TDD – Test-Driven Development

Разработка через тестирование

Подготовил: Динмухаметов Айнур
11-304

Кент Бек — разработчик программного обеспечения, создатель таких методологий разработки ПО как экстремальное программирование (XP) и разработка через тестирование (TDD)

Разработка через тестирование

Техника разработки ПО, основывающаяся на повторении очень коротких циклов разработки:

1. Написать тест, покрывающий желаемое изменение.
2. Написать код, который позволяет пройти тест.
3. Выполнить рефакторинг.

Тест

Тест – это процедура, которая позволяет либо подтвердить, либо опровергнуть работоспособность кода.

Тесты бывают ручные и автоматические.

Инверсия ответственности

От логики тестов и от их качества зависит, будет ли код соответствовать техническому заданию.

Методика TDD

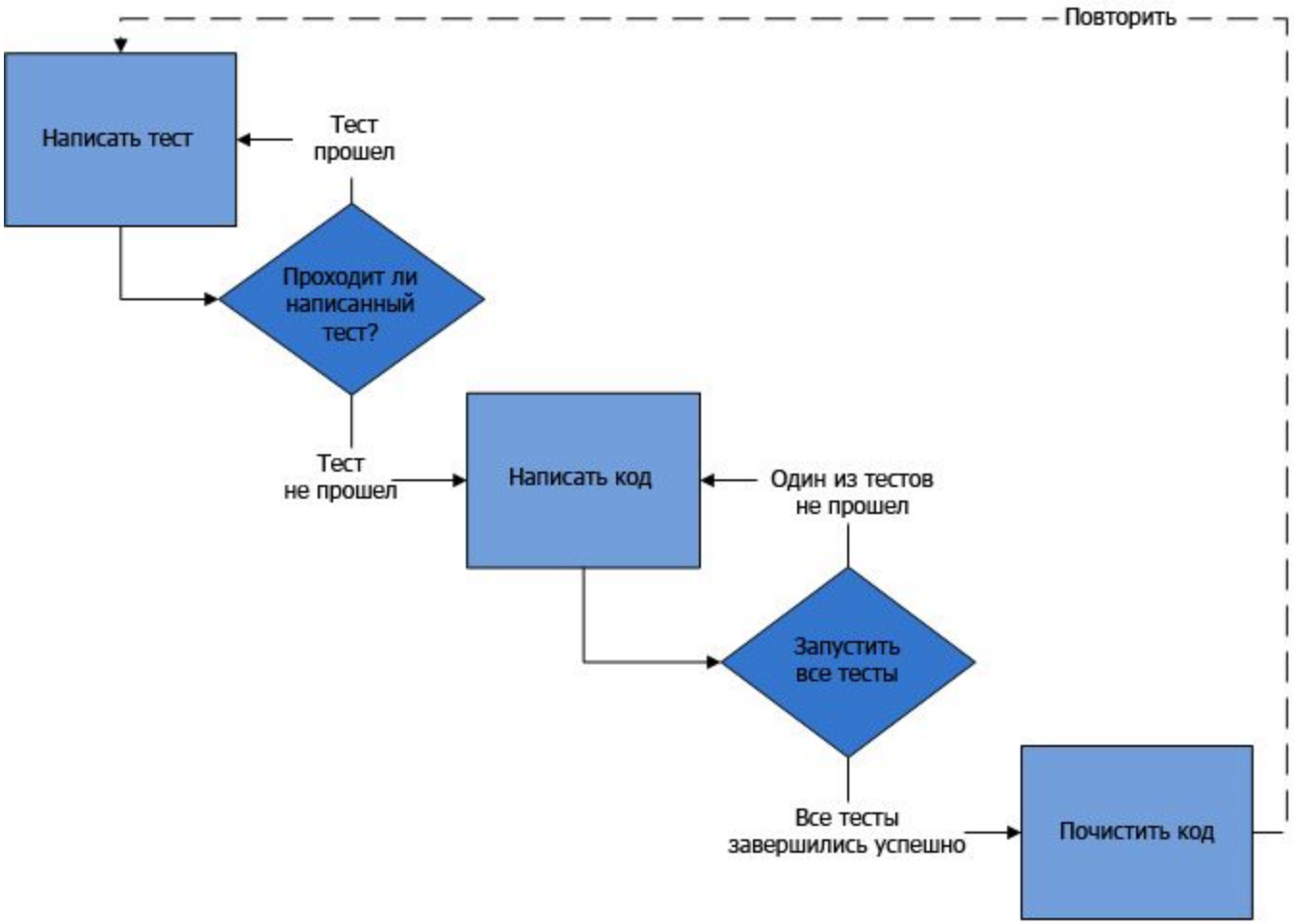
Методика TDD заключается, в основном, в организации автоматических тестов (unit testing) и выработке определенных навыков тестирования.

Одной из особенностей является написание тестов ДО написания кода.

Мантра TDD

- Написать тест;
- Добиться, чтобы тест сработал;
- Устранить дублирование (выполнить *рефакторинг*).

Цикл разработки TDD



Рефакторинг

- Процесс изменения внутренней структуры программы, не затрагивающий её внешнего поведения и имеющий целью облегчить понимание её работы.

Причины применения рефакторинга

Рефакторинг применяется постоянно при разработке кода. Основными стимулами являются:

1. Необходимо добавить новую функцию, которая недостаточно укладывается в принятое архитектурное решение.
2. Необходимо исправить ошибку, причины возникновения которой сразу не ясны. (*Плохой код*).
3. Преодоление трудностей в разработке, которые обусловлены сложной логикой программы. (*Плохой код*).

Пишем тест

```
[Test]
public void MultiplicationTest()
{
    Dollar five = new Dollar(5);
    five.times(2);

    Assert.AreEqual(five.Amount, 10);
}
```

```
[Test]
public void MultiplicationTest()
{
    Dollar five = new Dollar(5);
    five.times(2);

    Assert.AreEqual(five.Amount, 10);
}
```

Не компилируется – 4 ошибки!

1. Нет класса Dollar;
2. Нет конструктора;
3. Нет метода times()
4. Нет переменной класса Amount

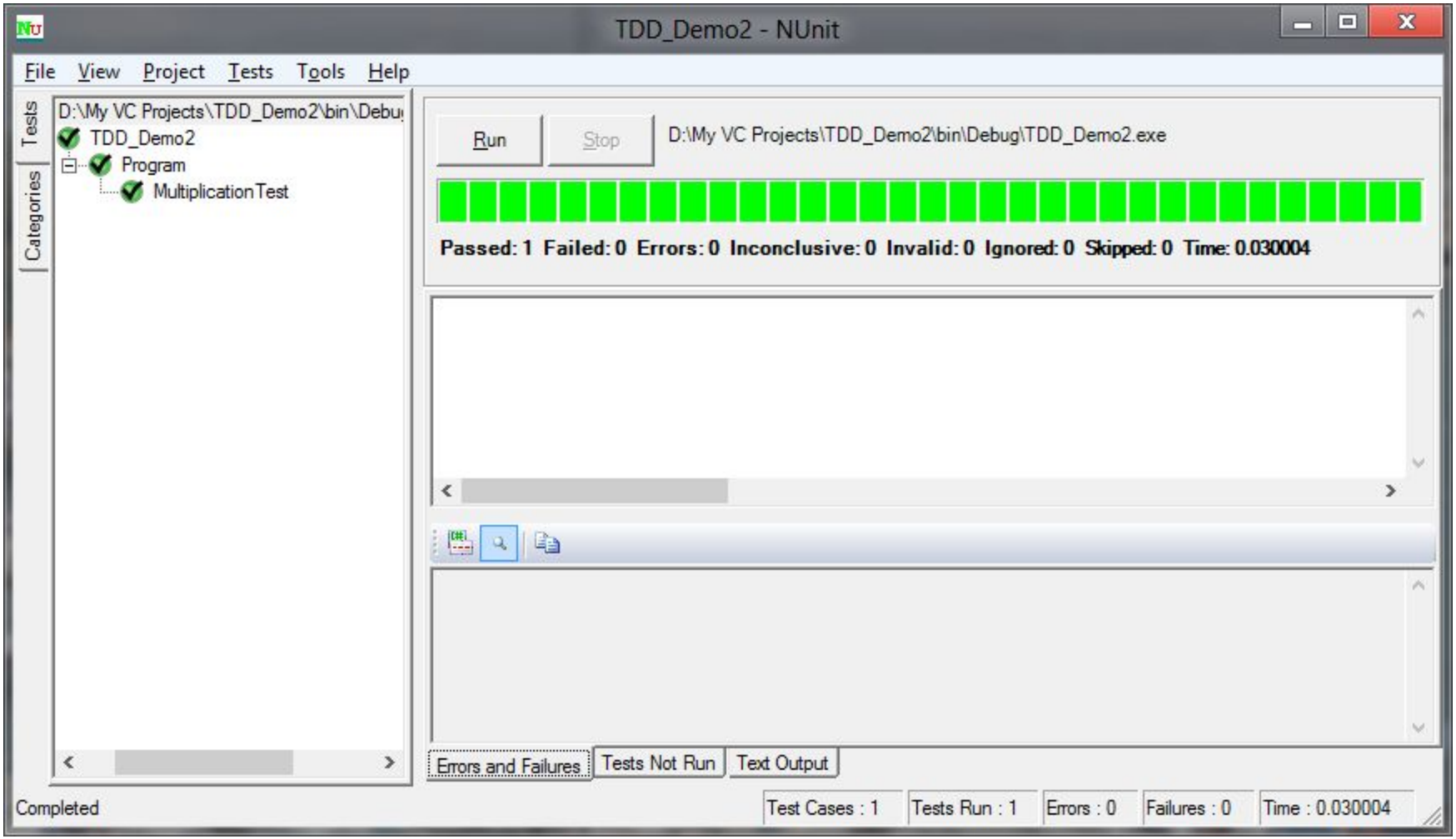
Решение

```
public class Dollar
{
    public Dollar(int amount)
    {
    }

    public int Amount = 10;

    public void times(int n)
    {
    }
}
```

Успешный тест – зеленая полоса!



Полный цикл TDD

1. Добавить небольшой тест.
2. Запустить все тесты, при этом обнаружить, что что-то не срабатывает.
3. Внести небольшое изменение.
4. Снова запустить тесты и убедиться, что все они успешно выполняются.
5. Устранить дублирование с помощью рефакторинга.

После рефакторинга

```
public class Dollar
{
    public Dollar(int amount)
    {
        this.Amount = amount;
    }

    public int Amount = 0;

    public void times(int n)
    {
        Amount *= n;
    }
}
```

Обычный цикл TDD

1. Напишите тест.

- Представьте, как будет реализована в коде воображаемая операция.
- Придумывая её интерфейс, опишите все элементы, которые, как вам кажется, понадобятся.
- Пример: в первом тесте мы добавили класс Dollar, функцию times и переменную-член Amount.

2. Заставьте тест работать

- Первоочередная задача – получить зеленую полосу.
- Если ОЧЕВИДНО простое и элегантное решение – создайте его.
- Если на реализацию такого решения потребуется время – ОТЛОЖИТЕ его. Просто отметьте, что к нему придется вернуться, когда будет решена основная задача – быстро получить зеленый индикатор.
- Противоречит правилам хорошей разработки?

3. Улучшите решение

- После того, как система работает, избавьтесь от прошлых прегрешений и вернитесь к хорошей разработке.
- Удалите дублирование (и другие огрехи) и быстро сделайте так, чтобы полоска снова стала зеленой.

Чистый код, который работает

1. Сначала мы получаем код, который работает.
2. Затем делаем из него чистый код.

Слабые места TDD

- Сложно привыкнуть.
- Сложно применять TDD в ряде случаев, например, при разработке GUI.
- Требуется больше времени на разработку, т.к. необходимо писать тесты.
- Т.к. модульные тесты обычно пишутся теми же, кто написал код, то в случае неверной трактовки требований к приложению, и тест и тестируемый код могут содержать ошибки.
- И др.

СПАСИБО ЗА ВНИМАНИЕ!