**Рефакторинг – необходимость или мода?**

### Михаил Фленов ([www.flenov.net](http://www.flenov.net))

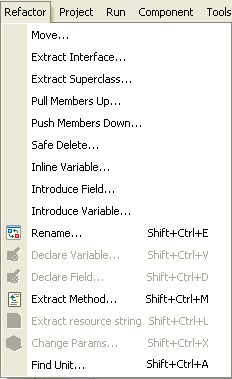
Что такое рефакторинг? Я видел достаточно много определений этому понятию, но все они сводяться к улучшению существующего кода. Если ты пишешь только идеальный код, который нужно улучшать только в тех случаях, когда он не работает, то сильно заблуждаешся. Улучшения нужны даже тогда, когда код работает вполне корректно. Для чего, когда и как нужно улучшать мы поговорим в этой статье.

Что можно улучшать в коде, который и так уже работает и выполняет возложенные на него функции? Если программу не планируется улучшать и добавлять новые возможности, то можно больше уже ничего не улучшать. Лучше даже удалить исходники, дабы не пытаться разбираться в бардаке или использовать его в будущем. Но если программа нужна не один день, то рефакторинг необходим.

Ты хотя бы иногда убираешь на своем рабочем столе? Хоть иногда чистишь компьютер от мусора и от не используемых программ? Ну и наконец, убираешь в квартире, чтобы не ходить по мусору? Последний пример может быть и не очень хороший, потому что убираться в квартире большинство из нас не любит (сбрасывая это занятие на маму/жену/подругу), но ходить по мусору и жить в бардаке уж точно не приятно.

То же самое касается и кода. Если он написан хорошо и легко читается, то его приятно сопровождать, добавлять новые функции или оптимизировать. Такой мод можно даже использовать в других проектах. Но если код написан ужасно, то намного приятнее написать все с нуля. Из-за плохого кода лет пять назад я забросил один из любимых своих проектов и переписал заного. Этот проект я начинал еще в 1996-м году и ниразу не задумывался о рефакторинге. Тогда даже понятия такого не знал (возможно, его и небыло). Получив серьезный ожег я теперь на всех этапах написания кода задумываюсь о его улучшении и при первом же появлении мусора улучшаю и очищаю код.

Важность рефакторинга подчеркивает и то, что во всех последних версиях сред разработки (Delphi 2005/2006, JBuilder 9 и выше, Visual Studio 2005) появились различные мастера и функции для улучшения кода. Эти функции не могут охватить все сферы рефакторинга, да и без знания основ их использовать проблематично.

  
Меню Refactor в Delphi 2006

В этой статье я постараюсь дать основные принципы улучшения кода из личного опыта. Это личный многолетний опыт и кто-то может с ним не согласиться, а кто-то найдет для себя что-то новое.

**Комментарии**

Большинство из нас во время написания кода никогда не задумывается о комментариях и не любит писать их. Да, хорошо написанный и оформленный код должен читаться без комментариев, но все же, небольшие пояснения никогда не помешают. После написания метода я стараюсь выделить пару минут своего драгоценного времени на комментарии, которые в последствии позволят мгновенно понять, что я натворил.

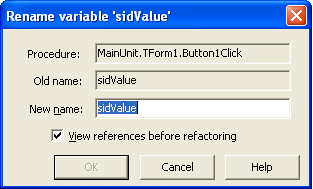
К комментариям можно отнести отдно простое правило – они должны быть короткими и понятными. Если среда разработки поддерживает комментарии TODO, то не стесняйся использовать их, они реально помогают.

**Читабельность**

Многие из программистов абсолютно не обращают внимание на имена переменных. Когда порграмма состоит из 1000 строк кода, то понять о назначении переменной не сложно. Но когда исходный код исчисляется 5 тысячами строк и более, начинаются серьезные проблемы. Особенно через годик, после написания кода. Спросите программиста, что может храниться в переменной Temp, Str или param? Первое, что приходит в голову – там хранится отходы жизнидеятельности человека, которые мы сбрасываем в туалете.

Если по имени переменной нельзя понять, для чего она и что хранит, то на чтение кода приходится тратить лишнее время. Необходимо сначала найти место, где объявляется переменная, а потом определить, что в нее записывается. А иногда, без бутылки пива с кодом разобраться просто невозможно.

А что если переменная называется iFileLength? Вот тут уже легко понять, что это целочисленная переменная Integer или int (в зависимости от языка) и она содержит длину файла. Чтение такого кода и сопровождение значительно упрощается.

  
Окно переименования переменных в Delphi 2006 (улучшение кода переименованием)

О переменных можно говорить очень много, но я кратко дам несколько основных правил, которых желательно придерживаться (у каждого могут быть свои подходы к именованию, поэтому мой вариант не стоит считать идеальным):

* Имя переменной должно содержать какой-либо префикс, который укажет нам на тип данных;
* Имя должно быть осмысленное и без лишних пояснений должно быть понятно, что за ним скрывается.
* Одна переменная никогда не должна выполнять сразу два действия в одном и том же блоке кода. Например, внутри одной процедуры переменная iFileLength не должна сначала содержать длину файла, а потом использоваться в качестве счетчика в цикле. В последствии, очень просто забыть этот нюанс и неправильно использовать значение переменной, когда там уже совершенно другое значение. А это уже приведет к проблемам безопасности.

Для улучшения переменных, необходимо банально дать им всем понятные имена и убедиться, что каждая из них используется только для одной цели.

Да, лишняя переменная отнимает место в памяти, но если она имеет простой тип, то это не смертельный размер и минимальная плата за простую, но очень хорошую защиту от неправильного использования переменных. Прикинь сейчас, какие ошибки ты чаще всего исправляешь? Лично я чаще исправляю собственную невнимательность и большинство, я думаю, тоже. Если следовать описанным выше правилам, вероятность ошибок из-за невнимательности уменьшается.

**Методы**

Я думаю, не стоит уже говорить о том, что имя метода должно быть понятным. Понятными должны быть все имена. Давайте поговорим о нюансах, которые относятся именно к методам/процедурам/функциям.

Каждый метод должен выполнять одну и только одну функцию. Лучше всего, если эта функцая будет максимально узкой. Например, если мы создаем метод загрузки файла, то метод должен только загружать файл, а анализ и другие возможности необходимо реализовывать в других методах. Но при этом, нельзя отделять такие функции, как проверка корректности и безопасности. Их выносить в отдельный метод очень сложно, иногда невозможно, а в большинстве случаев глупо. Реализация и проверка корректности неразделимы, поэтому, в методе загрузки файла нужно не забыть проверить наличие файла, корректность его открытия, доступность данных, размер буфера для чтения и т.д.

Как же мы любим объединять весь возможный код в одном методе, а ведь это грозит нам следующими проблемами:

* метод очень сложно читать, даже при наличии большого числа и подробных комментариев;
* код сложнее использовать повторно. Когда метод выполняет узкую задачу, то его можно использовать в другом месте программы, где необходимы те же расчеты. Если метод решает несколько задач, то вероятность встретить необходимость выполнить все то же самое намного ниже.
* усложняется отладка метода, а в больших проектах отладка и тестирования отнимает достаточно много времени и сил.

Проблем у больших и универсальных методом очень много, но эти пункты я бы выделил в качестве основных. Если методы выполняют узкие задачи, то мы избегаем всех этих узких мест.

У данного совета могут быть противники, потому что избыточные вызовы методов – снижение скорости. Да, вызов каждой процедуры требует лишних расходов, особенно, если она получает много параметров. А если процедура за время выполнения программы будет вызываться сотни, а то и тысячи раз, то это уже серьезный удар по производительности. Да, лишние методы не нужны, но когда приходиться выбирать между качеством кода и оптимизацией, я всегда выбираю первое и всем советую действовать так же.

**Улучшение методов**

В случае с методами уже банальным переименованием не обойтись. Тут уже приходиться вмешиваться в код более координально. Если функция получилась очень большой или бросается в глаза выполнение нескольких задач, необходимо разбить ее на несколько более маленьких функций.

В Delphi 2005 появилась достаточно интеллектуальная функция Refactor/Extract Method. В JBuilder та же функция спрятана в меню Edit/Extract Method. Чтобы воспользоваться ею, необходимо выделить отрывок кода, который нужно выделить в отдельный метод и выбрать указанный пункт меню. Перед нами открывается диалоговое окно, в котором достаточно ввести имя нового метода. Все остальное среда разработки сделает сама, а именно:

1. Будет создан и корректно объявлен новый метод.
2. Вместо кода, который мы выделили будет вставлен вызов вновь созданного метода.

Улучшение может потребоваться и в тех случаях, когда схожый код встречается в нескольких методах. Пусть они решают и узкую задачу, но повторение кода не есть хорошо. Намного лучше выделить отдельный метод, это упростит сопровождение программы.

**Размер метода**

Многие авторитетные программисты считают, что методы должны быть максимально короткими и если код не помещается на экран, то его необходимо разбит на несколько методов. Прекрасное правило и я его поддерживаюсь в 99% случаев и только 1% моих функций не помещается на экран. Если метод решает одну и только одну задачу, то не стоит его делить на несколько, даже если он занимает два экрана. Просто купи монитор побольше :). Это конечно же шутка, монитор побольше не нужен. Достаточно один раз отладить задачу и забыть про нее.

**Видимость методов**

Видимость методов также относиться к рефакторингу. Ни один лишний метод не должен быть виден другим классам для прямого вызова. Дабы не следить за видимостью, я всегда создаю все методы закрытыми и только если какой-то из них понадобилось вызвать из вне, я делаю его открытым.

Когда ты заканчиваешь работу над программой, просмотри все классы и видимость их методов. Возможно, какой-либо метод ты сделал открытым, а потом нашел другой способ решения задачи. Не помешает закрыть этот метод. Да, на производительности программы это не скажеться, но код станет лучше и приятнее.

**Читабельность метода**

Язык С++ великолепен, и позволяет нам одну и ту же операцию записать несколькими способами. Программисты на других языках так же могут записывать определенные операции в одну строку. Если строка кода становиться не читабельной, то следует задуматься, а не разбить ли ее на две строки?

Длиные строки кода очень тяжелы для восприятия. Лучше всего, если строка будет помещаться на экран полностью. Если с вертикальной прокруткой для просмотра метода целиком можно смериться, то горизонтальная неудобна и с ней необходимо бороться.

**Классы**

Помимо хорошего именования классов, они также как и методы должны решать узкую задачу. Не стоит создавать один класс, который будет решать задачу дома, сарая и гаража одновременно. Лучше выстроить иерархию из нескольких классов (возможно, с одним базовым, который будет содержать общие методы и свойства).

В JBuilder и Delphi 2006 есть очень удобная возможность выделения методов в отдельный класс или интерфейс. Эти возможности спрятаны в меню Refactor/Extract Interface и Refactor/Extract superclass. Допустим, ты написал класс, который характеризует дом. Затем в программе понадобилось создать гараж. Так как некоторые характеристики гаража будут такими же, как и для дома, можно выделить их в суперкласс, а затем наследовать от него гараж, сарай и другие строения со схожими параметрами.

**Улучшение классов**

Что еще можно сделать, чтобы классы стали лучше? Тут уже нужен более глубокий анализ на уровне методов. Необходимо выяснить, действительно ли методы относяться к задаче, решаемой классом. Если нет, то метод необходимо вынести за пределы класса. Куда, тут мы ничего хорошего посоветовать не можем, потому что все зависит от программы.

Если метод принимает очень много параметров, то это не есть хорошо. Посмотри, возможно некоторые параметры можно хранить в качестве членов класса. Да, в этом случае понадобяться методы для установки значения данного члена (ничего пошлого, так называют свойства классов), но если данный параметр нужен в нескольких методах, то эти затраты оправданы на все 100%.

**Форматирование кода**

Оформление кода также можно отнести к рефакторингу. Тут у программистов Visual Studio немного больше преимуществ. Чего стоит меню Edit/Advanced. Тут есть все необходимое, чтобы сделать код более удобным для чтения. В Delphi тоже есть большинство из этих функций, но они по умолчанию спрятаны, но зная горячие клавиши или если вынести соответствующие кнопки на панель, то ими можно воспользоваться.

О том, как оформлять код написано уже много. Лично мое мнение – он должен быть оформлен так, чтобы тебе было удобно читать. Если для понимания отдельного метода или отдельной строки приходиться применять усилия, то форматирование неудачное и следует задуматься о его улучшении.

Современные среды разработки автоматически форматирую код. Visual Studio, JBuilder делают это уже давно, а теперь и в Delphi появились подобные возможности. Но настройки JBuilder пошли дальше. Здесь можно выбрать где и как должны располагаться скобки { и }. Существуют и отдельные мастера для разных сред разработки, которые автоматически отформатируют код.

**Структура кода**

Банальная сортировка методов также может повысить читабельность. Как сортировать методы? Это зависит от многих факторов, но желательно, чтобы порядок методов соответсвовал заранее определенному правилу. Какие могут быть порядки:

1. По алфавиту – банальный но очень удобный порядок, позволяющий быстро находить место, где должен находиться нужный метод. Рекомендую использовать его, когда модуль/класс очень большой и содержит очень много методов;

2. По типу. Сначала можно расположить методы, устанавливающие значения свойств, затем производящие расчеты, а потом, выводящие информацию на экран.

Возможно, ты предпочтешь другую сортировку, главное, чтобы она была удобной и позволяла быстро найти нужный метод. В современных средах разработки есть специальные окна, которые упрощают поиск, но при наличии сортировки, поиск будет еще проще.

**Рефакторинг и безопасность**

Если посмотреть на рефакторинг, то он не вносит в код ничего сверх естественного. Безопасность остается на том же уровне, а производительность может даже упасть. Но это только на первый взгляд. На самом деле, безопасность только растет, пусть косвено, но очень качественно:

* Во время рефакторинга ты лучше понимаешь код программы и в этот момент можешь найти недочеты в логике;
* Сделав код более понятным, его легче отлаживать и находить недочеты.
* Если метод или класс выполняет узкую задачу, то намного сложнее совершить ошибку.

**Refactoring complete**

[](http://www.books.ru/shop/books/330247?partner=horrific)

[Купить](http://www.books.ru/shop/books/330247?partner=horrific)

За отведенное мне место мы смогли рассмотреть только базовые понятия рефакторинга, но если следовать им, то ты сможешь реально повысить качество кода. Попробуй и уже скоро ты увидишь, что рефакторинг реально помогает. На банальное улучшение никогда не хватает времени, но если немного выделить этого времени, то код станет намного лучше и с ним будет приятно работать в будущем.

Настоятельно рекомендую почитать книгу Совершенный код от С. Макконнелла. С некоторыми тезисами я могу поспорить, но в целом книга великолепна. В своих книгах я тоже затрагивал рефакторинг, но их рекламировать не буду. Читая Макконнелла я бы не советовал следовать его совету программировать с использованием языка, а не на языке. Невозможно превратить Delphi в Java и наоборот. Это лично мое мнение, а как последуете вы, выбор только за вами.

О рефакторинге очень громко заговорили совсем недавно, когда во главу стали ставить не производительность, а качество и безопасность. Рефакторинг позволяет улучшить эти паказатели, хотя на первый взгляд это не заметно.Рефакторингом нужно заниматься сразу же в процессе разработки программы, а также по завершении работы над проектом. Мусор в коде имеет тенденцию к быстрому увеличению, поэтому лучше убирать сразу и не дожидаться катастрофы.