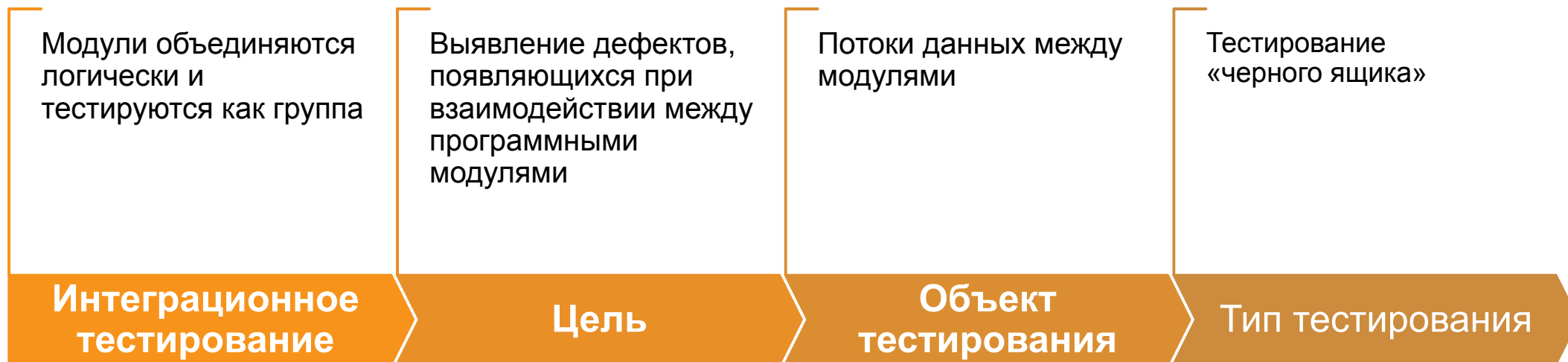


- 
- Интеграционное тестирование
 - TDD
-

Основные понятия



Необходимость ИТ

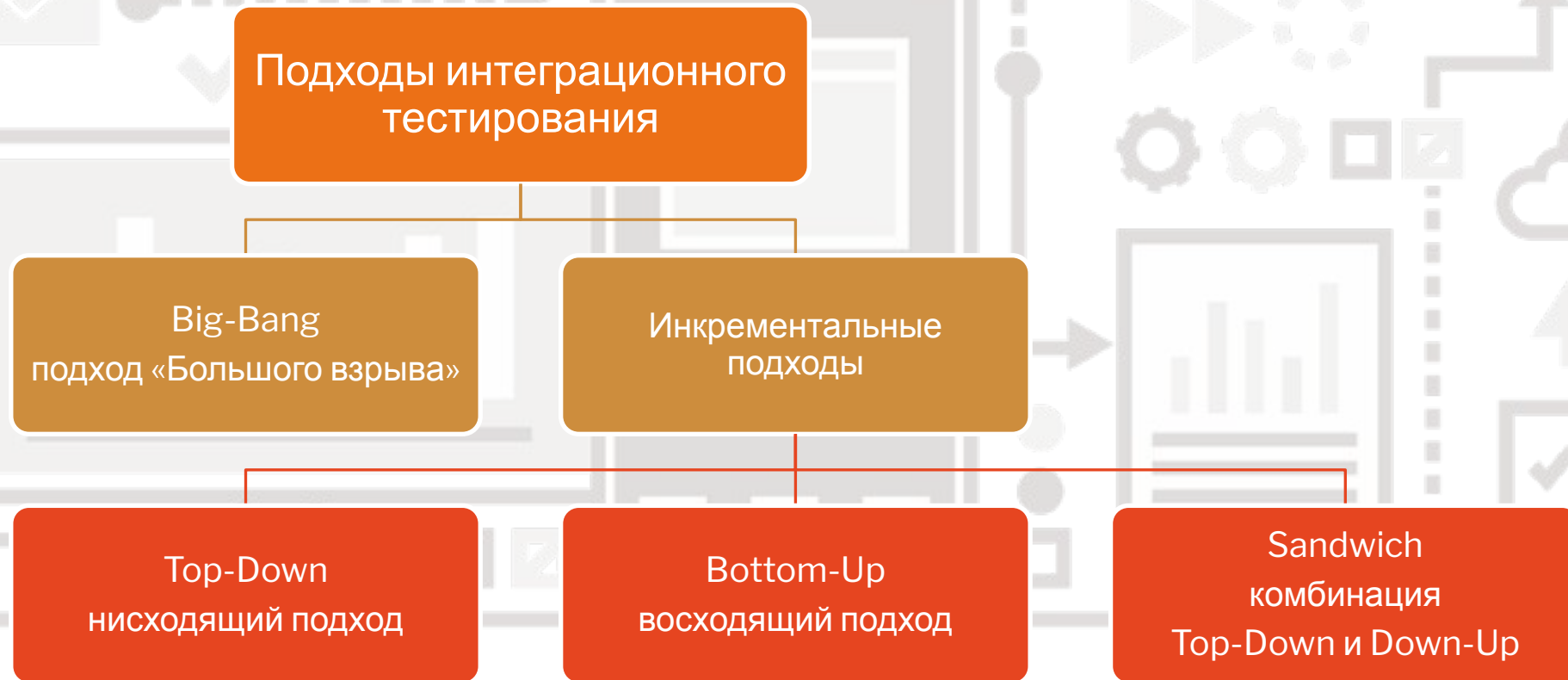
Интеграционное тестирование целесообразно для:

- Проверка взаимодействия модулей между собой;
- В условиях сжатых сроков или часто меняющихся требований позволяет пренебречь модульным тестированием;
- Проверка интерфейсов взаимодействия интерфейсов с источниками данных;
- Проверка аппаратных интерфейсов;
- Проверка корректности трактовки данных;



INTEGRATION TESTING

Применяемые подходы



Big-Bang ПОДХОД

Все модули собираются в единую систему и тестируются.

Подход оправдан только для малых проектов.

Достоинства:

- Значительная экономия времени;

Недостатки:

- Возможность пропуска тестирования отдельных связей;
- Требуется готовности всех модулей;
- Не осуществляется приоритезация процесса;
- Трудности локализации дефектов;
- Высока вероятность лавинообразного эффекта;





Инкрементальный подход

Тестирование не зависит от готовности всех блоков. Дефекты локализуются в модулях.

Данный подход включает три разновидности:

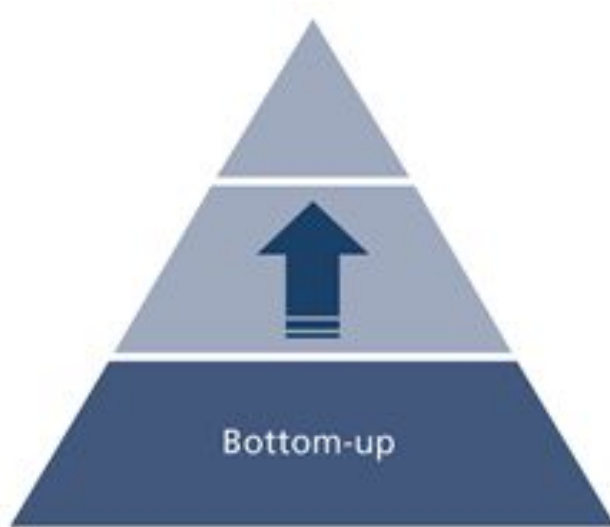
- Top-Down;
- Bottom-Up;
- Sandwich (hybrid) – совмещает оба подхода;

Тестирование осуществляется путем объединения двух и более модулей и последующим увеличением числа возвлеченных. Процесс осуществляется до полной сборки системы.

Инкрементальное тестирование определяет два вида фиктивных компонентов, предназначенных для имитации процесса обмена данными:

- **Заглушки** – компонент, вызываемый тестируемым модулем;
- **Драйверы** – компонент, который вызывает модуль для тестирования;

Bottom-Up подход



Каждый модуль низкого уровня (менее критичного) тестируется с модулями более высоких связанных с ним уровней (более критичных), пока не будет протестирована вся система.

Тестирование основано на работе с драйверами.

Достоинства:

- Модульная локализация ошибок;
- Возможность процентной оценки готовности системы:

Недостатки:

- Невозможно предоставить рабочий прототип на ранних этапах разработки;
- Процесс тестирования модулей высокого уровня откладывается;

Top-Down подход

Тестирование начинается с модулей высокого уровня.

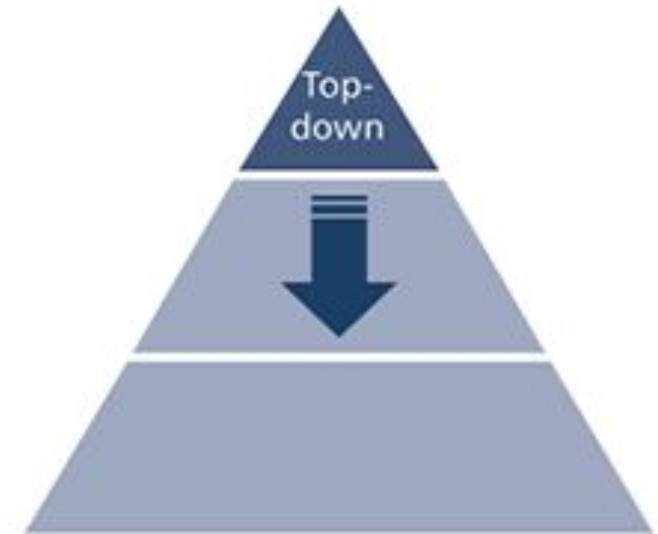
Для взаимодействия с нижестоящими модулями применяются заглушки.

Достоинства:

- Получение прототипа на ранних этапах разработки;
- Приоритетом тестирования являются критические модули, что приводит к ранней локализации архитектурных ошибок;

Недостатки:

- Большое количество заглушек;
- Тестирование низкоуровневых блоков откладывается на поздние этапы;





Рекомендации к проведению интеграционного тестирования

- Определить стратегию, которая должна соответствовать методологии разработки, а также требованиям заказчика;
- Изучить архитектуру приложения и определить критические модули и приоритет их тестирования;
- Описать тестовые сценарии и данные;
- Изучить интерфейсы взаимодействия;
- Выбрать тестовые данные для тестов;
- Провести тестирование и изучить результаты;

Анализ покрытия кода



%

Coverage

Покрытие кода тестами недостаточное, если при внесении изменений ломается функционал, но не ломаются тесты.

Покрытие чрезмерное, если ломается слишком много тестов.

Мартин Фоулер

Утилиты анализа покрытия кода:

JaCoCo:

- Eclipse Public License;
- Поддерживает Java 7-15;
- Интегрируется с продуктами: SonarQube, Jenkins, TeamCity, Gradle, Maven, IDEs;
- Динамично развивается;

OpenClover:

- Продукт Atlassian;
- Интегрируется с продуктами: Ant, Maven, Gradle, IDEs, Bamboo, Jenkins, SonarQube;
- Динамично развивается;

```
mvn clean clover:setup test clover:aggregate clover:clover
```

Cobertura;

JCov;

Serenity;

Метрики покрытия кода

Типы метрик:

- Покрытие метода;
- Покрытие класса;
- Покрытие строк;
- Проверка ветвлений;



Test-Driven Development

Разработка через тестирование – экстремальная техника разработки программного обеспечения, основанная на коротких циклах разработки по принципу

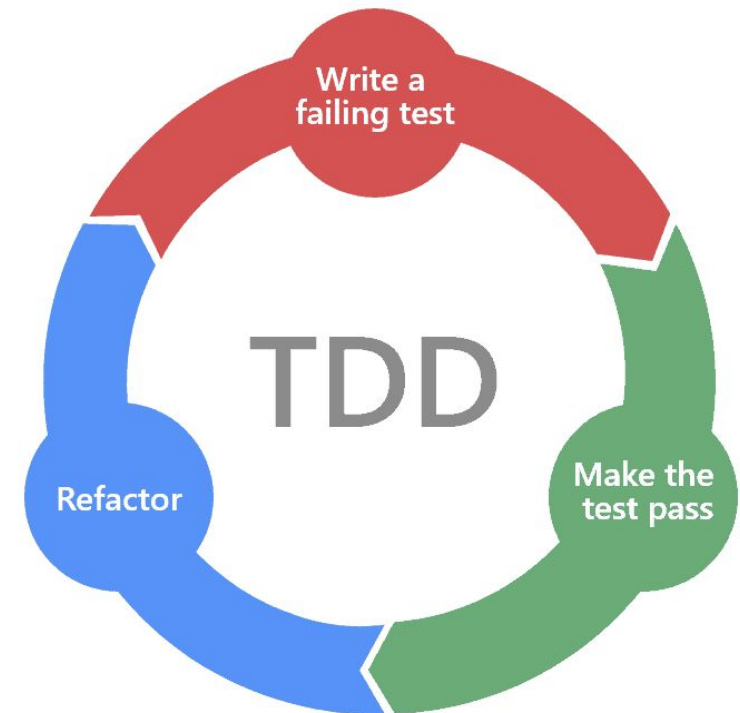
«Тест – Код – Рефакторинг»

Автор – **Кент Бек**.

Основное требование к тестам – их быстрота и автоматизация.

Требование к организации кода – высокая связность и слабая сцепленность.

В TDD наиболее ярко проявляют себя принципы KISS и YAGNI



Принципы в TDD

KISS

YAGNI

Low coupling

High cohesion

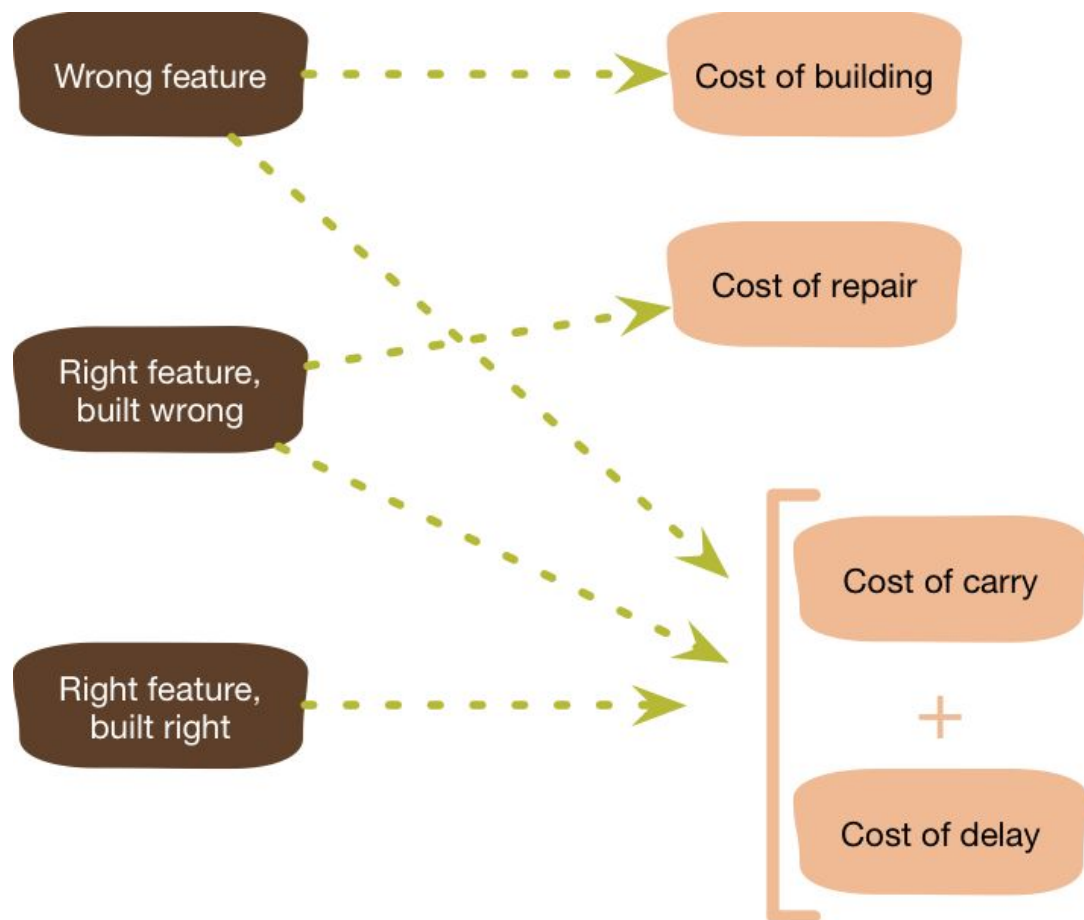
**KEEP
IT
SIMPLE
STUPID**

KISS

Основные правила:

- Разбиение задачи на множество маленьких атомарных подзадач;
- Маленькие методы, обеспечивающие единственную безусловную функциональность;
- Маленькие классы узкой направленности;
- Сначала продуманное решение, затем код;
- Постоянная переработка кода: рефакторинг и обновление состава согласно текущим задачам





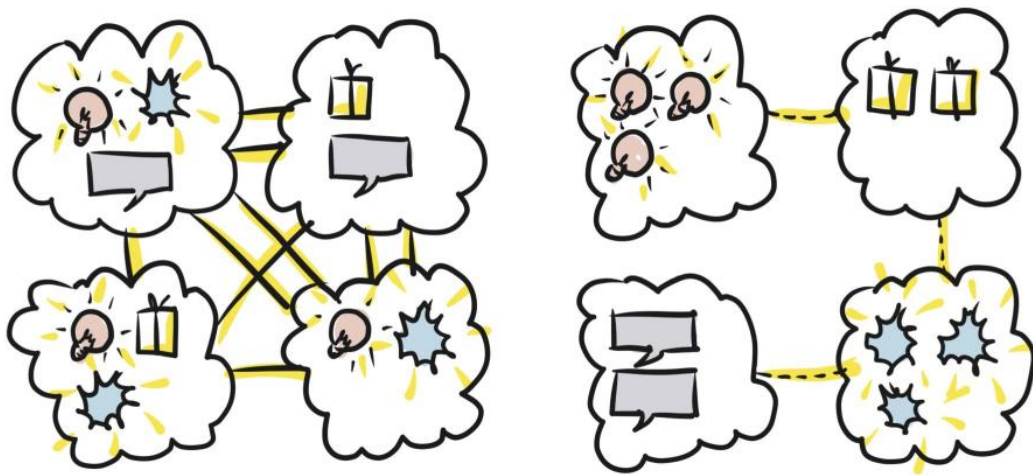
YAGNI

You ain't gonna need it

Основная цель – отказ от избыточной функциональности.

От программиста требуется разработка и совершенствование кода, который реализует поставленную задачу.

Код, который создается для потенциального применения в будущем, может быть опасен для текущего состояния продукта, а также делать продукт сложнее и тяжеловеснее.



Low Coupling (слабое зацепление)

Распределение ответственности и данных, обеспечивающих взаимную независимость классов.

- *слабая зависимость от внешних сущностей;*
- *независимость от изменений внешних сущностей;*
- *простота переиспользования;*

High Cohesion (высокая связность)

Обязанности элемента тесно связаны и сфокусированы.

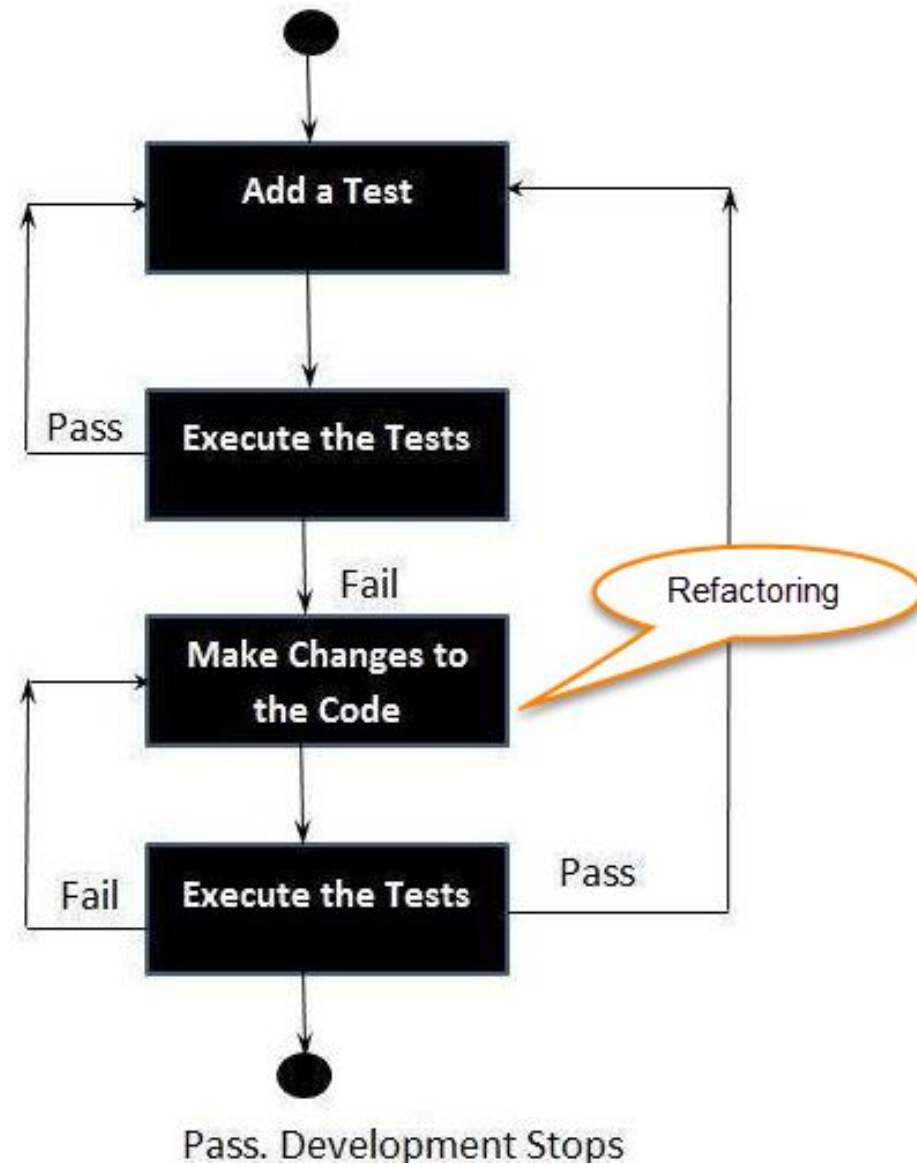
- *повышает восприятие сущности;*
- *упрощение поддержки;*
- *устойчивость к внешним изменениям;*
- *простота переиспользования;*

GRASP

шаблоны распределения
ответственности

- **Low Coupling** (слабое зацепление)
- **High Cohesion** (высокая связность)
- Information Expert (информационный эксперт)
- Creator (создатель)
- Polymorphism (полиморфизм)
- Pure Fabrication (чистая выдумка)
- Indirection (перенаправление)
- Protected Variations (устойчивость к изменениям)

Последовательность TDD



TTD VS Classic approach (TLD)

TLD

Входные параметры
получаются из исследования
внутренней логики

Покрытие кода тестами часто
синтетическое

Большая вероятность
избыточного кода

Порого входа ниже

TDD

Внутренняя логика
формируется из тестовых
параметров

В любой момент времени код
всегда протестирован

Код состоит из минимально
требуемой функциональности

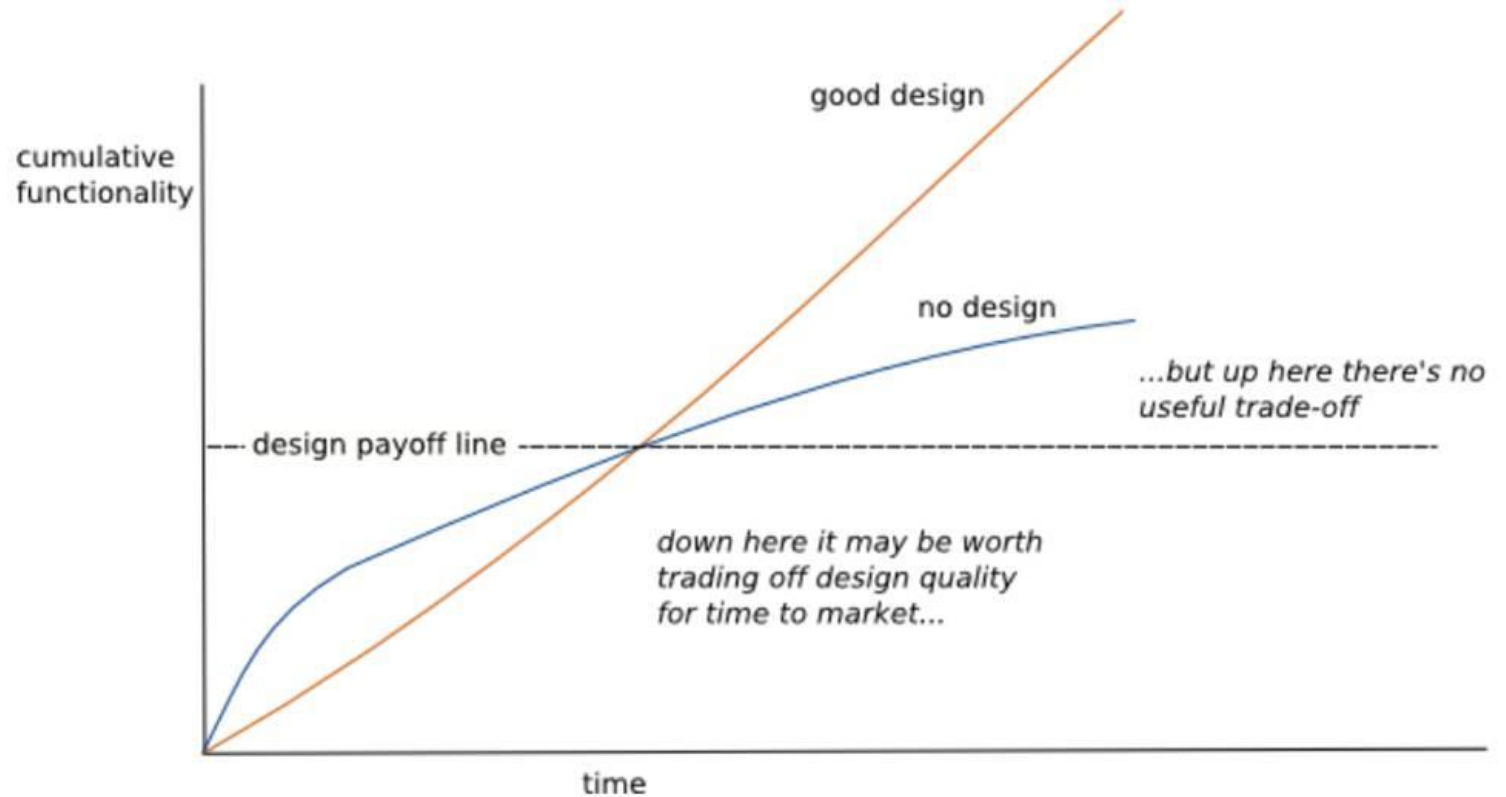
Имплементация принципов
SOLID

Выполняется принцип «Test
with a purpose»

100% покрытия кода тестами

Быстрая адаптация на проекте

Применение TTD



* в идеальной
пропорции



Идеальный TDD?

- Эффективен на ранних этапах разработки;
- Эффективен на малых дизайнах, в больших приложениях может прогрессивно увеличивать собственную сложность;
- Наилучшая область применения – модульное тестирование, практически не применим на этапе интеграционного тестирования;
- На сложных структурах приводит к порождению большого числа заглушек;
- Высокий инженерный порог входа;
- Сложность принятия подхода заказчиком;

