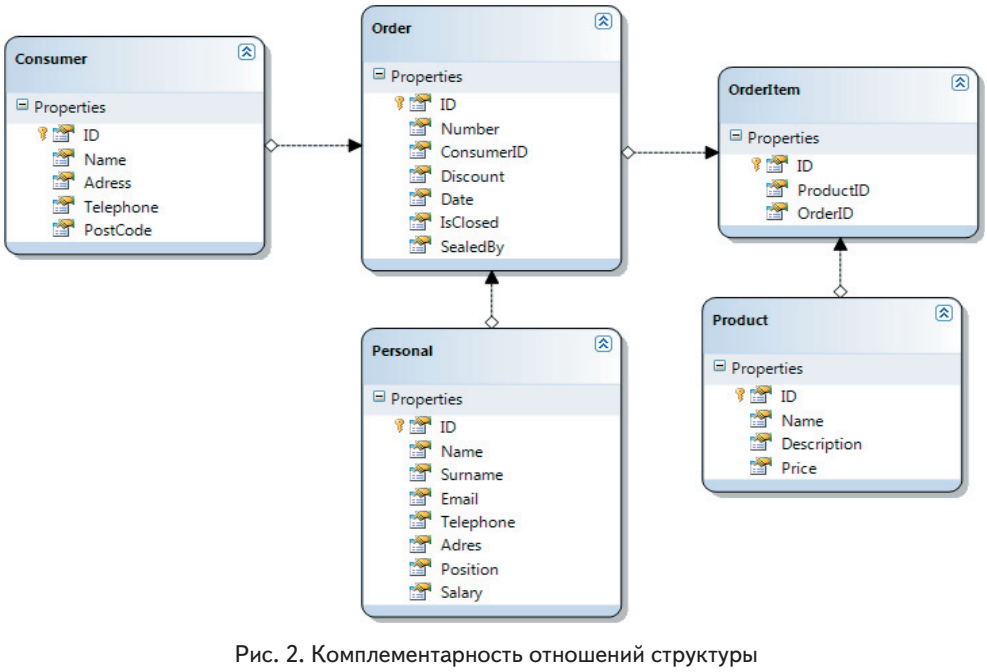

Рис. 2. Комплементарность отношений структуры

Семантические конструкции могут сделать это в качестве функции, или может предоставить программному модулю с помощью инструментов и обучение по использованию DMS для осуществления таких процессов рефакторинга (рис. 3) [2].

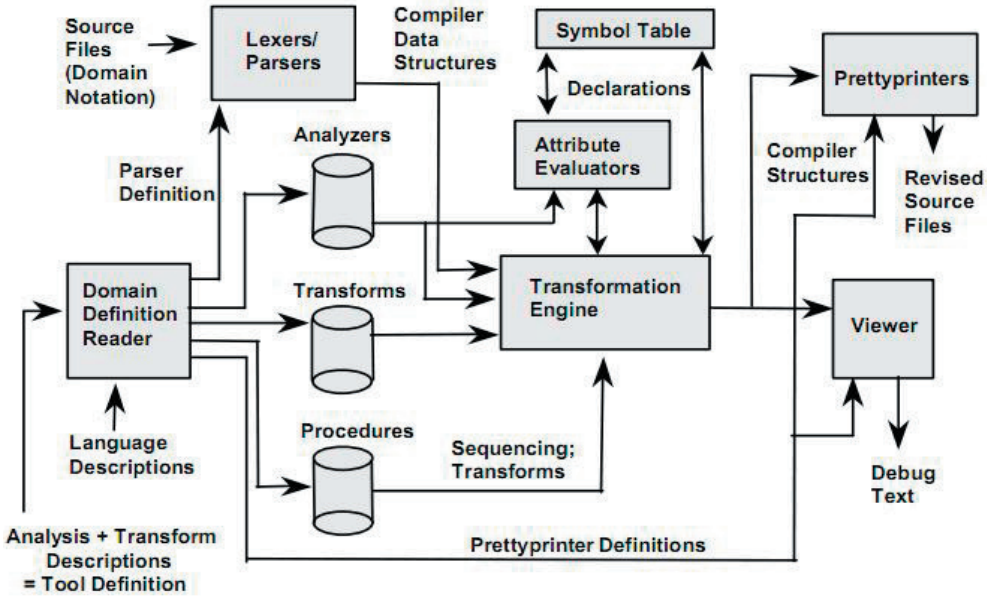

Рис. 3. Архитектура DMS

Если посмотреть на структуру объектов, то можно заметить, что всем им присущи некоторые одинаковые поля – ID. Если воспользоваться терминами DDD (Domain Driven Design), то можно выделить общий интерфейс для всех объектов (рис. 4):

```
interface IEntity
{
    int ID { get; set; }
}
```

Рис. 4. Фрагментарность цикла

Теперь все объекты имеют одинаковый интерфейс, и это открывает новые возможности. Приступим к рефакторингу сервисов, через которые приложение работает с БД. Идея проста: если объекты имеют общий интерфейс, то и сервисы, работающие с ними, тоже можно обобщить (однако это утверждение не истинно во многих специфических случаях при работе с данными, однако в тоже время позволяет сильно облегчить работу с БД).

Создадим обобщенный интерфейс сервиса (рис. 5):

```
interface IRepository<T> where T : IEntity
{
    IList<T> GetAllItems();
    T GetItemByID(int id);
    void AddItem(T item);
    void DeleteItemByID(int id);
}
```

Рис. 5. Интерференция базовости

Тут используется такая возможность языка C#, как обобщения, позволяющая воплотить следующую идею – обобщенный сервис для работы с данными (рис. 6).

```
class CommonRepository<T> : IRepository<T> where T : IEntity
{
    List<T> storage = new List<T>();

    public CommonRepository()
    {
        //заполняем данными из БД storage
    }

    public IList<T> GetAllItems()
    {
        //...
    }

    public T GetItemByID(int id)
    {
        //...
    }
    public void AddItem(T item)
    {
        //...
    }
    public void DeleteItemByID(int id)
    {
        //...
    }
}
```

Рис. 6. Комплементация модуля для работы с данными

Например, чтобы получить все проекты из БД, в клиентской части приложения достаточно написать

IRepository repository =

= new CommonRepository<Products>();

var products = repository.GetAllItems();

Теперь приложение не привязано к конкретной реализации сервиса, то есть общие манипуляции с данными выполняются довольно легко.

При создании более специфических классов, реализующих интерфейс IRepository, хорошо выделить общий абстрактный класс BaseRepository<T> where T: IEntity, в который вынести реализацию общих методов.

```
class ProductsRepository : BaseRepository<Product>, IRepository<Product>...
```

Рис. 7. Группирование кластера репозитария

Тогда CommonRepository и другие репозитории можно специфицировать так (рис. 7):

Выводы и предложения

Такой рефакторинг сервисов, предоставляющих работу с данными, открывает возможность для рефакторинга других частей приложения, позволяет выполнить преобразования некоторых частей приложения в соответствии с полезными методиками, однако тут есть и свои минусы – трудоемкость и порой невозможность проведения таких изменений, если приложение больших размеров и большая часть сервисов уже написана.

Все же лучше пользоваться практикой программирования по контракту, находить и обобщать контракты, выявлять это все на ранних стадиях разработки приложения, что позволит в дальнейшем производить экономию трудового ресурса разработчика.

Литература

1. Операции рефакторинга базы даннях [Електронний ресурс] ИД Вильямс // С.45-49. - Название с титул. экрана. - Режим доступа: http://www.williamspublishing.com/PDF/978-5-8459-1157-5/part.pdf.
2. Методы улучшения качества кода: рефакторинг. [Електронний ресурс] - Название с титул. экрана. - Режим доступа: http://social.msdn.microsoft.com/Forums/ru-ru/fordesktopru/thread/63d9ba19-491b-4e75-9796-6de5132e1a56.
3. Robert L., Ira D., Michael Mehlich. Re-engineering C++ Component Models Via Automatic Program Transformation. - Название с титул. экрана. - Режим доступа: http://www.semanticdesigns.com/Company/Publications/WCRE05.pdf.