Тестирование

Это процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы

«*…Любой долгосрочный проект без надлежащего покрытия тестами обречен рано или поздно быть переписанным с нуля…*»

Нужно ли тестирование? Нет?

- рекламные одностраничники, продающие страницы, баннеры или проекты для выставки
- статический сайт-визитка т.е. 1-4 html-страницы с одной или несколькими формами для отправки данных
- небольшое количество кода с отсутствием сложной логики
- срок работы на проектом – от нескольких недель до месяца

Подходы

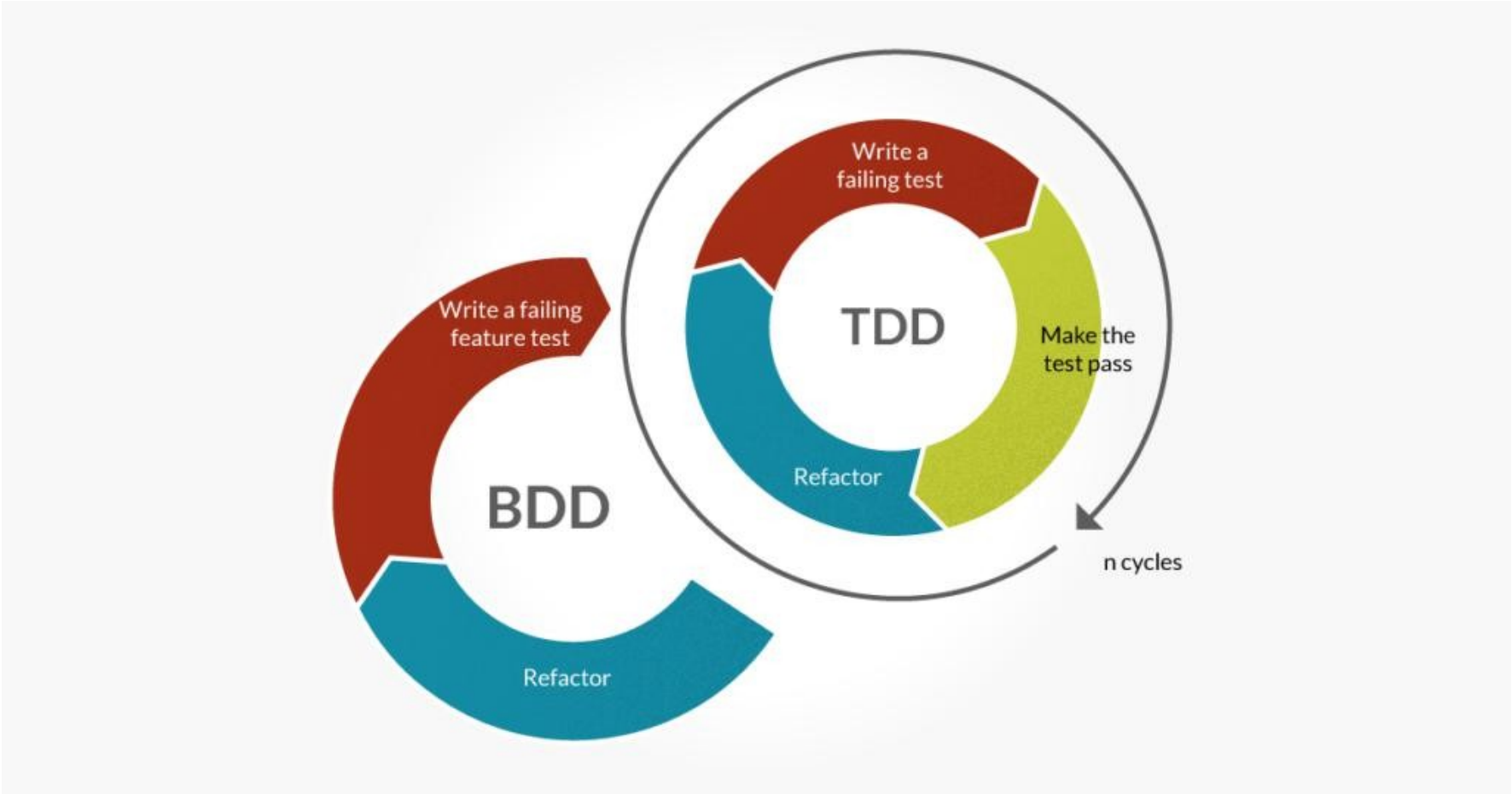
Вначале код (code first)

Сначала происходит написание кода, затем мы тестируем продукт и отправляем его или на доработку, или переходим к следующей стадии разработки

Вначале тесты (test first)

Означает, что мы можем начать тестирование еще до написания самой функции - например, мы можем создать единичный тест или автоматически выполняемый набор тестов до того, как функция или какой-то кусок кода будет разработан и внедрен в приложение

TDD vs BDD


Write a failing test
TDD
Make the test pass
Refactor
n cycles
Write a failing feature test
BDD
Refactor

Test-driven development (TDD)

- Пишем несколько тестов
- Запускаем эти тесты и (очевидно) они терпят неудачу, потому что ни одна из этих функций еще не реализована
- Реализуем эти тесты в коде
- Если все хорошо, то тесты проходят
- Затем следующая итерация

Behavior-driven development (BDD)

- Описываем поведение и спецификации (описываем что должно происходить, каково поведение функции. Пишем, не что мы проверяем, а то, что мы ожидаем от работы еще не реализованной функциональности
- Пишем несколько тестов
- Запускаем эти тесты
- Реализуем эти тесты в коде
- Если все хорошо, то тесты проходят
- Затем следующая итерация

Различия

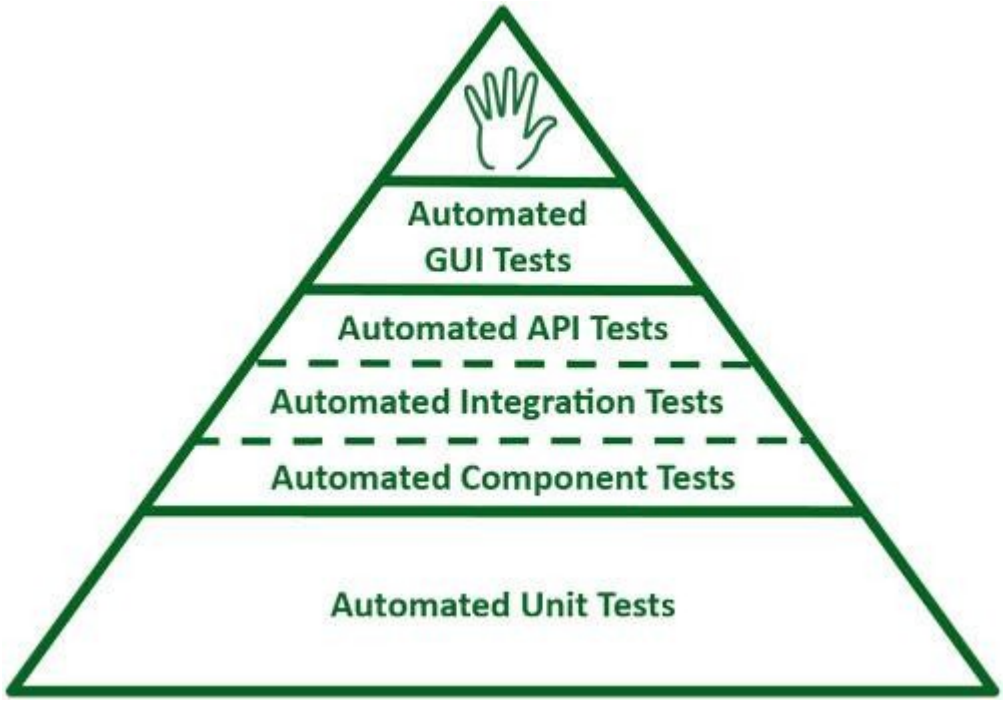
TDD

```
test('равно 2 для 1 + 1' , function (){
      assert.equal(2, 1 + 1);
});
```

BDD

```
it('должно вернуть 2 при сумме
двух элементов 1 и 1', function (){
      (1 + 1).should.equal(2);
});
```

Пирамида тестирования



Mock, stub, spy

Внешняя зависимость — это объект, с которым взаимодействует код и над которым нет прямого контроля. Для ликвидации внешних зависимостей в модульных тестах используются тестовые объекты, например такие как stubs (заглушки)

SPY

Используется для интеграционных тестов, основной функцией является запись данных и вызовов, поступающих из тестируемого объекта для последующей проверки корректности вызова зависимого объекта. Позволяет проверить логику именно нашего тестируемого объекта, без проверок зависимых объектов.

```
it('should call method once with each argument',
function () {
        var object = { method: function () {} };
        var spy = sinon.spy(object, "method");
        object.method(42);
        object.method(1);
        assert(spy.withArgs(42).calledOnce);
        assert(spy.withArgs(1).calledOnce);
})
```

Stub

Заглушка, используется для получения данных из внешней зависимости, подменяя её. При этом игнорирует все данные, кторые могут поступать из тестируемого объекта в stub. Один из самых популярных видов тестовых объектов

```
it('test should stub method differently based
on arguments', function () {
    var callback = sinon.stub();
    callback.withArgs(42).returns(1);
    callback.withArgs(1).throws("TypeError");
    callback(); // No return value, no
    exception
    callback(42); // Returns 1
    callback(1); // Throws TypeError
})
```

Mock

Очень похож на spy, однако не записывает последовательность вызовов с переданными параметрами для последующей проверки, а может сам выкидывать исключения при некорректно переданных данных. Т.е. именно мок-объект проверяет корректность поведения тестируемого объекта.

```
it('test should call all subscribers when exceptions', function () {
    var myAPI = { method: function () {} };
    var spy = sinon.spy();
    var mock = sinon.mock(myAPI);
    mock.expects("method").once().throws();
    PubSub.subscribe("message", myAPI.method);
    PubSub.subscribe("message", spy);
    PubSub.publish("message", "hello !");
    mock.verify();
    assert(spy.calledOnce);
})
```

Пишем тесты правильно

- Тесты в пределах проекта должны быть расположены в соответствии с общей логикой
- Особое внимание уделите именованию тестов, тестовых классов или методов
- Каждый тестирующий класс или метод должен тестировать только одну сущность
- Тестируем только то что нужно. Если это не основное поведение, то оно и не нуждается в тестировании
- Вы определенно ошиблись, если вам нужно запускать тесты в определенном порядке, или если они работают только при активной базе данных или сетевом соединении
- Не относитесь к своим тестам как к второсортному коду
- Тест должен легко поддерживаться, запускаться регулярно и в автоматическом режиме.

Успешность вашей системы тестирования

- Количество багов в новых релизах

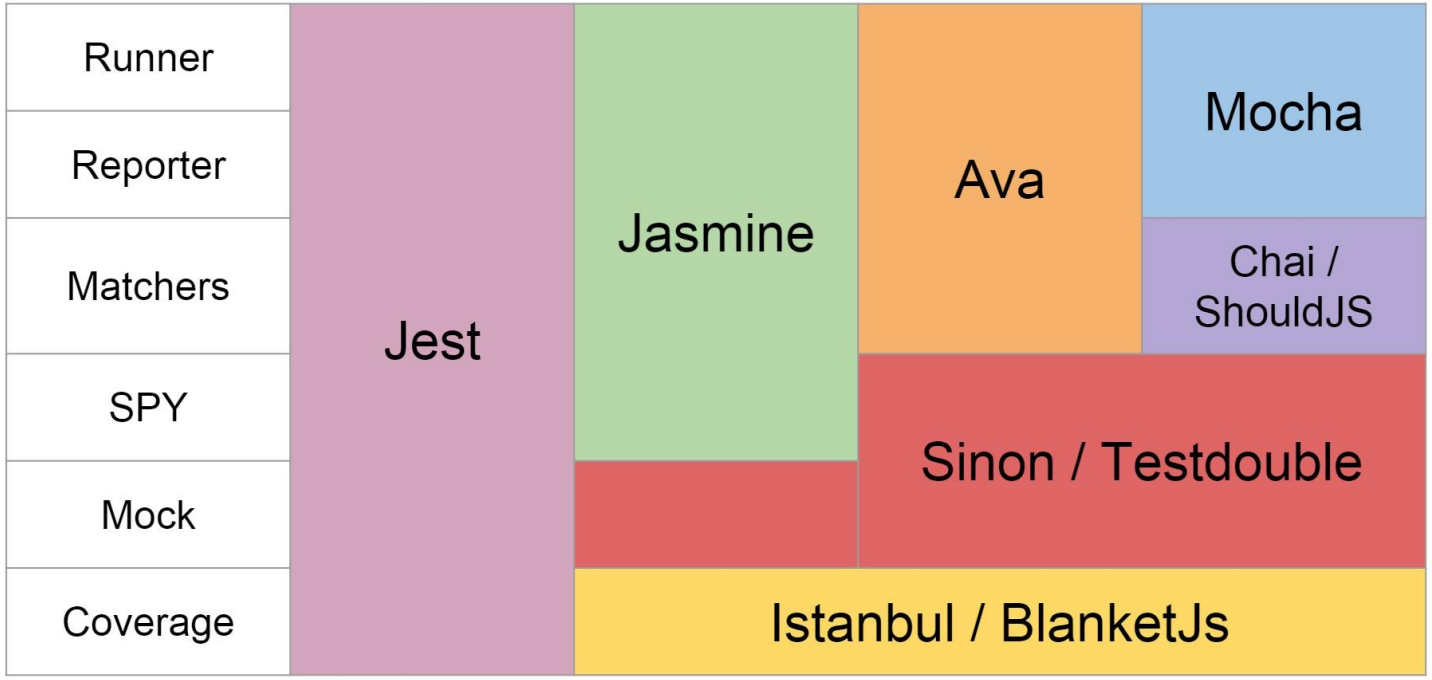
  “показывает, есть ли у наших действий результат, или мы впустую расходуем время”

- Покрытие кода

  “как много нам еще предстоит сделать, но тестовый охват мало полезен в качестве числового заявления о том, насколько хороши ваши тесты”

Успешность вашей системы тестирования

Технологический стек Unit тестов




ПРАКТИКА