УДК 681.30.068

И.В. Шишова, *старший преподаватель,*
ФГБОУ ВО «ДГТУ», г. Махачкала, РФ

О НЕКОТОРЫХ ПРИЕМАХ И СПОСОБАХ РЕФАКТОРИНГА ИСХОДНОГО КОДА ПРОГРАММ

Рассматриваются некоторые приемы и способы рефакторинга исходного кода программы как средства улучшения его читаемости и композиционной стройности структуры. Объясняется значимость этого подхода и необходимость изучения его обучающимися по направлению программная инженерия.

Ключевые слова: рефакторинг программного кода, именование переменных, венгерская нотация

Consider some techniques and methods for refactoring source code of the program as a means of improving its readability and the compositional harmony of the structure. Explains the significance of this approach and the need to understand its learners in the direction of software engineering.

Keywords: refactoring, program code, naming variables, Hungarian notation

Программирование сравнительно молодая и быстро развивающаяся отрасль науки и техники. Опыт ведения реальных разработок и совершенствование имеющихся программных средств постоянно переосмысливается, в результате появляются новые подходы, методы, технологии программотехники. Одним из таких подходов является рефакторинг. Рефакторинг – это изменения, вносимые в существующую программу, чтобы сделать ее легче для понимания, дешевле для модификации и не изменяющие ее функционального поведения. Буквальный русский перевод этого термина – реорганизация. Такой перевод как раз подчеркивает, что изменениям подвергается именно структура программного кода, и эти изменения не влияют ни на функциональность, ни на скорость работы.

Понятие рефакторинга исходного кода впервые было введено в методологию программной инженерии в 1992 году Уильямом Опдайком. В дальнейшем благодаря работам Мартина Фоулера рефакторинг сформировался как формализуемый подход программотехники. М.Фоулер дал следующее определение термину рефакторинг.

Рефакторинг кода – это процесс внесения изменений в исходный код

программы посредством небольших преобразований, смысл которых эквивалентен прежним фрагментам, но результат выглядит более стройным, органичным и логичным [1].

Значимость этого подхода заключается в том, что современное программное обеспечение имеет очень большие объемы и очень сложную структуру и в его создании участвует целый коллектив разработчиков. Поскольку состав разработчиков может меняться, модификация кода может выполняться другим программистом без полного понимания его структуры. Вследствие этого ясность структуры кода теряется, что может привести к множеству проблем. Регулярное применение рефакторингов способствует сохранению структуры кода. В дальнейшем это окупится уменьшением времени на отладку при добавлении новых возможностей в хорошо структурированную программу. Для начинающих программистов и студентов, обучающихся по направлению программная инженерия, очень важно знать и владеть основными приемами и способами рефакторинга, чтобы сразу приучиться писать хорошо структурированные программы.

Целью данной статьи является описание некоторых приемов и методов рефакторинга, которые обязательно должны быть усвоены начинающими программистами, а также иллюстрация их эффективности.

Правильное наименование программных объектов

Именование программных объектов не имеет прямого отношения к рефакторингу исходного кода, но оказывает влияние на его читабельность. Для улучшения читабельности программы следует давать программным объектам осмысленные имена. Содержательное имя упрощает понимание и модификацию кода. Если имя требует дополнительных комментариев, то его лучше изменить, так как иначе придется дублировать комментарий при каждом обращении к имени.

Существует несколько видов нотаций, т.е. групп рекомендаций по формированию имен. Так согласно Венгерской нотации имена переменных рекомендуется начинать с префикса, соответствующего типу величины, например, iSumma – переменная целого типа (int) для хранения суммы. Нотация Паскаля рекомендует каждое слово, входящее в имя, начинать с заглавной буквы, например, MaxSumma. Нотация Camel отличается от предыдущей тем, что первое слово имени пишется со строчной буквы, а остальные начинаются с прописной, например, maxSumma. Каждый разработчик со временем вырабатывает свой стиль кодирования и использует свои правила именования, частично или полностью заимствованные из различных нотаций.

Для того, чтобы показать значимость использования правильных имен программных объектов, был проведен эксперимент с группой студентов. Группа была разбита на две подгруппы, каждому студенту был предоставлен текст небольшой программы, написанный на известном ему языке программирования. Студенты одной подгруппы получили текст программы, в котором использовались многобуквенные содержательные идентификаторы, вторая подгруппа получила тот же текст программы, но в нем использовались только однобуквенные идентификаторы. Студенты должны были по тексту программы составить блок-схему. Эксперимент показал, что студенты первой подгруппы справились с заданием гораздо быстрее и качественнее.

Достаточно общей задачей при модернизации программы является нахождение кодов, которые имеют несоответствующие или вводящие в заблуждение имена, потому что коды в результате изменений работают несколько иначе, чем это планировалось вначале. Студентам был предоставлен текст программы, выполняющей выборочную сортировку массива чисел по возрастанию элементов. Студенты одной подгруппы получили программу, в которой были использованы переменные с именами min и min_pos для хранения минимального элемента и его индекса. Студенты второй подгруппы получили программу, в которой были использованы переменные с именами max и max_pos для хранения минимального элемента и его индекса. Эксперимент показал, что все студенты первой подгруппы смогли составить блок-схему программы, тогда как только 80% студентов второй подгруппы смогли справиться с задачей.

Разбиение цикла

Рассмотрим функцию, написанную на языке C++. В этой функции с клавиатуры вводятся элементы массива и сразу определяется сумма вводимых элементов, чтобы затем рассчитать среднее значение.

```
void PrintMiddleValueOfArray()
{
    int i, m[10], Summa = 0;
   float  MiddleValue;
    cout << ”Input elements”;
    for(i = 0; i < 10; i++)
    {
       cin >> m[i]; Summa += m[i];
     }
   MiddleValue = (float)Summa/10;
    cout << “MiddleValue = “ <<
MiddleValue:
}
```

Имя функции и имена локальных переменных записаны согласно нотации Паскаля и содержат информацию об их назначении, что позволяет легко понять смысл кода. Цикл

включает два действия: ввод элемента и добавление его значения к сумме. Действительно, большинство программистов посчитало бы неудобным разделение такого цикла на части, поскольку это приведет к тому, что цикл выполнится дважды. Однако, цикл, выполняющий два действия, мене ясен, чем цикл, выполняющий только одно действие. Использование общего цикла также может вызвать трудности при дальнейшем разложении кода на части, поскольку в нем используются общие локальные переменные. Хотя выполнение цикла дважды приводит к дополнительным затратам времени, но в большинстве случаев это время не является критичным, к тому же на основе разбиения цикла могут быть выявлены другие, более мощные пути оптимизации. Это мотивирует проведение рефакторинга «Разбиение цикла».

Упрощение условных выражений

Это целая группа способов, которые позволяют упростить понимание сложных условий. Рассмотрим следующий условный оператор.

```
if ( Number == 1) cout << “\n One”;
    else if ( Number == 2) cout << “\n TWO”;
        else if ( Number == 3) cout << “\n Three”;
            else cout << “\n Zero”;
```

В данном фрагменте использован вложенный оператор if-else-if. Для того чтобы улучшить читабельность кода использованы отступы, но этого недостаточно. В данном случае упрощение может быть достигнуто использованием другого оператора выбора.

```
switch (Number)
{case 1: cout << “\n One”; break;
 case 2: cout << “\n TWO”; break;
 case 3: cout << “\n Three”; break;
 default: cout << “\n Zero”;
}
```

К этой же группе рефакторингов относятся такие способы, как объединение условий, возвращающих один результат, или избавление от флагов, т.е. переменных, которые получают определенное значение (обычно 1) при свершении некоторого события, и замена их операторами break или continue.

Все рассмотренные модификации кода можно считать мелкими и нетрудоемкими, однако их применение значительно улучшит структуру программы при полностью сохраненной функциональности. Главным же результатом является то, что рассмотренные способы рефакторинга позволяют значительно улучшить читабельность кода и существенно облегчить сопровождение программы и интеграцию в нее новых функциональных возможностей.

Литература

1. Fowler M. Refactoring. Improving the Design of Exiting Code. – Boston: Addison-Wesley, 1999.

2. Реинжиниринг и рефакторинг программного обеспечения: Учеб. пособие/ С.А.Романенко, С.В.Савосин, А.В.Спицын, П.Б. Фельдман. – СПб.:Изд-во СПбГЭТУ «ЛЭТИ», 2002

3. А.Л. Калабин, Е.Н. Грязнов. Рефакторинг исходного кода. Основные приемы// Программные продукты и системы. 2013. №2

УДК 004.42

Х.С Ясулова, *Дагестанский государственный педагогический университет*

ПОСТРОИТЕЛЬ КОМПЬЮТЕРНОГО СЛОВАРЯ

В статье приводится пример построителя, который выступает как источник информации и помощник при построении различных слов и выражений, основной функцией которой является построения компьютерного словаря.

Ключевые слова: «построитель», «математическая модель», «компьютерная модель», «граф», «сеть», «морфология», «синтаксис», «семантика», «словосочетание», «синтаксический анализатор», «морфологический анализатор» модели синтаксиса».

The article provides an example of a Builder that acts as a source of information and assistant to build different words and expressions which main function is building computer dictionary.

Keywords: builder, mathematical model, computer model", "graph", "network", "morphology", "syntax", "semantics", "phrase", "parser", "morphological analyser" model syntax".

Термин *построитель* (выражений, объектов и т.д.) широко используется в различных приложениях при визуальном программировании. Построитель выступает как источник информации и помощник при построении различных слов и выражений. Построитель компьютерного словообразовательного словаря русского языка должен быть важной составной частью (службой) текстового редактора и любого анализатора предложений ЕЯ.

В русском языке имена и глаголы являются изменяемыми частями речи. Имена существительные имеют наименьшее число форм, но не так просто по исходной форме слова определить все 12 его форм. Необходимость в наличии такой процедуры возникает в тех случаях, когда при наборе или анализе текста встречается слово, отсутствующее в словаре. Предположим, что таким словом оказалось имя «доска». Приложение