

Разработка через тестирование на примере небольших приложений

Подготовил:

Студент гр. 550504 Стрельцов Г.Ю.

Руководитель:

Старший преподаватель Искра Н.А.

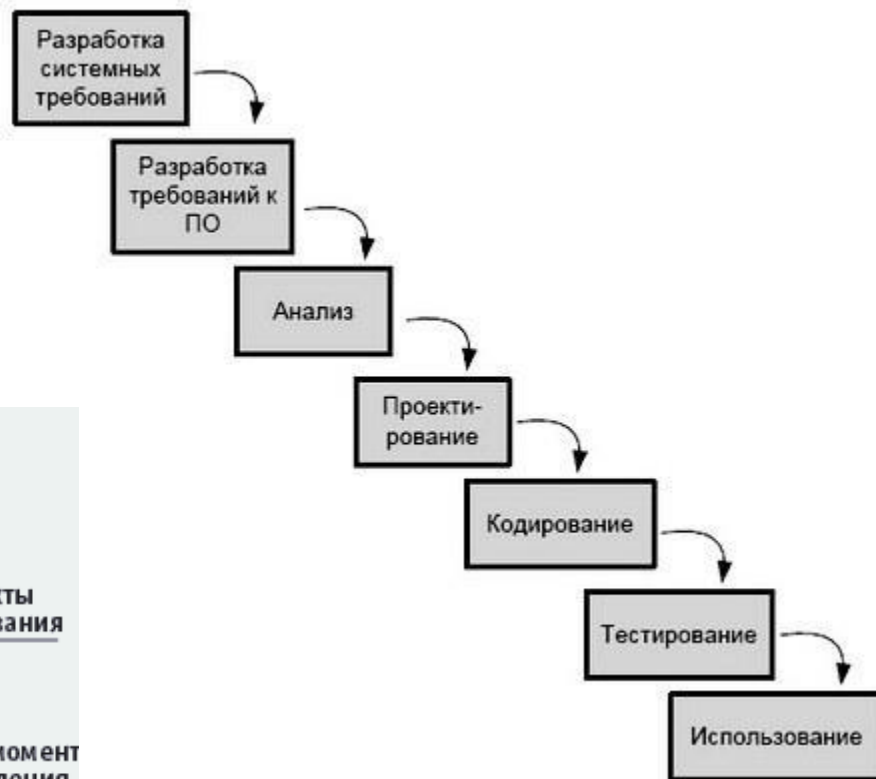
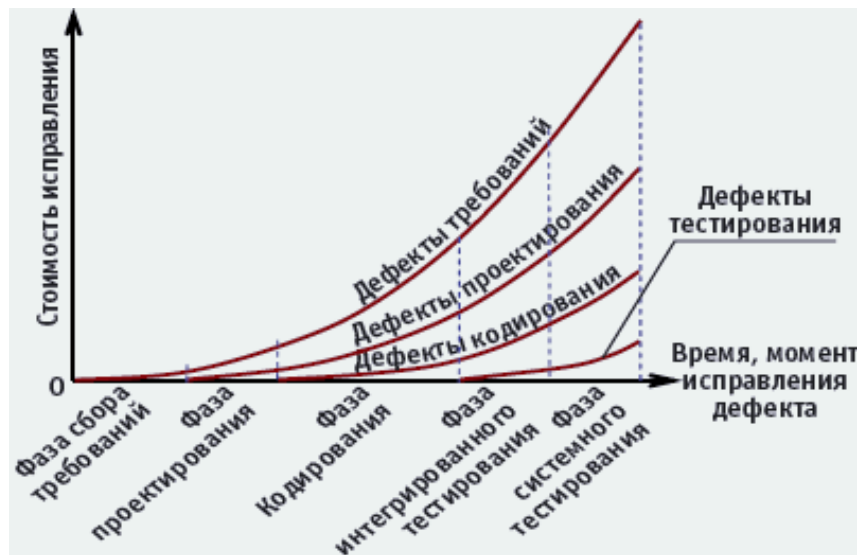
План:



1. Введение
2. Типы тестов для разработчиков
3. Основные стратегии
4. Базовый арсенал
5. JUnit
6. Mockito
7. Примеры
8. Преимущества подхода TDD
9. Недостатки подхода TDD

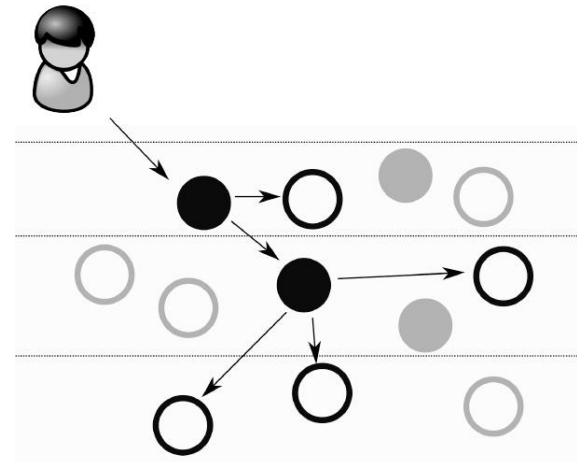
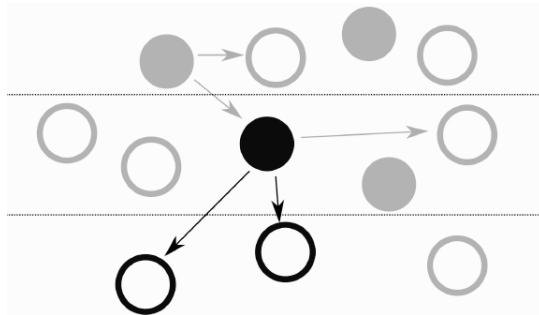
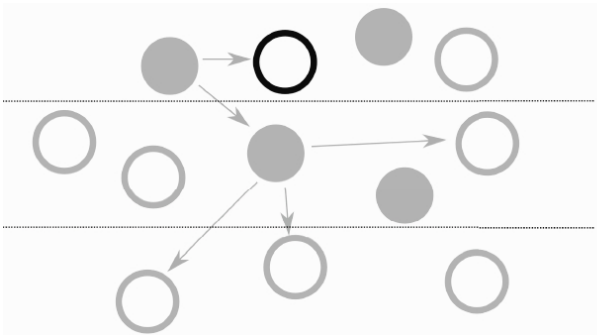
Зачем?

- Быстрее
- Дешевле
- Качественнее



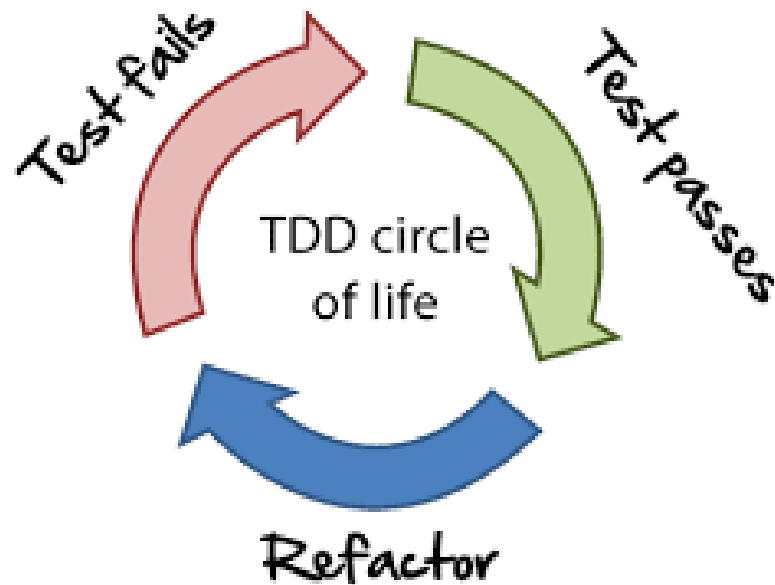
Типы тестов для разработчиков

1. Модульные тесты (Unit tests)
2. Интеграционные тесты (Integration tests)
3. Сквозные тесты (End-to-end tests)



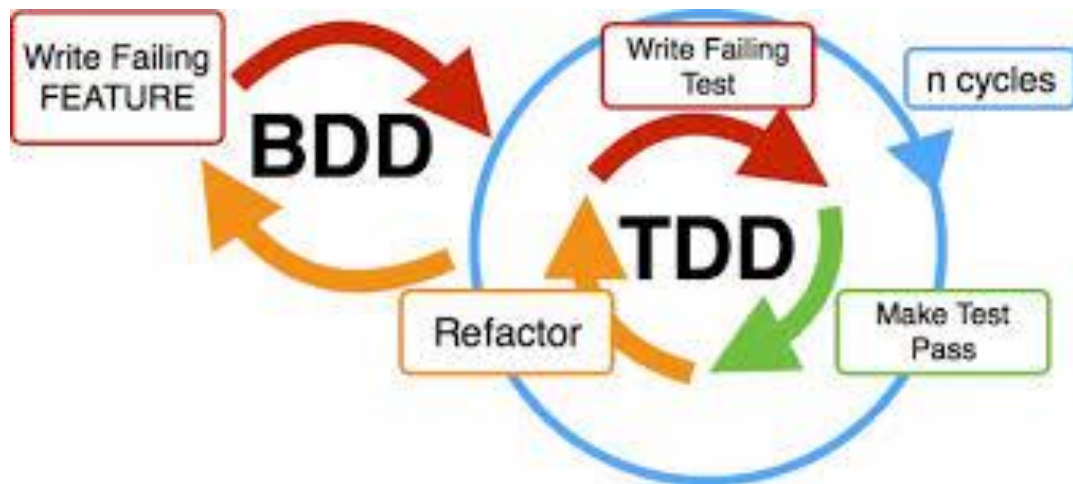
Основные стратегии

TDD (Test-driven development)



Основные стратегии

BDD (Behavior-driven development)





Базовый арсенал

- JAVA
- JUNIT
- MOCKITO
- Build tools (Maven, Gradle...)

JUnit



Другие инструменты

EasyTest & JUnitParams (улучшения для параметризированных тестов)

Hamcrest (тестирование табличных данных в excel, БД, веб-страницах и т.д.)

FEST (тестирование ГУИ на Swing)

EasyMock & PowerMock (альтернатива Mockito)

и т.д.



JUNIT

<http://junit.org>

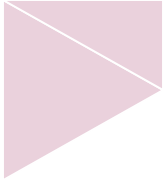
```
import org.junit.Test;
import org.junit.Assert;
public class MathTest {
    @Test
    public void testEquals() {
        Assert.assertEquals(4, 2 + 2);
        Assert.assertTrue(4 == 2 + 2);
    }
    @Test
    public void testNotEquals() {
        Assert.assertFalse(5 == 2 + 2);
    }
}
```

Доступные аннотации JUnit

Аннотация	Описание
@Test public void method()	Аннотация @Test определяет что метод method() является тестовым.
@Before public void method()	Аннотация @Before указывает на то, что метод будет выполняться перед каждым тестируемым методом @Test.
@After public void method()	Аннотация @After указывает на то, что метод будет выполняться после каждого тестируемого метода @Test
@BeforeClass public static void method()	Аннотация @BeforeClass указывает на то, что метод будет выполняться в начале всех тестов, а точнее в момент запуска тестов(перед всеми тестами @Test).
@AfterClass public static void method()	Аннотация @AfterClass указывает на то, что метод будет выполняться после всех тестов.
@Ignore public static void method()	Аннотация @Ignore говорит, что метод будет проигнорирован в момент проведения тестирования.
@Test (expected = Exception.class) public static void method()	(expected = Exception.class) — указывает на то, что в данном тестовом методе вы преднамеренно ожидается Exception.
@Test (timeout=100) public static void method()	(timeout=100) — указывает, что тестируемый метод не должен занимать больше чем 100 миллисекунд.

Проверяющие методы (Основные)

Метод	Описание
<code>fail(String)</code>	Указывает на то, чтобы тестовый метод завалился при этом выводя текстовое сообщение.
<code>assertTrue([message], boolean condition)</code>	Проверяет, что логическое условие истинно.
<code>assertEquals([String message], expected, actual)</code>	Проверяет, что два значения совпадают. <i>Примечание:</i> для массивов проверяются ссылки, а не содержание массивов.
<code>assertNull([message], object)</code>	Проверяет, что объект является пустым null .
<code>assertNotNull([message], object)</code>	Проверяет, что объект не является пустым null .
<code>assertSame([String], expected, actual)</code>	Проверяет, что обе переменные относятся к одному объекту.
<code>assertNotSame([String], expected, actual)</code>	Проверяет, что обе переменные относятся к разным объектам.



JUNIT

Достоинства:

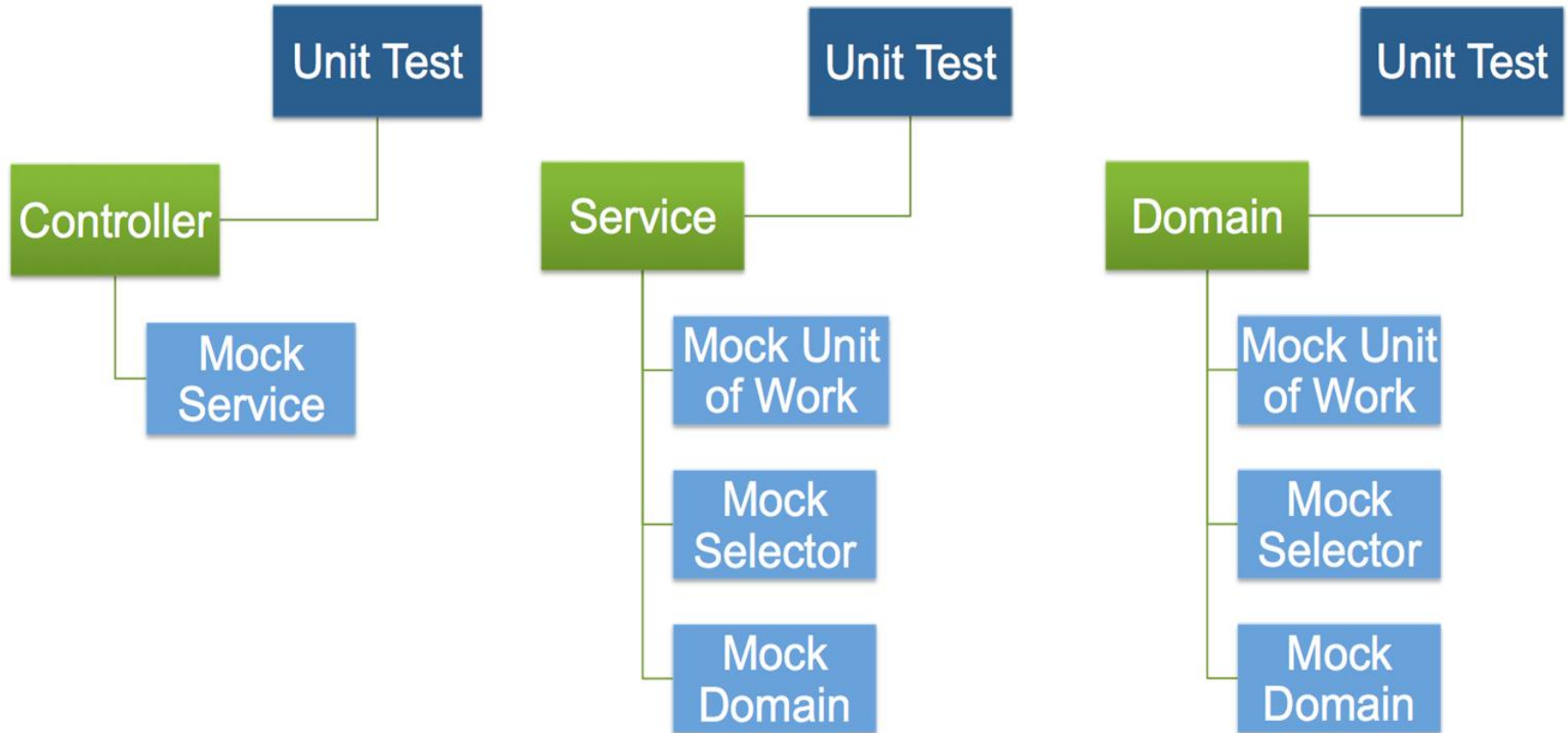
- Простота
- Удобство написания модульных тестов

Недостатки:

- Ограниченная поддержка параметризированных тестов (дополнительное использование JUnitParams)

Mockito

<http://site.mockito.org>





Mockito

Достоинства:

- Простота
- Отличная документация
- Ряд полезных особенностей (имитирует не только конкретные классы, но и интерфейсы)
- Очень легко и удобно использовать в связке с JUnit

Недостатки:

- с точки зрения использования пока не выявлено

Примеры

- 1) Тестирование бизнес-процессов
- 2) Тестирование взаимодействия с хранилищем данных

Тестирование бизнес-процессов

1. Разработка архитектуры
разрабатываемого
модуля с
использованием
интерфейса

```
1      package com.gv.archive.cryptography.interfaces;
2
3      /**
4       * specifies contract for cryptography algorithm
5       */
6      public interface Cryptographer {
7
8          /**
9           * encrypts input string
10          * @param str - input string object
11          * @return encrypt string object
12          */
13          String encrypt(String str);
14
15          /**
16           * decrypts input string
17           * @param str - input string object
18           * @return decrypt string object
19           */
20          String decrypt(String str);
21      }
22
```


Тестирование бизнес-процессов

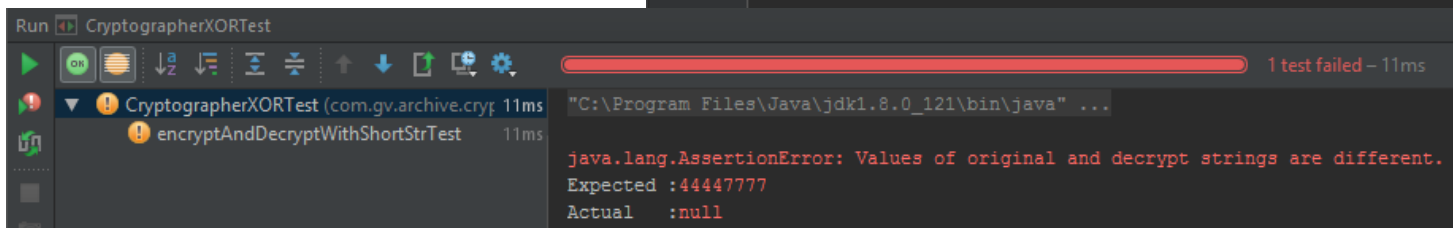
2. Создание сигнатуры конкретной реализации интерфейса

```
1 package com.gv.archive.cryptography.implementations;
2
3 import com.gv.archive.cryptography.interfaces.Cryptographer;
4
5 /**
6  * implements Cryptographer interface according XOR crypt algorithm
7  */
8 public class CryptographerXOR implements Cryptographer {
9
10     @Override
11     public String encrypt(String str) {
12         return null;
13     }
14
15     @Override
16     public String decrypt(String str) {
17         return null;
18     }
19 }
```

Тестирование бизнес-процессов

3. Написание тестов

```
1 package com.gv.archive.cryptography.implementations;
2
3 import com.gv.archive.cryptography.interfaces.Cryptographer;
4 import org.junit.Assert;
5 import org.junit.Test;
6
7 public class CryptographerXORTest {
8
9     private Cryptographer cryptographer = new CryptographerXOR();
10    private String testStr;
11    private String encryptStr;
12
13    @Test
14    public void encryptAndDecryptWithShortStrTest() throws Exception {
15        testStr = "44447777";
16        encryptStr = cryptographer.encrypt(testStr);
17        String errorMessage = "Values of original and decrypt strings are different.";
18        Assert.assertEquals(errorMessage, testStr, cryptographer.decrypt(encryptStr));
19    }
20 }
```



Тестирование бизнес-процессов

4. Доработка методов тестируемого класса для прохождения тестов

```
10 public class CryptographerXOR implements Cryptographer {
11
12     /** object for extracting properties from resource bundle cryptoKey.properties */
13     private static final ResourceBundle RESOURCE_BUNDLE = ResourceBundle.getBundle("cryptoKey");
14
15     /** name of property in resource bundle that correspond to crypto key */
16     private static final String KEY_PROPERTY_NAME = "key";
17
18     @Override
19     public String encrypt(String str) {
20         byte[] key = RESOURCE_BUNDLE.getString(KEY_PROPERTY_NAME).getBytes();
21         byte[] text = str.getBytes();
22         byte[] result = new byte[str.length()];
23         for(int i = 0; i < text.length; i++){
24             result[i] = (byte)(text[i] ^ key[i % key.length]);
25         }
26         return new String(result);
27     }
28
29     @Override
30     public String decrypt(String str) {
31         return encrypt(str);
32     }
33 }
```

```
▼ OK CryptographerXORTest (com.gv.archive.cryp Oms
    OK encryptAndDecryptWithShortStrTest Oms
    "C:\Program Files\Java\jdk1.8.0_121\bin\java" ...
    Process finished with exit code 0
```

Тестирование бизнес-процессов

5. Больше тестов и доработка кода

```
20
21     @Test
22     public void encryptAndDecryptWithEmptyStrTest() throws Exception {
23         testStr = "";
24         encryptStr = cryptographer.encrypt(testStr);
25         String errorMessage = "Result string after decrypting is not empty as original.";
26         Assert.assertEquals(errorMessage, testStr, cryptographer.decrypt(encryptStr));
27     }
28
29     @Test
30     public void encryptAndDecryptWithNullStrTest() throws Exception {
31         testStr = null;
32         encryptStr = cryptographer.encrypt(testStr);
33         String errorMessage = "Result result of encrypting null must be also null.";
34         Assert.assertNull(errorMessage, encryptStr);
35     }
36
37     @Test
38     public void encryptAndDecryptWithComplicatedStrTest() throws Exception {
39         testStr = "hello hello hello hello hello 4444 7777!!! Test Test";
40         encryptStr = cryptographer.encrypt(testStr);
41         String errorMessage = "Values of original and decrypt strings are different.";
42         Assert.assertEquals(errorMessage, testStr, cryptographer.decrypt(encryptStr));
43     }
44 }
```

CryptographerXORTest

4 tests done: 1 failed - 12ms

Test Name	Duration	Status
encryptAndDecryptWithNullStrTest	12ms	Failed
encryptAndDecryptWithEmptyStrTest	1ms	OK
encryptAndDecryptWithShortStrTest	0ms	OK
encryptAndDecryptWithComplicatedStrTest	0ms	OK

Stack Trace:

```
"C:\Program Files\Java\jdk1.8.0_121\bin\java" ...
java.lang.NullPointerException
    at com.gv.archive.cryptography.implementations.CryptographerXOR.encrypt(CryptographerXOR.java:22)
    at com.gv.archive.cryptography.implementations.CryptographerXORTest.encryptAndDecryptWithNullStrTest
```

Тестирование взаимодействия с хранилищем данных

- в эпицентре системы

```
public class LoginRequestStrategyTest extends Mockito{

    private RequestStrategy strategy = new LoginRequestStrategy(new DomXMLUserParser());

    private User existedUser = new User( login: "vi4477", name: "Veronika Sanko", Role.GUEST, password: "5588");

    XMLUserConverter converter = new XStreamXMLUserConverter();

    @Test
    public void executeLoginRequestWithExistedUserTest() throws Exception {
        Request request = mock(Request.class);

        String requestBodyStr = "vi4477 5588";
        when(request.getRequestBody()).thenReturn(requestBodyStr);

        Response response = strategy.executeRequest(request);
        verify(request).getRequestBody();

        User result = converter.convertXMLStringToUserObject(response.getResponseBody());
        String errorMessage = "Result and existed user objects are different";
        Assert.assertEquals(errorMessage, result, existedUser);
    }
}
```

OK LoginRequestStrategyTest (com.gv.archive.communication.strategies.implementations)

OK executeLoginRequestWithExistedUserTest

Преимущества подхода TDD

- ваш код полностью покрыт тестами;
- создавая тесты до написания кода класса, вы заранее задумаетесь об его использовании, что положительно скажется как на качестве внешнего интерфейса класса, так и на архитектуре проекта в целом;
- в общем случае, затрачивается меньше времени на качественную разработку ПО;
- хорошие тесты могут легко заменить документацию, т.к. наглядно демонстрируют использование тестируемого кода (live documentation);

Недостатки подхода TDD

- высокий порог вхождения;
- ошибочный или некорректный тест приводит к написанию такого же ошибочного кода;
- много времени тратится при использовании интеграционных и сквозных тестов;
- необходимо игнорировать слишком простые/сложные ситуации для тестирования;
- существуют ситуации в которых TDD использовать нецелесообразно

Вопросы

Материалы: <https://github.com/Strelts0v/tdd-conference>