Разработка программного обеспечения. От анархии к BDD.

Анархия

Особенности:

- маленькие команды (1-3 человека)
- отсутствие коммуникаций с заказчиком (или PO)
- отсутствие требований
- отсутствие приоритетов задач
- отсутствие как такового тестирования
- быстрые релизы

Анархия

Разработка по требованиям

Особенности:

- средний размер команд (5-8 человек)
- есть кто-то, кто пишет требования (возможно это даже аналитик)
- есть разбивка на задачи по этим требованиям
- есть либо тестировщик, либо отдельная команда тестирования
- понятно куда движется продукт

Разработка по требованиям

Плюсы:

+ постоянные коммуникации с заказчиком (или PO)
+ есть план развития продукта (если это продуктовая разработка)
+ понятны этапы разработки (сроки и состав релизов)
+ понятно поведение системы в текущем релизе
+ в команде есть разделение ответственности (разработка, тестирование, аналитика, менеджмент)

Разработка по требованиям

Минусы:

- финальные требования сложно донести заказчику (PO)
- в основном ручное тестирование
- автотесты если и пишутся, то бессистемно
- после релизного регрессионного тестирования много багов

Test Driven Development (TDD)
“Разработка через тестирование”

Особенности:

- средний или большой размер команд (8 и более человек)
- есть аналитик или команда аналитиков
- есть команда тестирования
- сначала требования, потом тесты и только потом разработка
- подходит не всем

Test Driven Development (TDD)
“Разработка через тестирование”

Плюсы:

+ жёсткое ревью требований на этапе проработки тестов
+ много тестов
+ системный подход к написанию тестов
+ при релизном регрессионном тестировании не так много багов
+ стоимость добавления фичи с ростом проекта увеличивается незначительно или не увеличивается вовсе

Test Driven Development (TDD)
“Разработка через тестирование”

Минусы:

- много тестов
- нужно уметь правильно выбирать “чёрный ящик” или “модуль”, который будет тестироваться
- вечное выбивание времени на разработку у заказчика

Примеры тестов

Например, нужно протестировать корректность имени пользователя при регистрации.

Примеры тестов

Первая итерация теста:

```
it('test user name', () => {
    expect(UsernameValidator.validate('12345678')).toBeTrue();
    expect(UsernameValidator.validate('1234567891011')).toBeFalse();
});
```

Примеры тестов

Дальше:

```
it('test user name too long', () => {
  expect(UsernameValidator.validate('1234567891011')).toBeFalse();
});

it('test user name too short', () => {
  expect(UsernameValidator.validate('123456')).toBeTrue();
});
```

Примеры тестов

Дальше:

```
it('test user name correct if more than 6 and less than 13', () => {
  expect(UsernameValidator.validate('12345678')).toBeTrue();
});

it('test user name incorrect if less than 7', () => {
  expect(UsernameValidator.validate('123456')).toBeFalse();
  expect(UsernameValidator.validate('1234')).toBeFalse();
});

it('test user name incorrect if more than 13', () => {
  expect(UsernameValidator.validate('12345678910234')).toBeFalse();
});
```

Примеры тестов

Можем разбивать дальше, и в один момент придём к следующему результату:

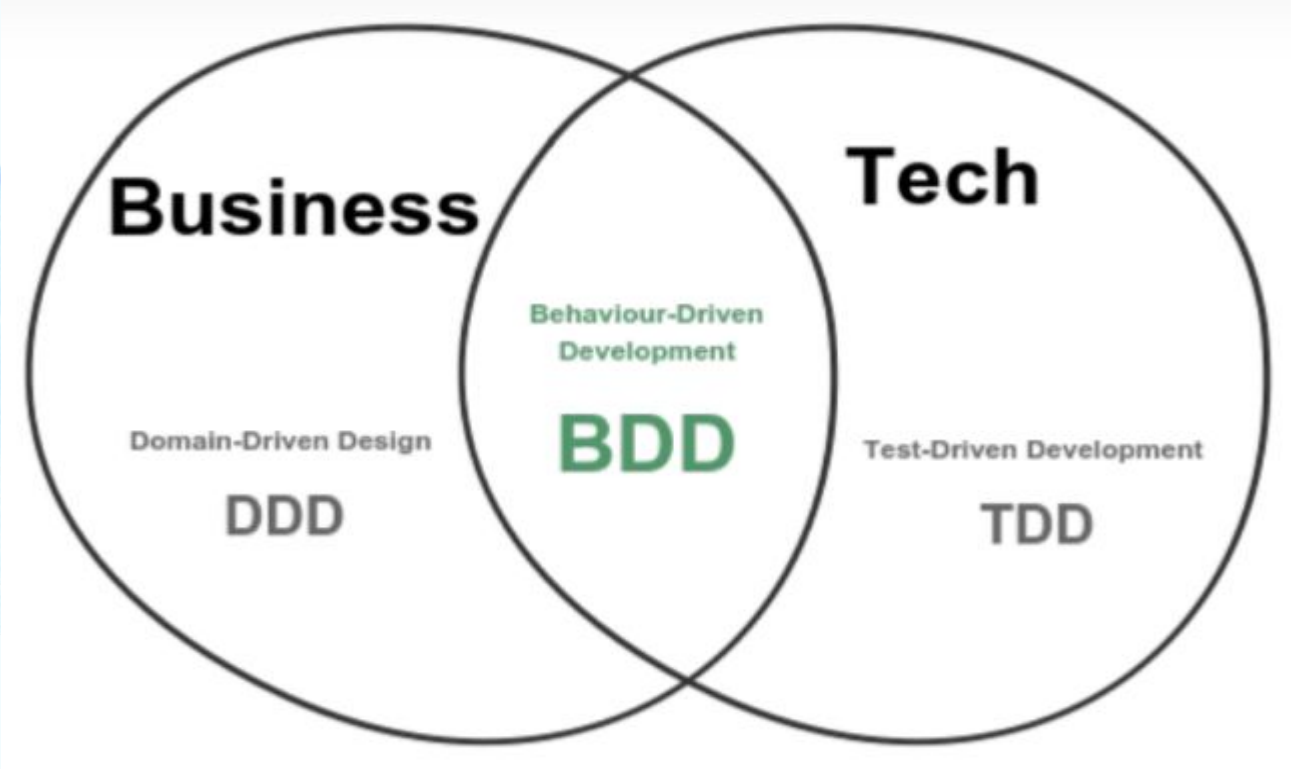
```
it('test user name should be shorter than 13 - UserNameTooLong - Error ', () => {
  //...
});
```

Behaviour Driven Development (BDD)
“Разработка через поведение”

Особенности:

- средний или большой размер команд (8 и более человек)
- есть аналитик или команда аналитиков
- есть команда тестирования
- сначала требования, потом тесты и только потом разработка
- подходит не всем

Behaviour Driven Development (BDD)
“Разработка через поведение”



Behaviour Driven Development (BDD)
“Разработка через поведение”

Плюсы:

+ жёсткое ревью требований на этапе проработки тестов
+ много тестов
+ системный подход к написанию тестов
+ при релизном регрессионном тестировании не так много багов
+ стоимость добавления фичи с ростом проекта увеличивается незначительно или не увеличивается вовсе
+ тесты могут писать не только разработчики, но и команда тестирования

Behaviour Driven Development (BDD)
“Разработка через поведение”

Минусы:

- довольно высокий порог вхождения в команду
- нужно полное понимание требований по фиче
- нужно разработать строгие правила именования тестов

Behaviour Driven Development (BDD)
“Разработка через поведение”

На что не стоит писать тесты:

- Связующий код (например, проверять, что на вызов конкретной команды отрабатывает конкретный обработчик и т.д.)
- Маппинги

Behaviour Driven Development (BDD)
“Разработка через поведение”

На что нужно писать тесты:

- Бизнес правила
- API эндпоинты

Behaviour Driven Development (BDD)
“Разработка через поведение”

Возможно, стоит написать:

- Архитектурные тесты на зависимости в проекте и используемые библиотеки
- Сложные алгоритмы
- Нагрузочные тесты

Behaviour Driven Development (BDD)
“Разработка через поведение”

Для аналитика - Specification By Example, язык Gherkin.

Behaviour Driven Development (BDD)
“Разработка через поведение”

Для разработчиков - SpecFlow (исполняемая спецификация), язык Gherkin.

The End

Репозиторий на GitHub:

https://github.com/feeshka/education/tree/presentation/spec-flow

(ветка presentation/spec-flow)

Полезные статьи:

https://habr.com/ru/articles/107262/

https://habr.com/ru/articles/166747/

https://habr.com/ru/articles/268561/

https://www.uml2.ru/partners/5004/