Лабораторная работа №1

Рефакторинг (англ. refactoring), или перепроектирование кода, переработка кода, равносильное преобразование алгоритмов — процесс изменения внутренней структуры программы, не затрагивающий её внешнего поведения и имеющий целью облегчить понимание её работы. В основе рефакторинга лежит последовательность небольших эквивалентных (то есть сохраняющих поведение) преобразований. Поскольку каждое преобразование маленькое, программисту легче проследить за его правильностью, и в то же время вся последовательность может привести к существенной перестройке программы и улучшению её согласованности и чёткости.

Рефакторинг следует отличать от оптимизации производительности. Как и рефакторинг, оптимизация обычно не изменяет поведение программы, а только ускоряет её работу. Но оптимизация часто затрудняет понимание кода, что противоположно рефакторингу.

С другой стороны, нужно отличать рефакторинг от реинжиниринга, который осуществляется для расширения функциональности программного обеспечения. Как правило, крупные рефакторинги предваряют реинжиниринг.

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

1. необходимо добавить новую функцию, которая недостаточно укладывается в принятое архитектурное решение;
2. необходимо исправить ошибку, причины возникновения которой сразу не ясны;
3. преодоление трудностей в командной разработке, которые обусловлены сложной логикой программы.

Рефакторинг изначально не предназначен для исправления ошибок и добавления новой функциональности, он вообще не меняет поведение программного обеспечения и это помогает избежать ошибок и облегчить добавление функциональности. Он выполняется для улучшения понятности кода или изменения его структуры, для удаления «мёртвого кода» — всё это для того, чтобы в будущем код было легче поддерживать и развивать. В частности, добавление в программу нового поведения может оказаться сложным с существующей структурой — в этом случае разработчик может выполнить необходимый рефакторинг, а уже затем добавить новую функциональность.

Запахи кода — это ключевые признаки необходимости рефакторинга. Существуют запахи, специфичные как для парадигм программирования, так и для конкретных языков. Основной проблемой, с которой сталкиваются разработчики при борьбе с запахами кода, является то, что критерии своевременности рефакторинга невозможно четко формализовать без апелляции к эстетике и условному чувству прекрасного. Запахи кода это не набор четких правил, а описание мест, на которые нужно обращать внимание при рефакторинге. Они легко обнаруживаются, но при этом не во всех случаях свидетельствуют о проблемах.

Код с запашком ведет к распаду кода, разработчики должны стремиться к устранению запашков путём применения однократного или многократного рефакторинга. В процессе рефакторинга происходит избавление от запахов кода, что обеспечивает возможность дальнейшего развития приложения с той же или большей скоростью. Отсутствие регулярного рефакторинга с течением времени способно полностью парализовать проект, поэтому запахи кода необходимо устранять на ранних стадиях. Существуют инструменты поиска и исправления запахов кода, однако опыт показывает, что никакие системы показателей не могут соперничать с человеческой интуицией, основанной на информации.

К запахам кода можно отнести следующие моменты:

- Имена переменных. В данном случае необходимо помнить простое правило: Чем крупнее проект, тем более говорящими должны быть имена переменных.

- Комментарии в коде. Наличие громоздких комментариев может свидетельствовать о том, что имена методов и переменных не раскрывают сути происходящего в программе.

- Дублирование кода. Если в программе встречается набор повторяющихся процедур, то это может свидетельствовать о необходимости создания метода.

- Соответствие типов. Следует отслеживать оправданность использования тех или иных типов данных. К этому разделу относится, как использование излишне точных типов (к примеру, тип double в случае, когда точности float было бы достаточно), так и преобразование типов с возможными потерями (читай присвоение типа с плавающей запятой целочисленной переменной).

Задание.

В файле Source.cpp приведен код программы, который необходимо проверить на наличие «запахов» и устранить их.