Построение хороших программ

- *Убедитесь в том, что поведение программы соответствует требованиям заказчика*
- *Применяйте базовые ОО-принципы для повышения гибкости*
- *Постарайтесь создать структуру кода, упрощающую его сопровождение и повторное использование*

Концепции

- *Абстракция*
- *Инкапсуляция*
- *Полиморфизм*
- *Наследование*
- *Делегирование*

Принципы ООП

- *Инкапсулируйте то, что изменяется*
- *Предпочитайте композицию наследованию*
- *Программируйте на уровне интерфейсов*
- *Стремитесь к слабой связанности взаимодействующих объектов*
- *Классы должны быть открыты для расширения, но закрыты для изменения*
- *Код должен зависеть только от абстракций, а не от конкретных классов*
- *Взаимодействуйте только с «друзьями»*
- *Не вызывайте нас – мы сами вас вызовем*
- *Класс должен иметь только одну причину для изменения*
- *Классы создаются ради поведения и функциональности*

Анализ и проектирование

- *Хорошо спроектированный код легко изменяется и расширяется*
- *Используйте базовые ОО-принципы (инкапсуляцию и наследование) для повышения гибкости*
- *Если структура кода недостаточно гибка, ИЗМЕНИТЕ ЕЕ ! Не используйте плохую архитектуру, даже если она сделана вами*
- *Классы должны обладать высоким сцеплением; каждый класс должен хорошо делать что-то одно*
- *На протяжении всего жизненного цикла структуры кода стремитесь к повышению сцепления*

Требования

- *Хорошие требования помогают привести программную систему в соответствие с ожиданиями заказчика*
- *Проследите за тем, чтобы требования охватывали все шаги всех вариантов использования программной системы*
- *Варианты использования помогают узнать о программной системе то, о чем забыл упомянуть заказчик*
- *Анализ вариантов использования выявляет неполные и пропущенные требования, которые, возможно, придется добавить в программную систему*
- *Требования всегда изменяются (и растут) со временем*

Решение больших задач

- *Выслушайте заказчика, чтобы понять какая программная система ему нужна*
- *Постройте список функциональных возможностей на языке, понятном заказчику*
- *Убедитесь, что функциональные возможности соответствуют нуждам заказчика*
- *Постройте общие планы системы с помощью диаграмм вариантов использования (и вариантов использования)*
- *Разбейте большую систему на множество небольших частей*
- *Применяйте паттерны проектирования к фрагментам программной системы*
- *Используйте ООАП при работе с фрагментами программной системы*

SOLID

- *Принцип единственной ответственности (SRP) : каждый объект должен иметь только одну обязанность, полностью инкапсулированную в класс*
- *Принцип открытости/закрытости (OCP): программные сущности должны быть открыты для расширения, но закрыты для изменения*
- *Принцип подстановки Лисков (LSP) : Сущности, использующие базовый тип, должны иметь возможность использовать подтипы базового типа, не зная об этом*
- *Принцип разделения интерфейса (ISP) : Клиенты не должны зависеть от методов, которые они не используют*
- *Принцип инверсии зависимостей (DIP) : Абстракции не должны зависеть от деталей, но детали должны зависеть от абстракций*

Принцип «Не повторяй себя» (DRY) : Старайтесь избегать кода с дублирующейся функциональностью