удк 629.12.001.2

Применение рефакторинга при модернизации программного обеспечения в судостроении

Аспирант. Звягин Константин Николаевич
СПБГМТУ
190008, Санкт-Петербург, ул. Лоцманская, 3 e-mail: 4250046@gmail.com

В работе освещены актуальные проблемы при применении механизма рефакторинга в судостроении к программам автоматизации и алгоритмам написанных на не объектно-ориентированных языках . В мире существует множество программ не поддающихся расширению и модернизации из-за того, что технологии с помощью которых они были написаны устарели. Рефакторинг позволяет эту проблему решить.

В современном мире существует множество удовлетворительно работающих технических объектов, программное обеспечение которых писалось 20-30 лет назад с использованием имевшихся на тот момент сред, алгоритмов, системных и программных возможностей. Это положение особенно актуально для судостроения, в котором во время и после дефолта были приостановлены проектно-конструкторские и научные разработки.

Поэтому на сегодняшний день в практике программирования сложилась ситуация, когда программы, написанные для судостроительных организаций порядка 20 лет назад методами, которые были доступны и общеизвестны на тот момент, устарели как по некоторой части технических данных, так и по сервисным возможностям. Персонал, приходящий из вузов, сейчас зачастую не понимает структуры старых программ. Это связано не только с утерей документации на них, но и с изменением учебных программ вузов, которые потеряли некоторые технические вопросы из-за увеличившейся гуманитарной части.

Тем не менее, ранее созданные судостроительные программы зачастую невозможно полностью вывести из эксплуатации, так как они обслуживают устоявшиеся технические процессы. В этом случае на помощь приходит механизм рефакторинга - улучшения существующего кода. Принципы рефакторинга разработаны в последние десятилетия.

Термин рефакторинг означает изменение исходного кода программы без изменения его внешнего поведения. В основе рефакторинга лежит выполнение последовательности небольших локальных преобразований программы. Эти изменения должны быть эквивалентны старому коду, то есть должны сохранять поведение программы. Каждое преобразование при рефакторинге небольшое, поэтому программисту легче проследить за его правильностью. Но в целом весь набор изменений может привести к значительной перестройке программы и к улучшению её ясности и надежности.

Основные причины, требующие применения рефакторинга, это:

- необходимость добавить новую функцию;
- необходимость исправить ошибку, причины появления которой не всегда ясны до конца;
- преодоление трудностей, связанных со сложной логикой программы;
- необходимость подключения нового персонала к эксплуатации написанного ранее программного обеспечения;
- преодоление трудностей, появляющихся при командной модернизации развитого программного обеспечения.

При использовании рефакторинга в судостроении мы сталкиваемся с рядом проблем, из-за которых рефакторинг может быть выполнен некачественно.

Первая проблема - это рефакторинг, приводящий к увеличению потребления программой памяти и ресурсов процессора. Она может появиться, когда дописываются излишние объектные оболочки и/или плохо реорганизуются данные. Эта же проблема может возникнуть не только при простом рефакторинге, но и при модификации или модернизации имеющегося программного обеспечения. Такая проблема иногда возникает у недавних выпускников вузов, применяющих мощные современные возможности программирования, не имевшиеся ранее, к решению не слишком сложных по своей логике

задач.

Вторая проблема некачественного рефаторинга - это оставшееся незамеченным изменение внутренних последовательностей операций, а значит, и алгоритма. Это приводит к тому, что программное обеспечение (ПО) просто становится невозможным эксплуатировать. Обычно возникают следующие особенности:

- непредсказуемость поведения на разных объектах, которую можно устранить только предварительным реальным тестированием, которое не всегда возможно организовать,
- потребность в проведении новых пуско-наладочных работ, которые могут быть очень затратными в связи с наличием большого числа контрагентов в судостроении,
- необходимость заново проходить лицензирование, сертифицирование, военную приемку и прочие правоустанавливающие и согласовательные процедуры,
- необходимость переучивания персонала, эксплуатирующего ПО, в связи с изменением представления и компоновки входных и выходных данных, изменением (увеличением или сокращением) объема текста, изменением времени.

Рассмотрим теперь требования, которые необходимо или очень желательно выполнить при рефакторинге:

- объем используемой ПО памяти должен либо сократиться, либо остаться прежним,
- время выполнения программы должно также либо остаться прежним, либо сократиться,
- организация данных должна быть проста и понятна эксплуатирующему ПО инженеру,
- должна иметься возможность несложного добавления новых и/или изменения имеющихся в программе методов и данных,
- должно обеспечиваться как можно более полное соответствие исходному поведению программы,
- текст должен быть хорошо прокомментирован,
- должен вестись учет версий данного ПО и обеспечены удобные возможности для обращения эксплуатационников к более поздней или более ранней версии ПО.

Можно отметить некоторые причины, при которых программный код следует подвергнуть рефакторингу:

- дублирование фрагментов кода;
- длинный метод;
- большой класс;
- длинный список параметров;
- метод, который излишне часто обращается к данным другого объекта;
- избыточные временные переменные, требующие лишнюю память;
- неаккуратно написанные классы данных;
- плохо организованные данные.

Существует ряд традиционных методов, которыми проводится рефакторинг. Среди них можно назвать изменение сигнатуры метода, инкапсуляцию поля, выделение метода, перемещение метода, замену условного оператора полиморфизмом и другие.

Выполняя рефакторинг программного обеспечения для корабельных энергоустановок можно выделить некоторые основные вехи в процессе работы. Во- первых, это части исполняемого кода, имеющие дело со временем: например, регулирование и время основного цикла программы. Сюда относятся также менее важные события, связанные с предупредительной сигнализацией и информированием об этом оператора. Иногда можно встретить крайне некорректные попытки применения методов рефакторинга, например в случае, когда регулирование происходит по ходу алгоритма, перемежаясь фрагментами основного алгоритма и другими проверками аппаратуры или событий. В таких случаях первозданная программа может вести себя корректно, так как программист, написавший ее, мог отталкиваться от времени, затрачиваемого центральным процессором на каждую операцию, и нехитрым суммированием получить примерно соответствующие действительности временные промежутки. Однако простой перенос данной программы на центральный процессор (ЦП) большей или меньшей вычислительной мощности сделает ее работу некорректной. Аналогично, запуск дополнительных задач на многозадачной встраиваемой операционной системе даст примерно такой же результат.

Подход к такому ПО, широко распространенному в судостроении, должен быть достаточно осторожным. Для этого необходимо отследить временные промежутки, используемые в старом ПО, и реализовать на базе внутренних таймеров нужные временные промежутки, которые подвергаются дальнейшему рефакторингу , например для вынесения в отдельный класс, а затем и в поток процессов регулирования. Это является одним из необходимых требований плавного и адекватного регулирования в ПО, предназначенном для множества проектов с различной вычислительной мощностью ЦП.

Во-вторых, судостроительное ПО зачастую содержит фрагменты кода, отвечающие за параллелизм и

«горячую» замену работающих модулей. Здесь надо быть особенно осторожным с последовательностью выполнения методов и команд, так как небольшие перестановки могут привести к потере синхронизации и сделают невозможной «горячую» замену модулей управления энергоустановкой. А это может повлечь на реальном объекте аварию глобальных масштабов.

В третьих, специальное внимание следует уделять реорганизации данных и временных переменных. Бывают случаи, когда программа состоит из одного метода main, в котором, собственно, и реализуется весь алгоритм. В этих случаях необходимо обратить внимание на переменные, которые используются в нескольких различных смысловых кусках когда. Скорее всего они отправятся в поля классов. Однако переменные, используемы кратковременно, (например, для обмена значениями между переменными) следует выделить в переменные метода в классе. Это даст нам экономию памяти. Переменные же, используемые на протяжение всего алгоритма, можно вынести в базовый класс, который станет родительским для всех остальных. Это упростит взаимодействие между методами классов и увеличит скорость выполнения, то есть даст нам экономию времени.

Подводя итог, можно сказать, что рефакторинг является неотъемлемой работой каждого программиста. Большую часть существующего ПО обычно можно подвергнуть рефакторингу и получить улучшения по скорости выполнения, проценту использования ЦП и по понятности кода. В заключении отметим также, что ПО часто требует нескольких циклов проведения рефакторинга. Затем проект, приведенный с помощью рефакторинга в надлежащий вид, можно с успехом использовать в течение дальнейшего времени..

Литература

1. Белладжио Д., Миллиган Т. Разработка программного обеспечения, изд. ДМК Пресс, 2009, 348с.
2. Кириевски Дж. Рефакторинг с использованием шаблонов. Изд. Вильямс, 2006, 400с.