

# Tests

**Тестирование - это надежда найти в  
своем коде ошибку**

***Steve McConnell***

Автоматизированное тестирование - это процесс верификации программного обеспечения, при котором основные функции и шаги теста, такие как запуск, инициализация, выполнение, анализ и выдача результата, выполняются автоматически при помощи инструментов для автоматизированного тестирования.

© <http://www.protesting.ru/automation/>



# Зачем нужны тесты

Виды тестов

“Чистые” тесты

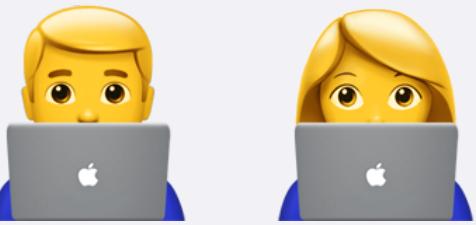
Test doubles

Tips and tricks

# Зачем писать тесты для...

1. Владельца продукта 

2. Тестировщика 

3. Разработчика 

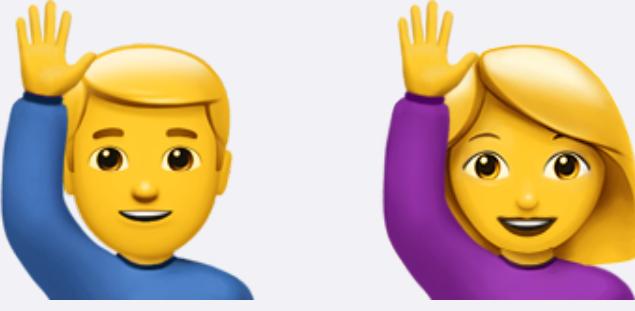
# MAINTAINABILITY

# Владелец продукта



```
1 //  
2 // main.m  
3 // Copyright (c) 2011 iD EAST. All rights reserved.  
4 //  
5  
6 @import SBFoundation;  
7  
8  
9 int main(int argc, char *argv[])  
10 {  
11  
12 #ifndef RELEASE_MODE  
13     sbf_register_dyld_loading();  
14 #endif  
15  
16     @autoreleasepool  
17     {  
18         return UIApplicationMain(argc, argv, @"Application", @"AppDelegate");  
19     }  
20 }
```

# Владелец продукта



На 20,9% дефектов меньше

# Владелец продукта



Стоимость устранения бага, обнаруженного  
после релиза, в **4-5 раз выше**

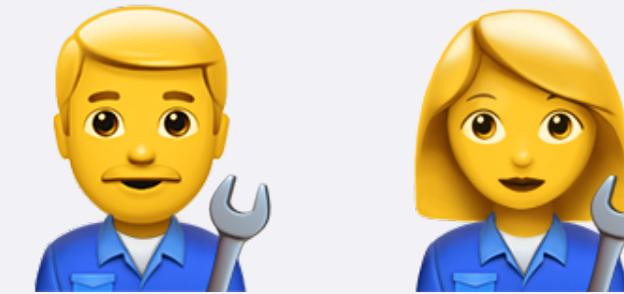
© Systems Sciences Institute at IBM

# Тестировщик



It's harder to find bugs. Now, all the obvious  
bugs are gone.

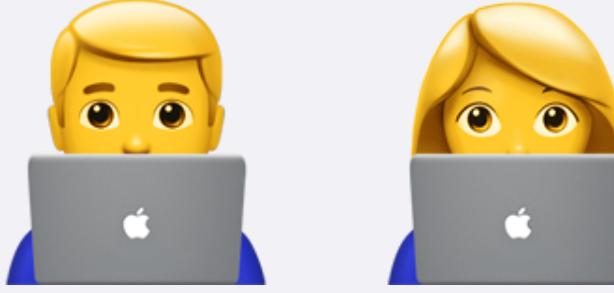
# Тестирующий



**Defect Severity Distribution**

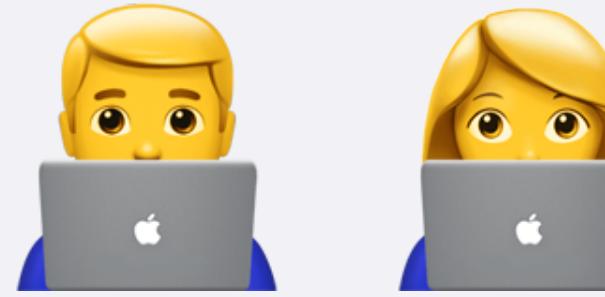
Defect Severity	Version 1 (%)	Version 2 (%)
Severity 1	15.5%	16.8%
Severity 2	49.8%	40.1%
Severity 3	28.7%	18.8%
Severity 4	6.0%	3.4%

# Разработчик



- Понимание требований
- Стабильность при рефакторинге
- Документация
- Хороший API
- Меньше ошибок

# Разработчик



## Developer Survey

When a bug is found in my code, writing a unit test that covers my fix is useful.

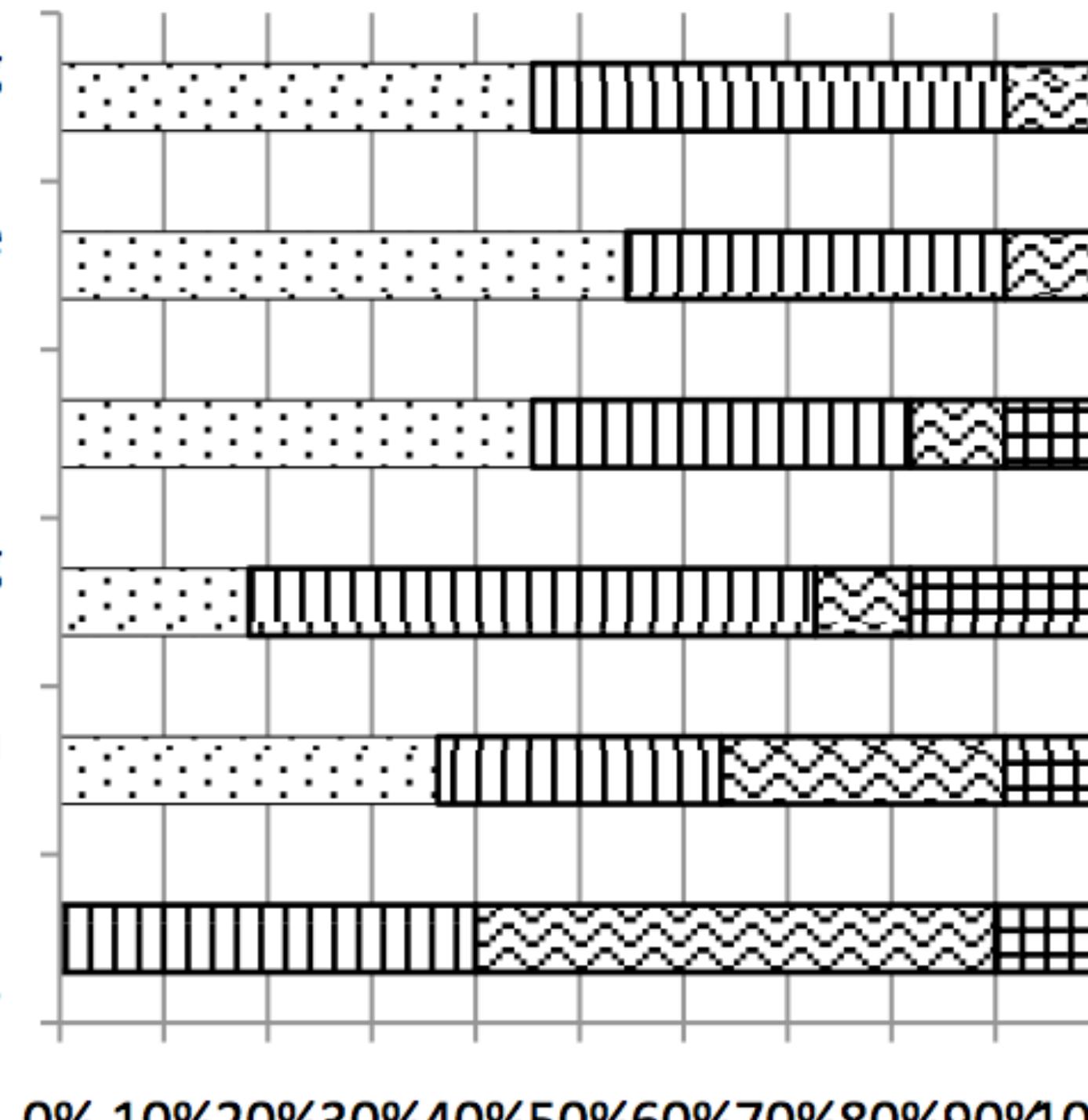
Overall, unit tests help me produce higher quality code.

Unit tests help me write solid code from the start.

When I inherit code in the team, having unit tests helps me to understand the code better.

Unit tests help me debug when I find a problem.

When a bug is found in the code, it is usually in places that have no unit tests.



- Strongly agree
- Agree
- Neutral
- Disagree
- Strongly disagree

Разработка дольше, но и качество выше

# Зачем нужны тесты

## Виды тестов

“Чистые” тесты

Test doubles

Tips and tricks

# Виды тестирования

## Классификации видов и методов тестирования [ [править](#) | [править код](#) ]

Существует несколько признаков, по которым принято производить классификацию видов тестирования. Обычно выделяют следующие:

### По объекту тестирования

- Функциональное тестирование
- Тестирование производительности
  - Нагрузочное тестирование
  - Стресстестирование
  - Тестирование стабильности
- Конфигурационное тестирование
- Юзабилити-тестирование
- Тестирование безопасности
- Тестирование локализации
- Тестирование совместимости

### По знанию внутреннего строения системы

- Тестирование чёрного ящика
- Тестирование белого ящика
- Тестирование серого ящика

### По степени автоматизации

- Ручное тестирование
- Автоматизированное тестирование
- Полуавтоматизированное тестирование

### По степени изолированности<sup>[7][8]</sup>

- Тестирование компонентов
- Интеграционное тестирование
- Системное тестирование

### По времени проведения тестирования<sup>[источник не указан 2376 дней]</sup>

- Альфа-тестирование
  - Дымовое тестирование (англ. *smoke testing*)
  - Тестирование новой функции (*new feature testing*)
  - Подтверждающее тестирование
  - Регрессионное тестирование
  - Приёмочное тестирование
- Бета-тестирование

### По признаку позитивности сценариев

- Позитивное тестирование
- Негативное тестирование

### По степени подготовленности к тестированию

- Тестирование по документации (формальное тестирование)
- Интуитивное тестирование (англ. *ad hoc testing*)

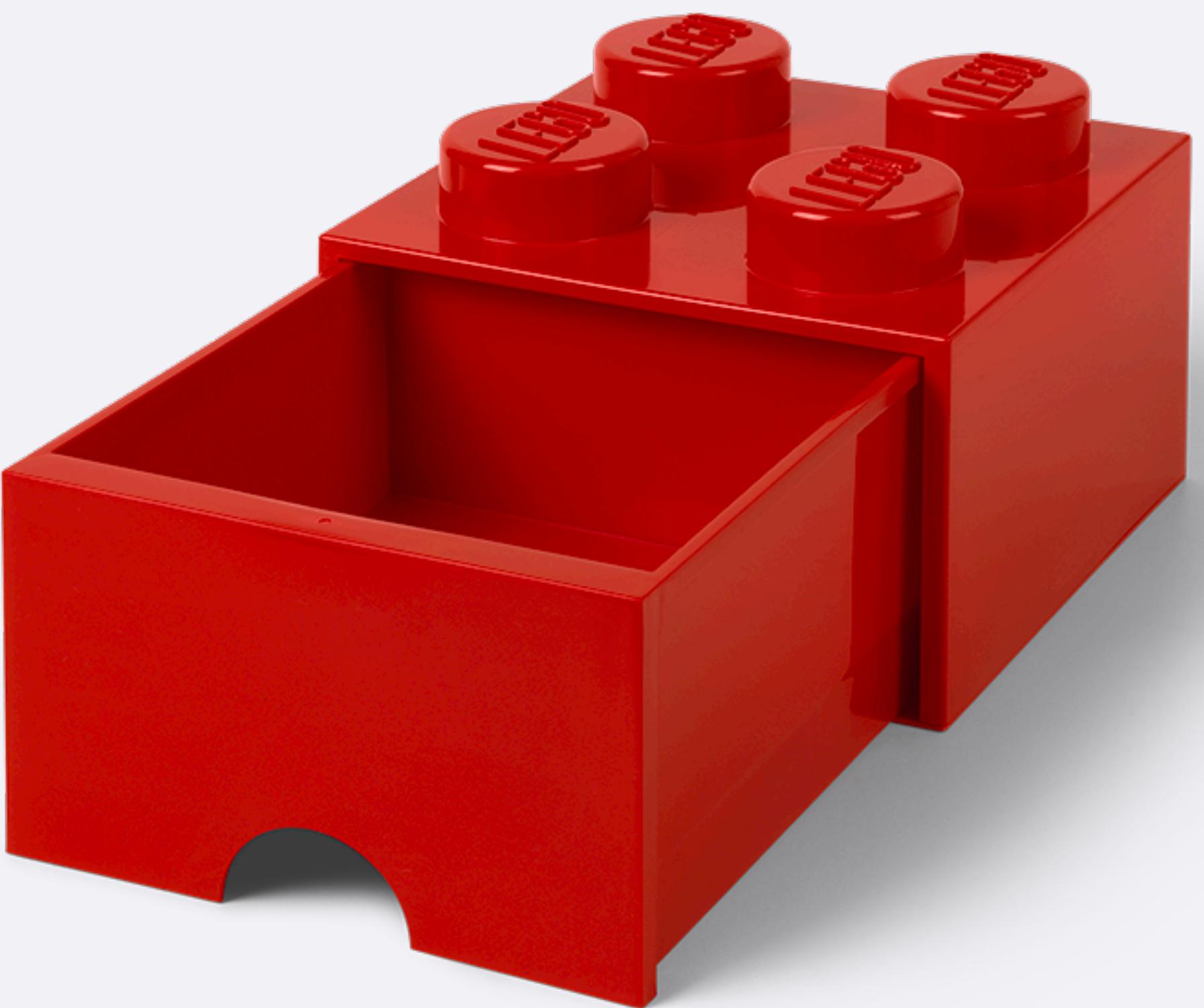
# Виды тестирования

- Модульное (блочное)
  - поведенческое
- Интеграционное
- Приемочное
- UI-тестирование
- Ручное (регрессивное)



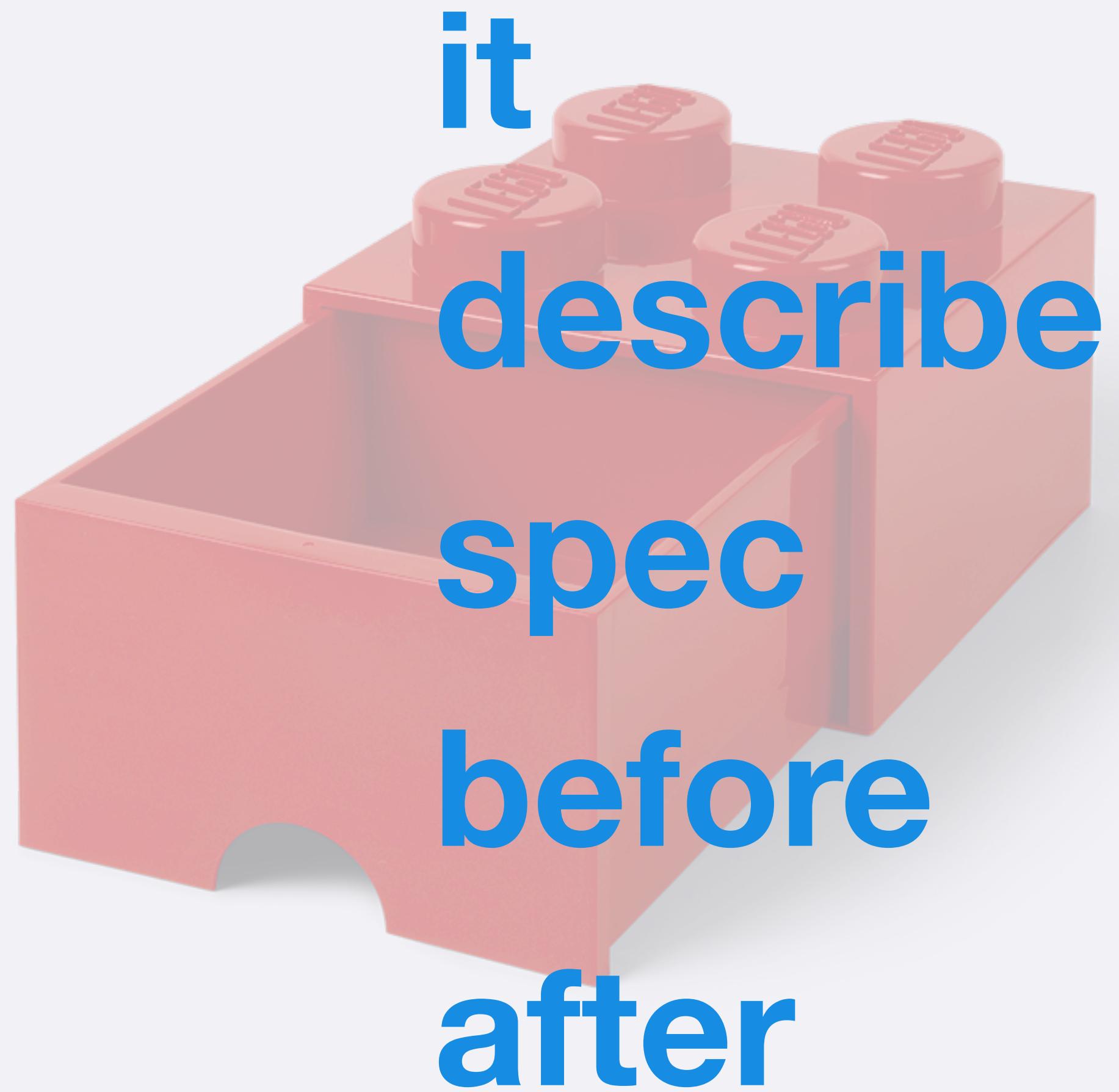
# Виды тестирования

- **Модульное (блочное)**
  - поведенческое
- Интеграционное
- Приемочное
- UI-тестирование
- Ручное (регрессивное)



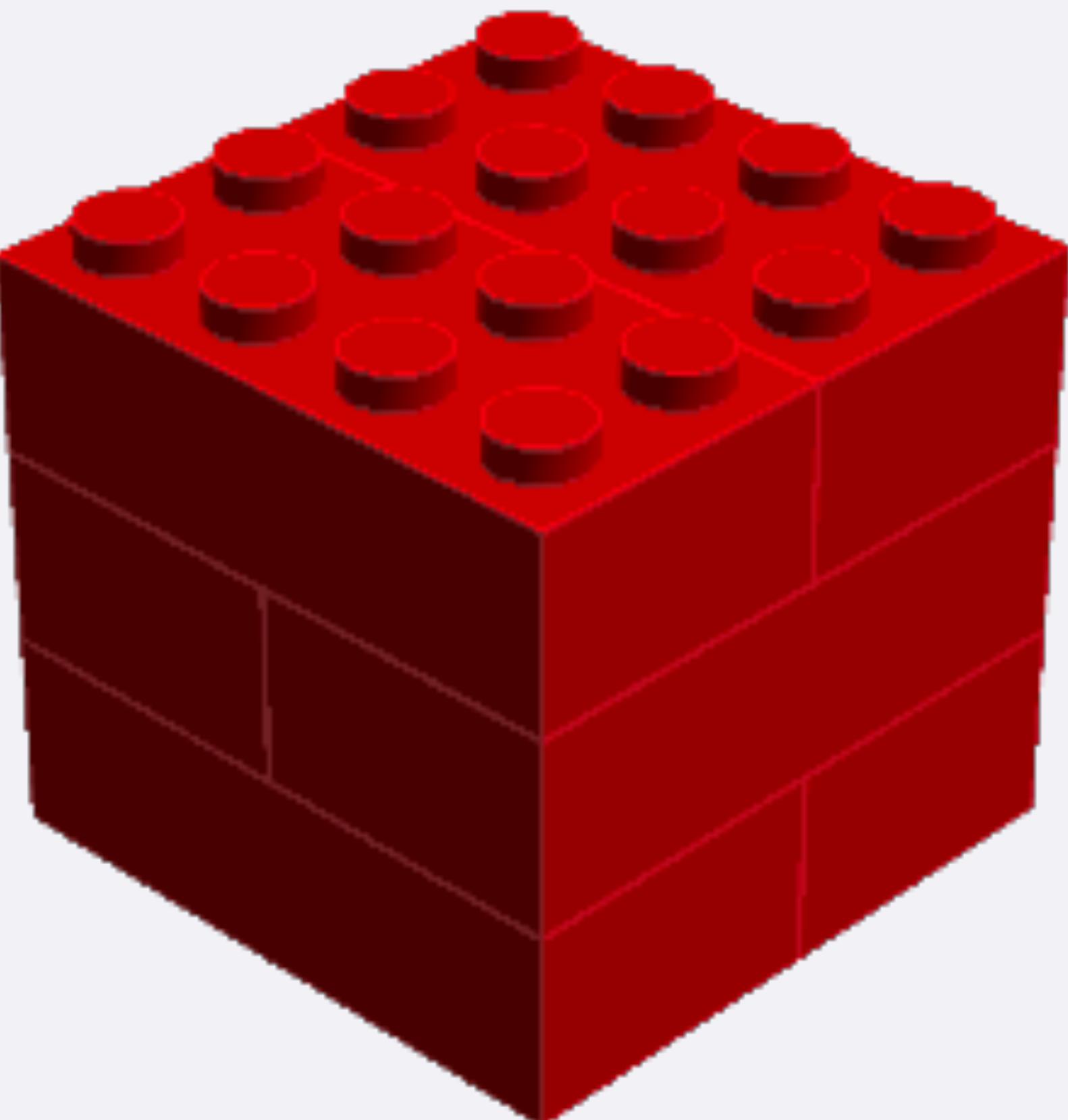
# Виды тестирования

- Модульное (блочное)
  - поведенческое
- Интеграционное
- Приемочное
- UI-тестирование
- Ручное (регрессивное)



# Виды тестирования

- Модульное (блочное)
  - поведенческое
- Интеграционное
- Приемочное
- UI-тестирование
- Ручное (регрессивное)



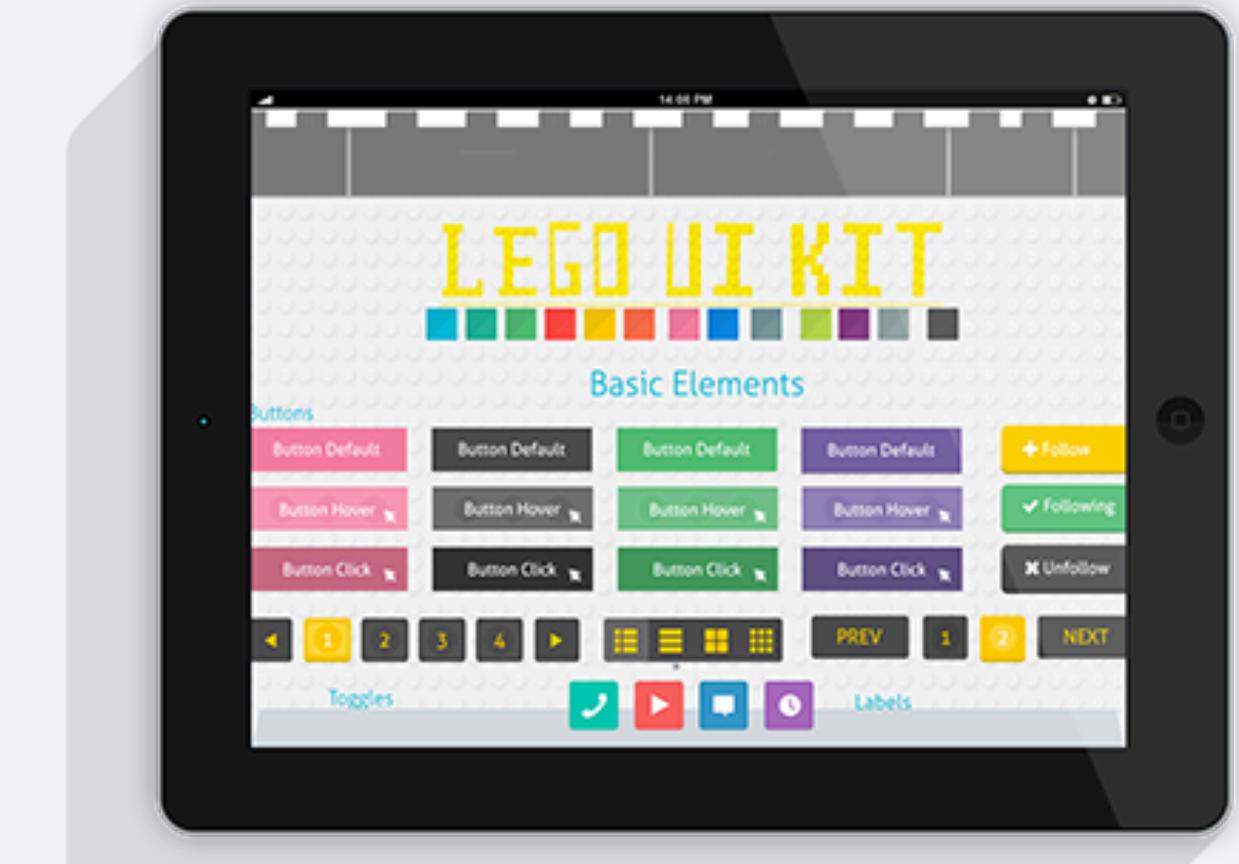
# Виды тестирования

- Модульное (блочное)
  - поведенческое
- Интеграционное
- Приемочное
- UI-тестирование
- Ручное (регрессивное)



# Виды тестирования

- Модульное (блочное)
  - поведенческое
- Интеграционное
- Приемочное
- **UI-тестирование**
- Ручное (регрессивное)



# Виды тестирования

- Модульное (блочное)
  - поведенческое
- Интеграционное
- Приемочное
- UI-тестирование
- Ручное (регрессивное)



The Addison-Wesley Signature Series

# SUCCEEDING WITH AGILE

SOFTWARE DEVELOPMENT  
USING SCRUM

---

MIKE COHN



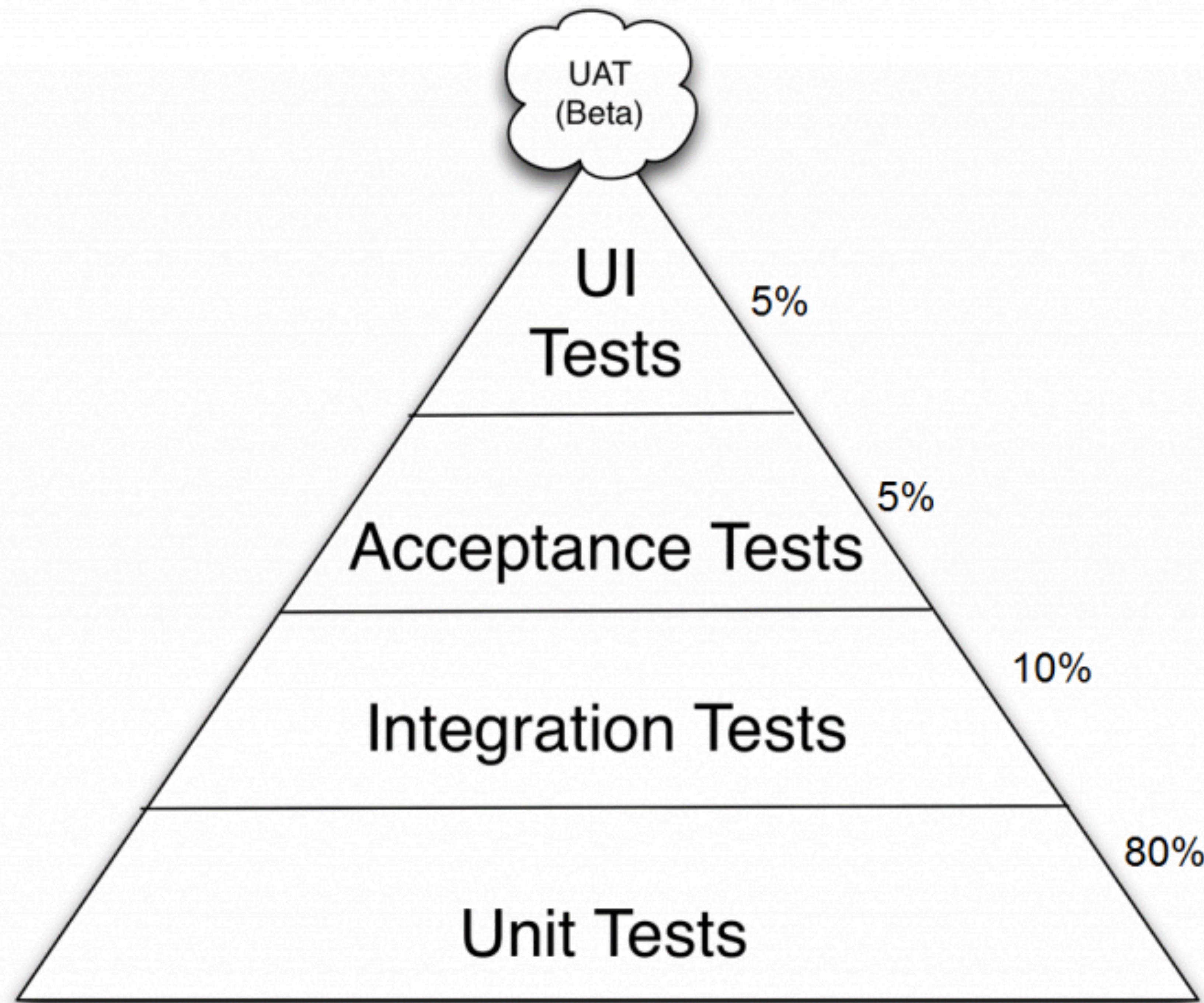
Foreword by Tim Lister

Copyrighted Material

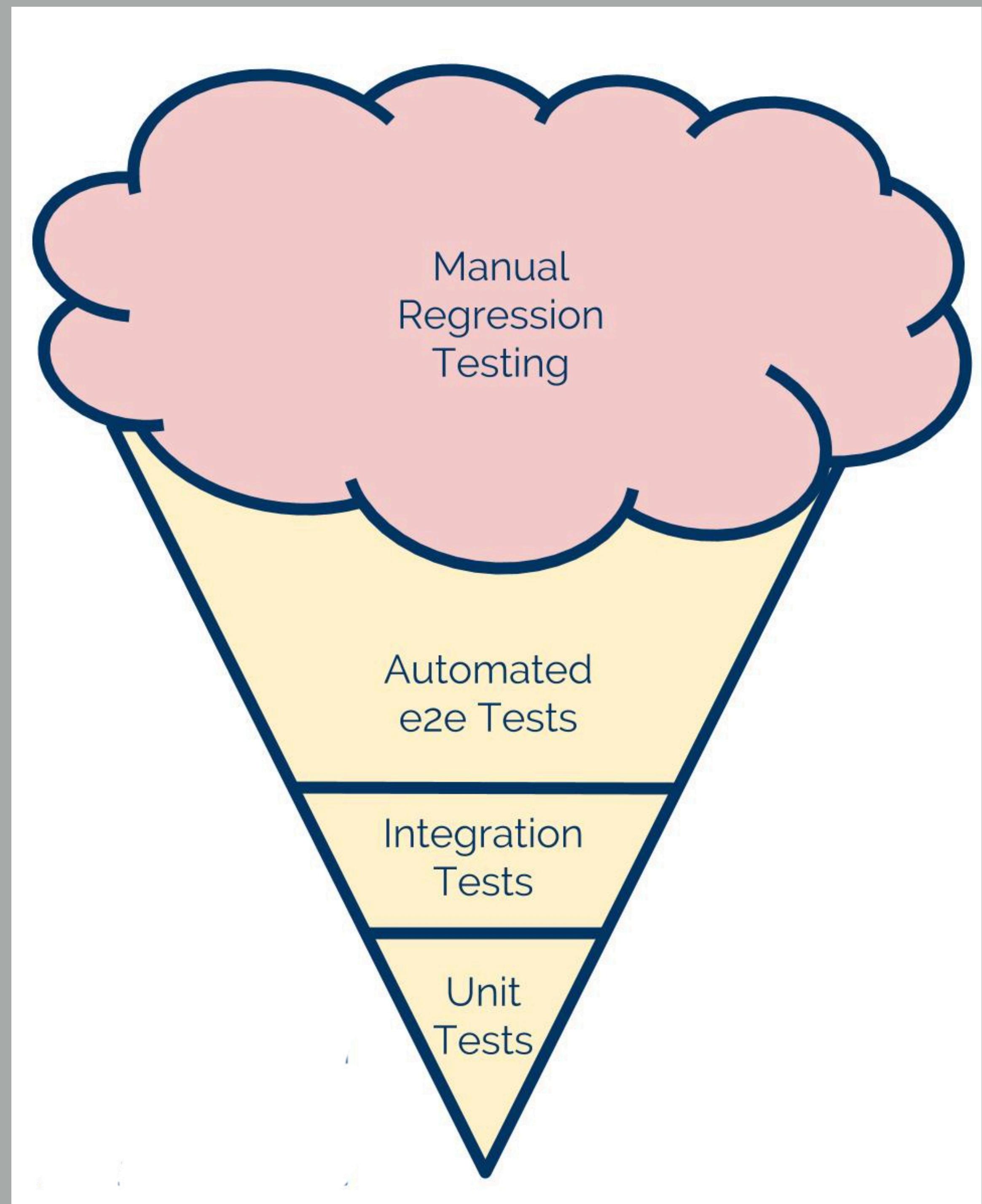
A MIKE COHN  
SIGNATURE  
BOOK  
Mike Cohn



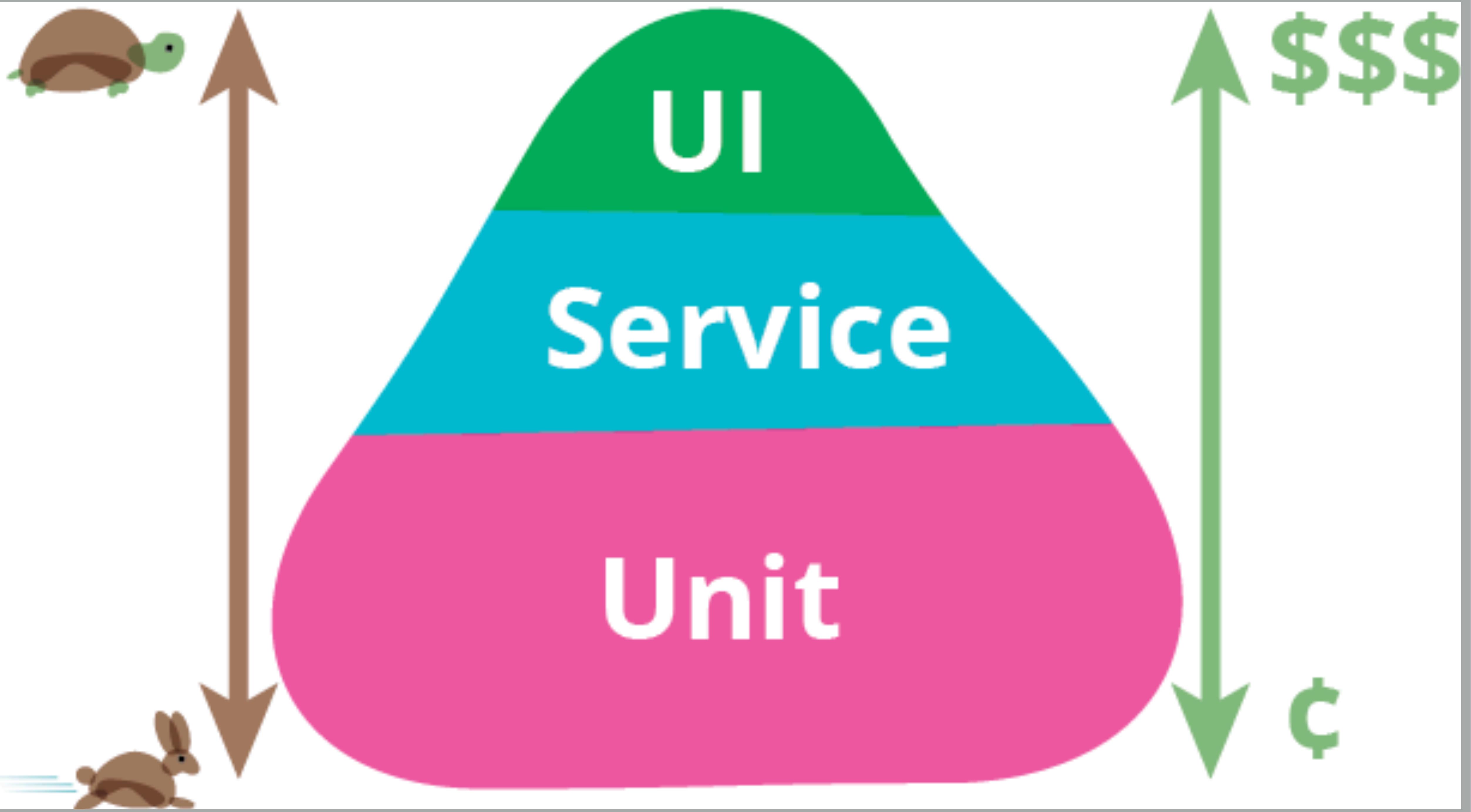
Devteam



**Анти-паттерн**



**Анти-паттерн**



# UI vs Unit

1. Хрупкие
2. Медленней пишутся
3. Медленней дают обратную связь

ui < Unit

# Зачем нужны тесты

## Виды тестов

### “Чистые” тесты

#### Test doubles

#### Tips and tricks

# XCTest

Framework

## XCTest

Create and run unit tests, performance tests, and UI tests for your Xcode project.

SDK

Xcode 5.0+

On This Page

[Overview](#) 

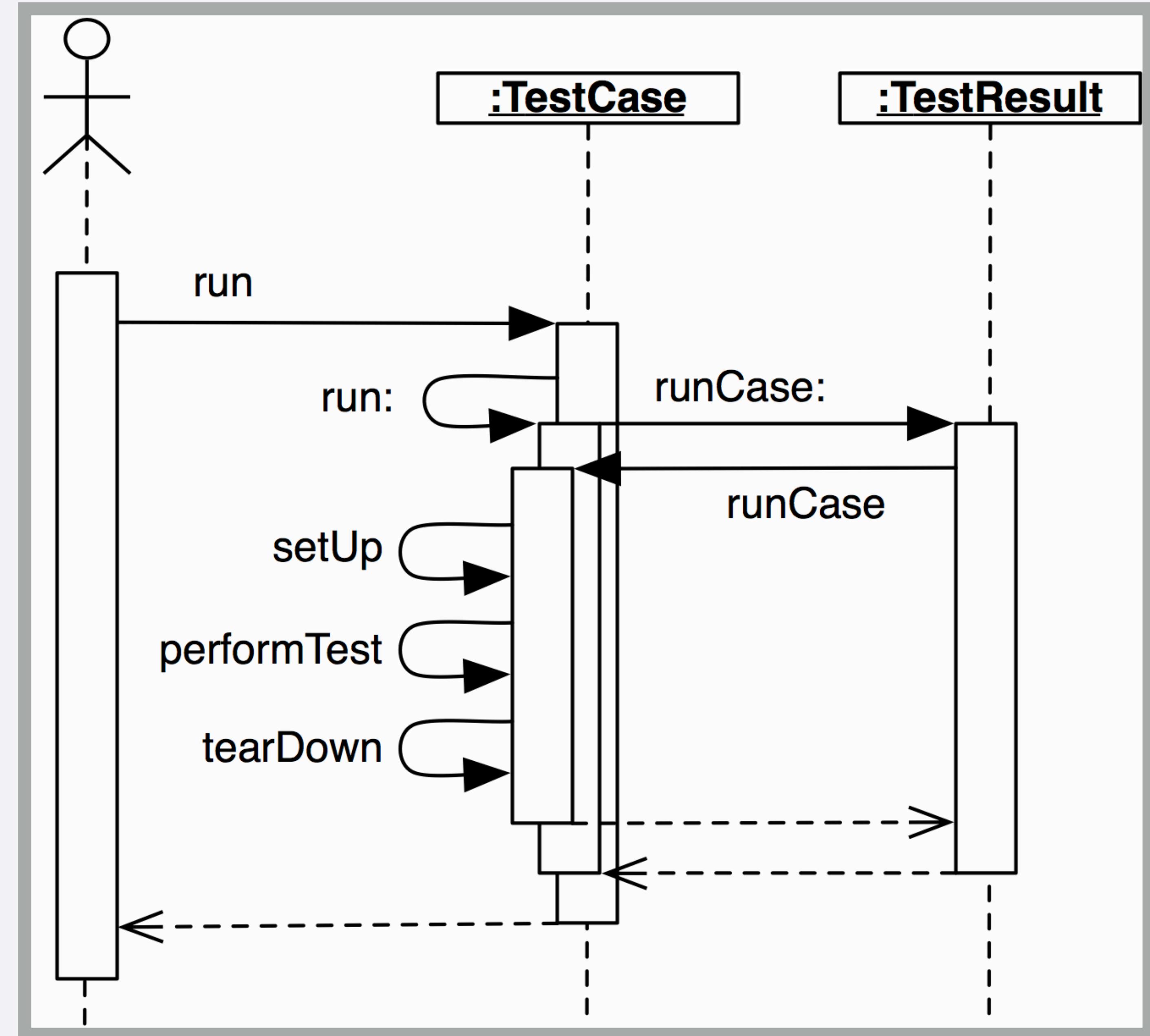
[Topics](#) 

## Overview

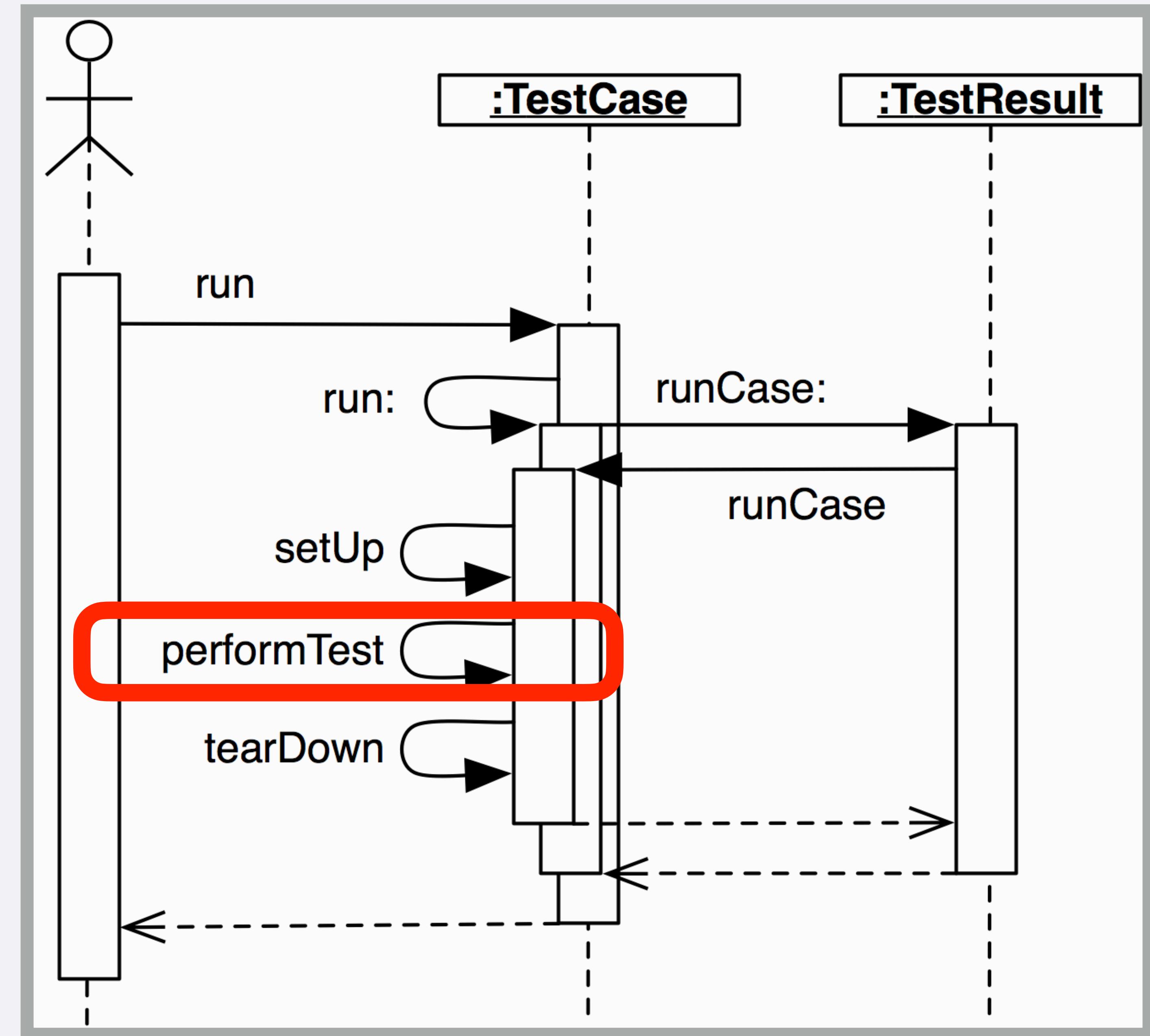
Use the XCTest framework to write unit tests for your Xcode projects that integrate seamlessly with Xcode's testing workflow.

Tests assert that certain conditions are satisfied during code execution, and record test failures (with optional messages) if those conditions are not satisfied. Tests can also measure the performance of blocks of code to check for performance regressions, and can interact with an application's UI to validate user interaction flows.

# xUnit



# Assert



# Assert. Примеры

```
public func XCTFail(_ message: String = "",  
                    file: StaticString = #filePath,  
                    line: UInt = #line)
```

```
public func XCTAssertTrue(_ expression: @autoclosure () throws -> Bool,  
                         _ message: @autoclosure () -> String = "",  
                         file: StaticString = #filePath,  
                         line: UInt = #line)
```

```
public func XCTAssertNotEqual<T>(_ expression1: @autoclosure () throws -> T,  
                                  _ expression2: @autoclosure () throws -> T,  
                                  accuracy: T,  
                                  _ message: @autoclosure () -> String = "",  
                                  file: StaticString = #filePath,  
                                  line: UInt = #line) where T : FloatingPoint
```

# Assert. Примеры

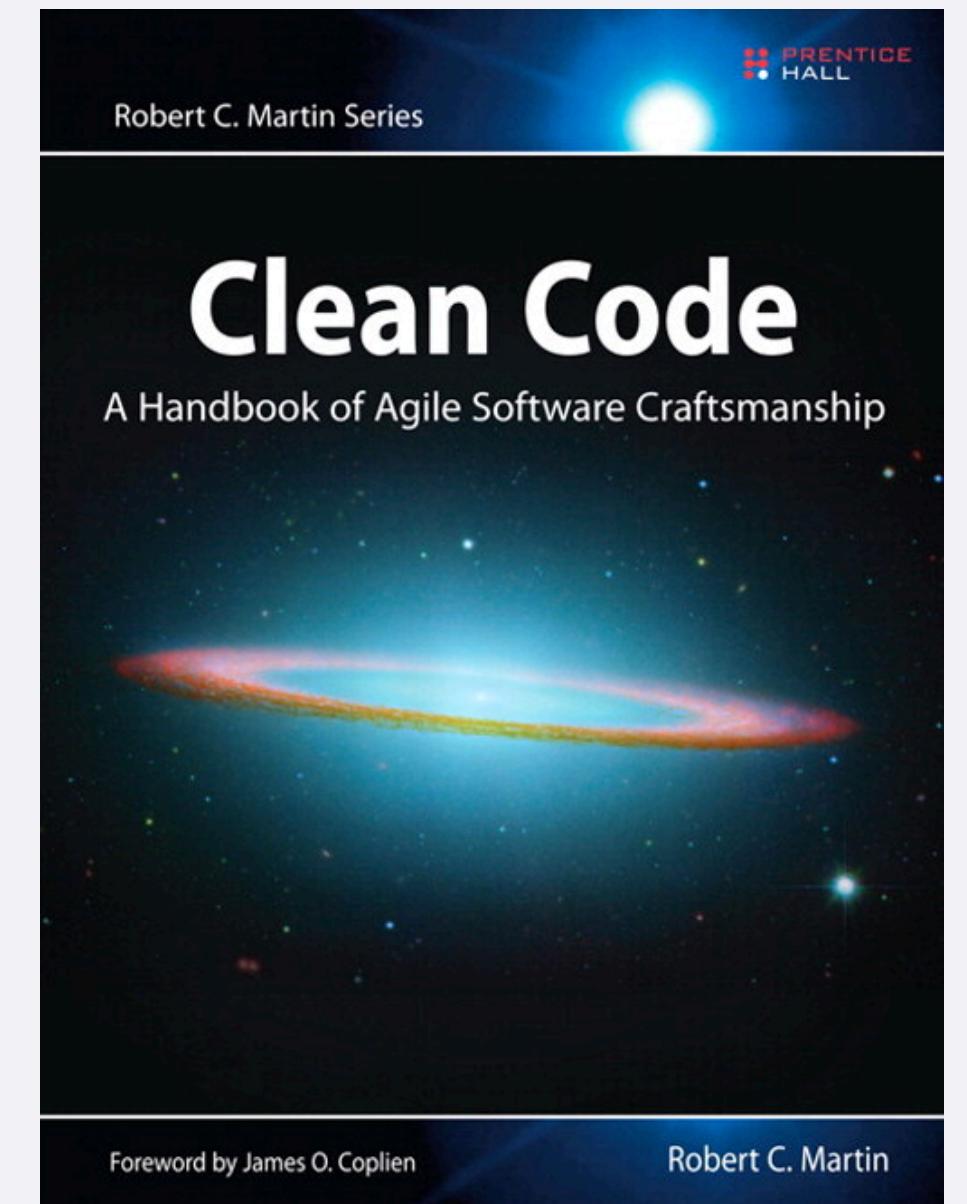
```
func testExample() {  
    XCTAssertEqual(0.01, 0.02, accuracy: 0.01) // success  
    XCTAssertEqual(0.01, 0.02, accuracy: 0.001) // failure  
}
```

# Demo



“... the ratio of time spent **reading versus writing** is well over 10 to 1. We are constantly reading old code as part of the effort to write new code. ...[Therefore,] making it easy to read makes it easier to write.”

Robert C. Martin

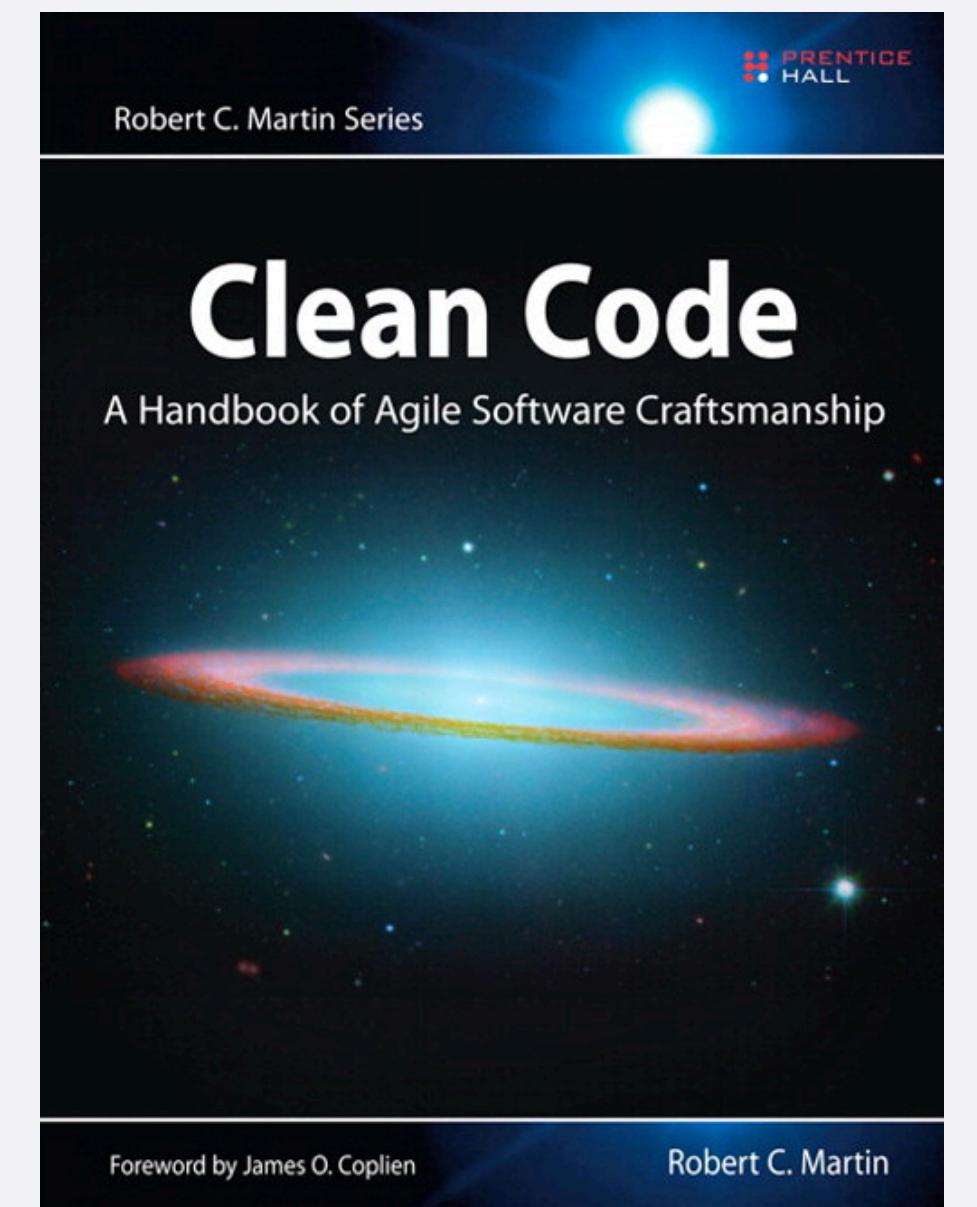


# MAINTAINABILITY



“... the ratio of time spent **reading versus writing** is well over 10 to 1. We are constantly reading old code as part of the effort to write new code. ...[Therefore,] making it easy to read makes it easier to write.”

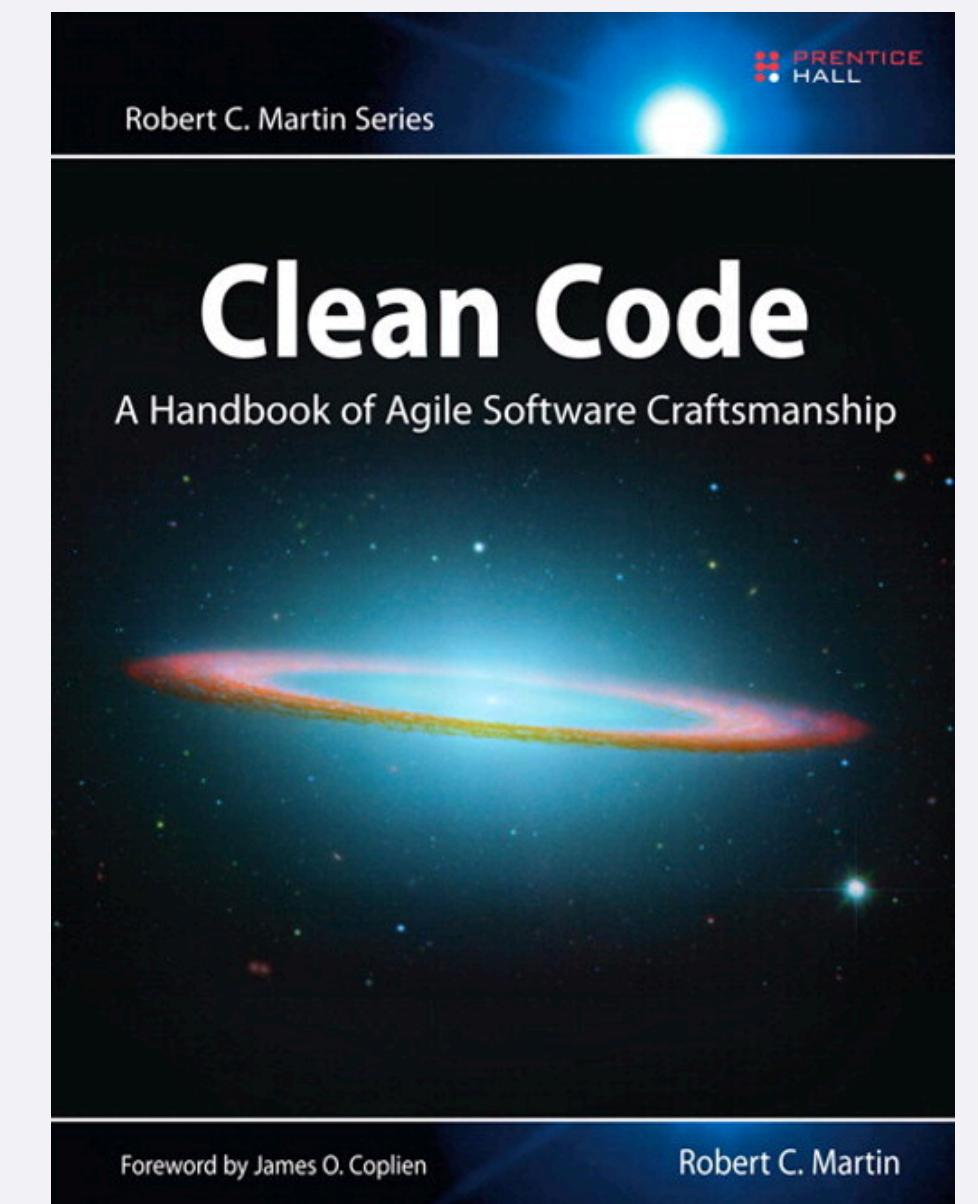
Robert C. Martin



# What makes a clean test? Three things.

## Readability, readability, and readability.

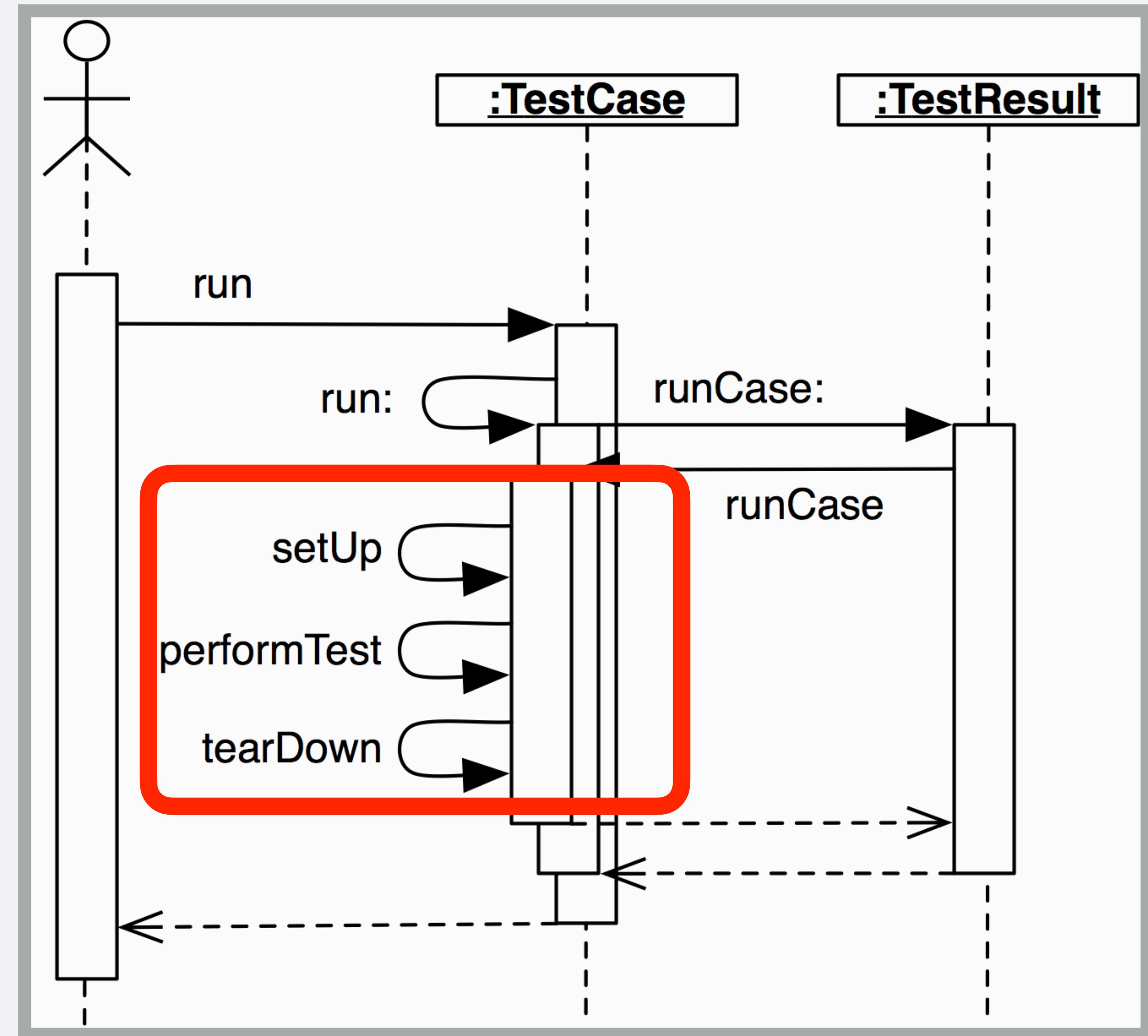
Robert C. Martin



# Чистый тест

- Предметно-ориентированный язык
- Нет лишнего контекста
- Проверка одной истины

# xUnit



# Чистый тест

- Предметно-ориентированный язык
- Нет лишнего контекста
- Проверка одной истины

```
func test1() {  
    let l = Loader()  
  
    let r = l.perform()  
  
    XCTAssertEqual(r.count, 20)  
}
```

```
func test1() {  
    let l = Loader()  
  
    let r = l.perf()  
  
    XCTAssertEqual(r.count, 20)  
}
```

```
func testThatLoaderReceivesRightRubricsAmount() {  
    let rubricsLoader = Loader()  
  
    let rubrics = rubricsLoader.perform()  
  
    XCTAssertEqual(rubrics.count, 20, "<ОПИСАНИЕ>")  
}
```

testThat<SystemUnderTest><ExpectedRequirement>

testThatIt<ExpectedRequirement>

\*

ПроверяемЧто<Система><СоответствуетТребованию>

testThatLoaderReturnsAllObjects

testThatAnimatorHidesSpinnerAtTheEnd

testThatFormatterTranslatesFieldToPONSO

testThatResponseParsingWorksCorrectly

testResponseParsing



test<Precondition><ExpectedOutcome>

test\_<Precondition>\_<ExpectedOutcome>

\*

test\_<Условие>\_<Результат>

**test\_WhenNetworkRequestFails\_TellsUIToDisplayError**

**test\_WhenURLIsNotHTTPS\_RequestIsNotInitialised**

# Чистый тест

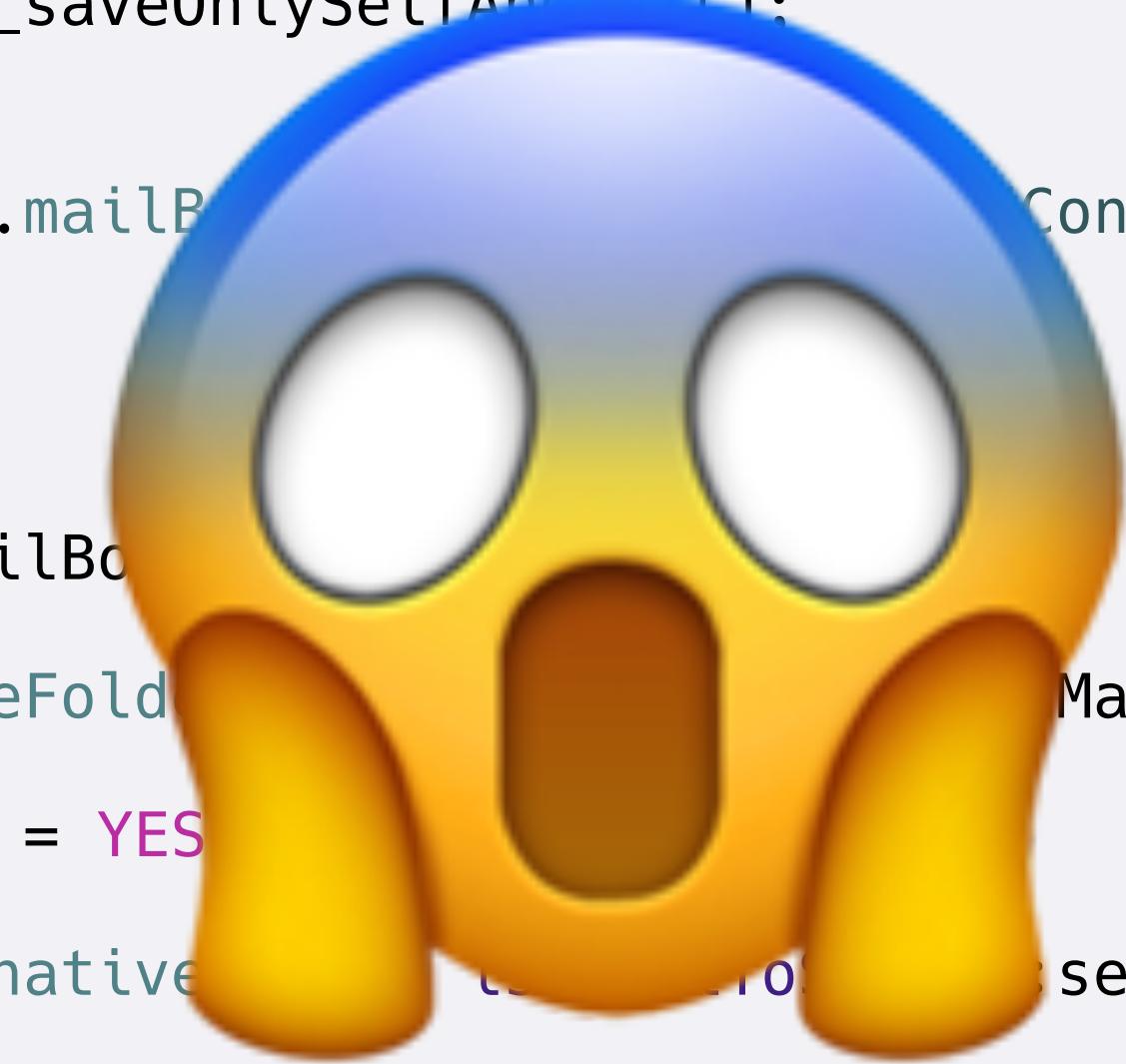
- Предметно-ориентированный язык
- **Нет лишнего контекста**
- Проверка одной истины

```
- (void)testThatMailBoxServiceObtainsCachedConnectedMailBoxConfigList
{
    NSString *firstMailBoxNativeFolderName = @"firstFolder";
    NSString *secondMailBoxNativeFolderName = @"secondFolder";

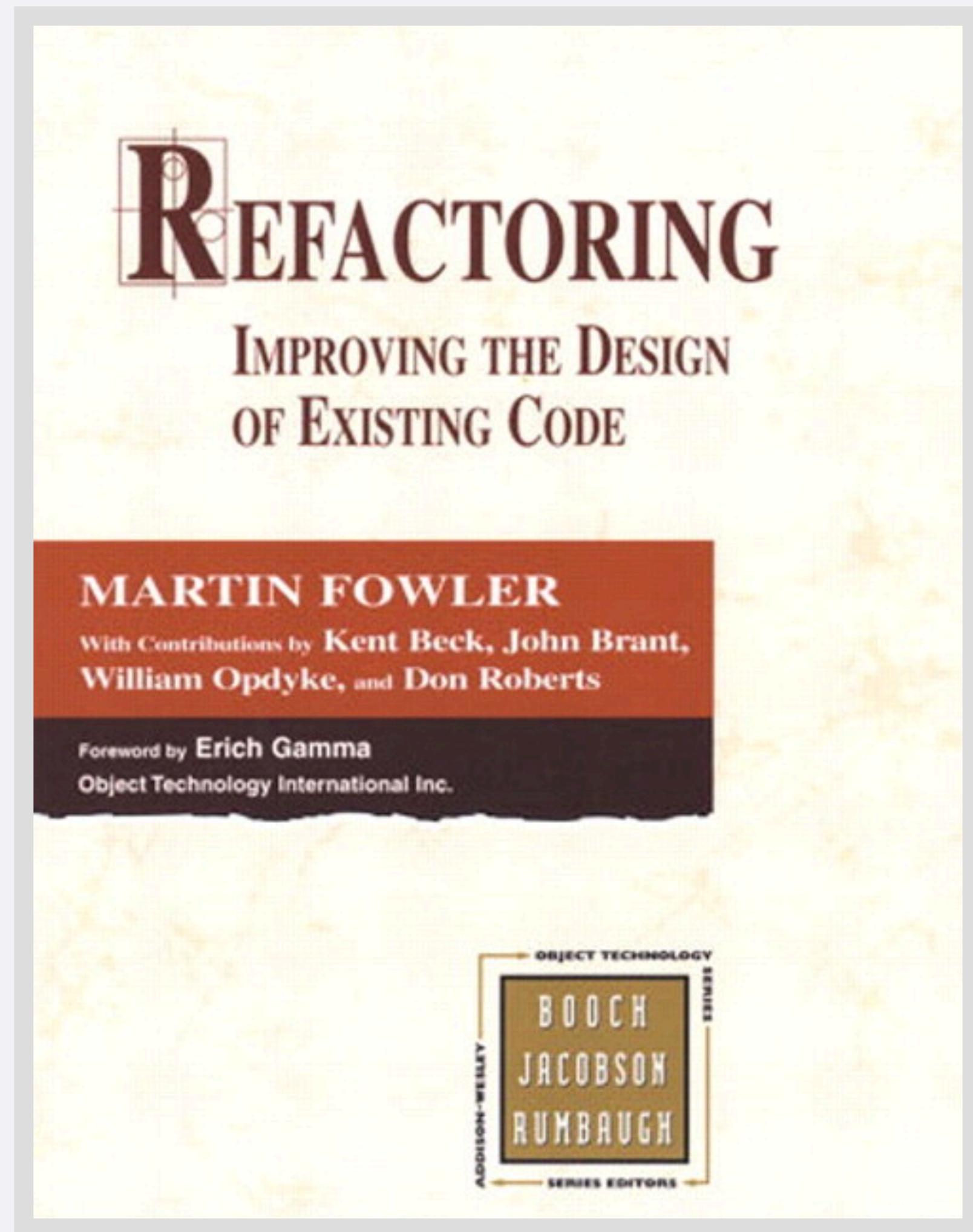
    NSManagedObjectContext *managedObjectContext = [NSManagedObjectContext MR_rootSavingContext];
    [managedObjectContext performBlockAndWait:^{
        XYZMailBox *firstMailBox = [XYZMailBox MR_createEntityInContext:managedObjectContext];
        firstMailBox.nativeFolder = firstMailBoxNativeFolderName;
        XYZMailBox *secondMailBox = [XYZMailBox MR_createEntityInContext:managedObjectContext];
        secondMailBox.nativeFolder = secondMailBoxNativeFolderName;
        [managedObjectContext MR_saveOnlySelfAndWait];
    }];

    NSArray *mailBoxList = [self.mailBoxService obtainConnectedMailBoxConfigList];
    XCTAssertTrue(mailBoxList.count == 2);
    XCTAssertTrue([mailBoxList containsObjectWithNativeFolderName:firstMailBoxNativeFolderName]);
    XCTAssertTrue([mailBoxList containsObjectWithNativeFolderName:secondMailBoxNativeFolderName]);
}

XCTAssertTrue(containsFirstMailBox);
XCTAssertTrue(containsSecondMailBox);
}
```



# Extract Method



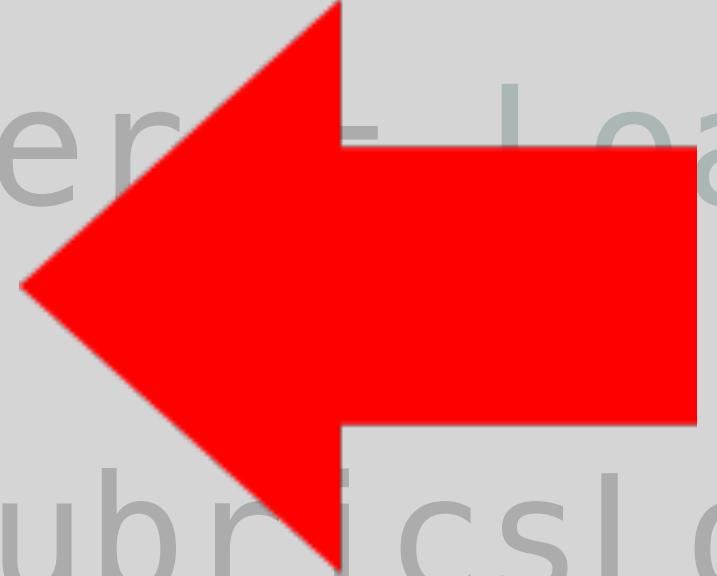
```
func testThatLoaderReceivesRightRubricsAmount() {  
  
    let rubricsLoader = Loader()  
  
    let rubrics = rubricsLoader.perform()  
  
    XCTAssertEqual(rubrics.count, 20, "<ОПИСАНИЕ>")  
}
```

```
// arrange
func testThatLoaderReceivesRightRubricsAmount() {
    let rubricsLoader = Loader()
    // act
    let rubrics = rubricsLoader.perform()

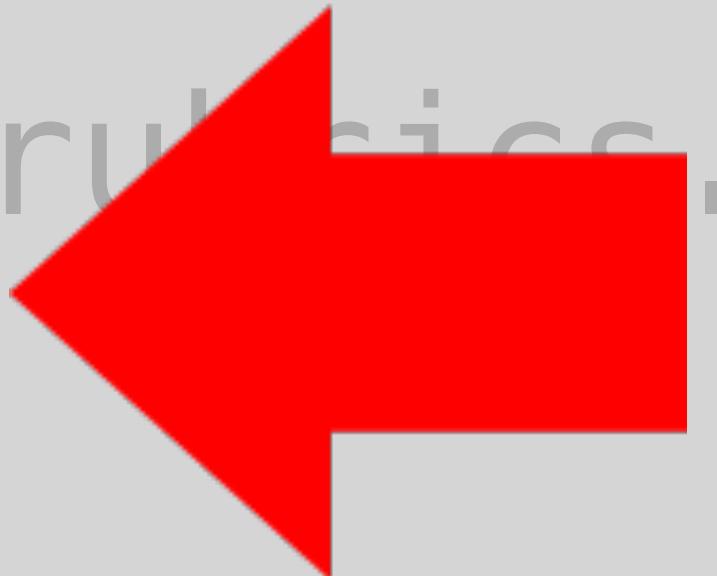
    XCTAssertEqual(rubrics.count, 20, "<ОПИСАНИЕ>")
} // assert
```

```
// arrange ←  
func testThatLoaderReceivesRightRubricsAmount() {  
  
    let rubricsLoader = Loader()  
    // act  
    let rubrics = rubricsLoader.perform()  
  
    XCTAssertEqual(rubrics.count, 20, "<ОПИСАНИЕ>")  
} // assert
```

```
// arrange
func testThatLoaderReceivesRightRubricsAmount() {
    let rubricsLoader = Loader()
// act
    let rubrics = rubricsLoader.perform()
    XCTAssertEqual(rubrics.count, 20, "<ОПИСАНИЕ>")
} // assert
```



```
// arrange
func testThatLoaderReceivesRightRubricsAmount() {
    let rubricsLoader = Loader()
    // act
    let rubrics = rubricsLoader.perform()
    // assert
    XCTAssertEqual(rubrics.count, 20, "<ОПИСАНИЕ>")
}
```



```
func testThatLoaderReceivesRightRubricsAmount() {  
  
    // arrange  
    let rubricsLoader = Loader()  
  
    // act  
    let rubrics = rubricsLoader.perform()  
  
    // assert  
    XCTAssertEqual(rubrics.count, 20, "<ОПИСАНИЕ>")  
}
```

# Другие нотации

- Given / When / Then
- Setup / Exercise / Verify / Teardown

```
func testThatLoaderReceivesRightRubricsAmount() {  
  
    // given  
    let rubricsLoader = Loader()  
  
    // when  
    let rubrics = rubricsLoader.perform()  
  
    // then  
    XCTAssertEqual(rubrics.count, 20, "<ОПИСАНИЕ>")  
}
```

```
- (void)testThatProviderReturnsMainRubric
{
    // setup
    id mock = OCMClassMock([RubricsObject class]);
    OCMStub([mock mainRubric]).andReturn(mock);

    // exercise
    RubricsObject *rubric = [self.rubricProvider mainRubric];

    // verify
    XCTAssertEqualObjects(mock, rubric);

    // teardown
    [mainRubric stopMocking];
}
```

# Чистый тест

- Предметно-ориентированный язык
- Нет лишнего контекста
- Проверка одной истины

# assert one truth

- `XCTFail (format...)`
- `XCTAssertNil (a1, format...)`
- `XCTAssertNotNil (a1, format...)`
- `XCTAssert (a1, format...)`
- `XCTAssertTrue (a1, format...)`
- `XCTAssertFalse (a1, format...)`
- `XCTAssertEqualObjects (a1, a2, format...)`
- `XCTAssertEquals (a1, a2, format...)`
- `XCTAssertEqualsWithAccuracy (a1, a2, accuracy, format...)`
- `XCTAssertThrows (expression, format...)`
- `XCTAssertThrowsSpecific (expression, specificException, format...)`
- `XCTAssertThrowsSpecificNamed (expression, specificException, exceptionName, format...)`
- `XCTAssertNoThrow (expression, format...)`
- `XCTAssertNoThrowSpecific (expression, specificException, format...)`
- `XCTAssertNoThrowSpecificNamed (expression, specificException, exceptionName, format...)`

# Assert one truth

```
func verifyDataTask(urlMatcher: ((URL?) -> Bool)) {  
    XCTAssertEqual(dataTaskCallCount, 1)  
    XCTAssertTrue(urlMatcher(dataTaskArgsURL.first))  
}
```

///...

```
mockURLSession.verifyDataTask(urlMatcher: { url in  
url?.host == "gateway.sber.com" })
```

# F.I.R.S.T.

- Fast** – должны быть способны часто запускаться.
- Isolated** – не должны зависеть от других тестов.
- Repeatable** – каждый раз должны иметь один результат.
- Self-verifying** – должны сами себя проверять.
- Thorough / Timely** – должны разрабатываться одновременно с основным кодом.

# F.I.R.S.T.

- Fast** — должны быть способны часто запускаться.
- Isolated** — не должны зависеть от других тестов.
- Repeatable** — каждый раз должны иметь один результат.
- Self-verifying** — должны сами себя проверять.
- Thorough / Timely** — должны разрабатываться одновременно с основным кодом.

# F.I.R.S.T.

- Fast** — должны быть способны часто запускаться.
- Isolated** — не должны зависеть от других тестов.
- Repeatable** — каждый раз должны иметь один результат.
- Self-verifying** — должны сами себя проверять.
- Thorough / Timely** — должны разрабатываться одновременно с основным кодом.

# F.I.R.S.T.

- Fast** — должны быть способны часто запускаться.
- Isolated** — не должны зависеть от других тестов.
- Repeatable** — каждый раз должны иметь один результат.
- Self-verifying** — должны сами себя проверять.
- Thorough / Timely** — должны разрабатываться одновременно с основным кодом.

# F.I.R.S.T.

- Fast** — должны быть способны часто запускаться.
- Isolated** — не должны зависеть от других тестов.
- Repeatable** — каждый раз должны иметь один результат.
- Self-verifying** — должны сами себя проверять.
- Thorough / Timely** — должны разрабатываться одновременно с основным кодом.

# F.I.R.S.T.

- Fast** — должны быть способны часто запускаться.
- Isolated** — не должны зависеть от других тестов.
- Repeatable** — каждый раз должны иметь один результат.
- Self-verifying** — должны сами себя проверять.
- Thorough / Timely** — должны разрабатываться одновременно с основным кодом.

# Требования

- Скорость
- Надежность

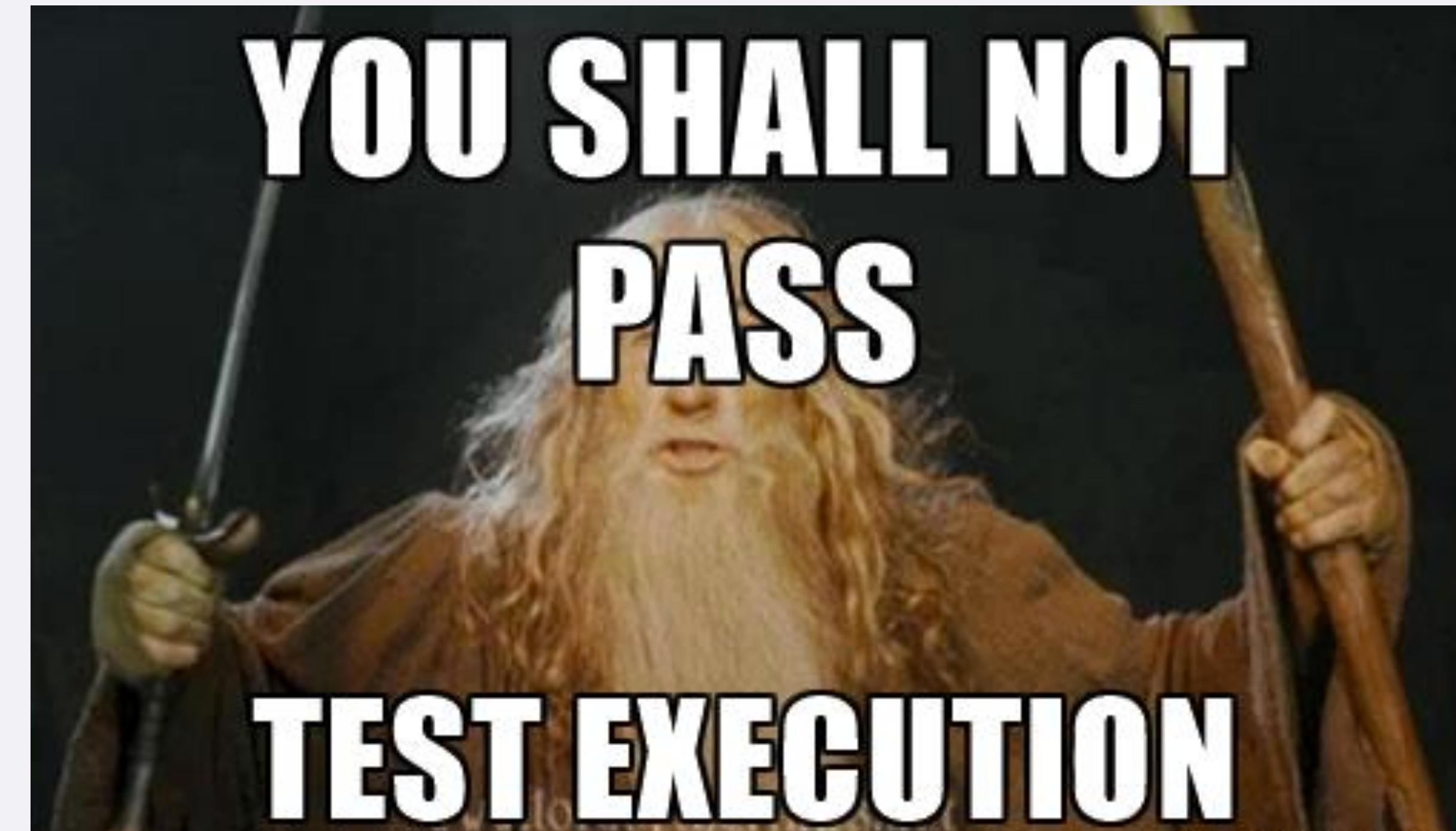
# Требования

- Скорость
- Надежность



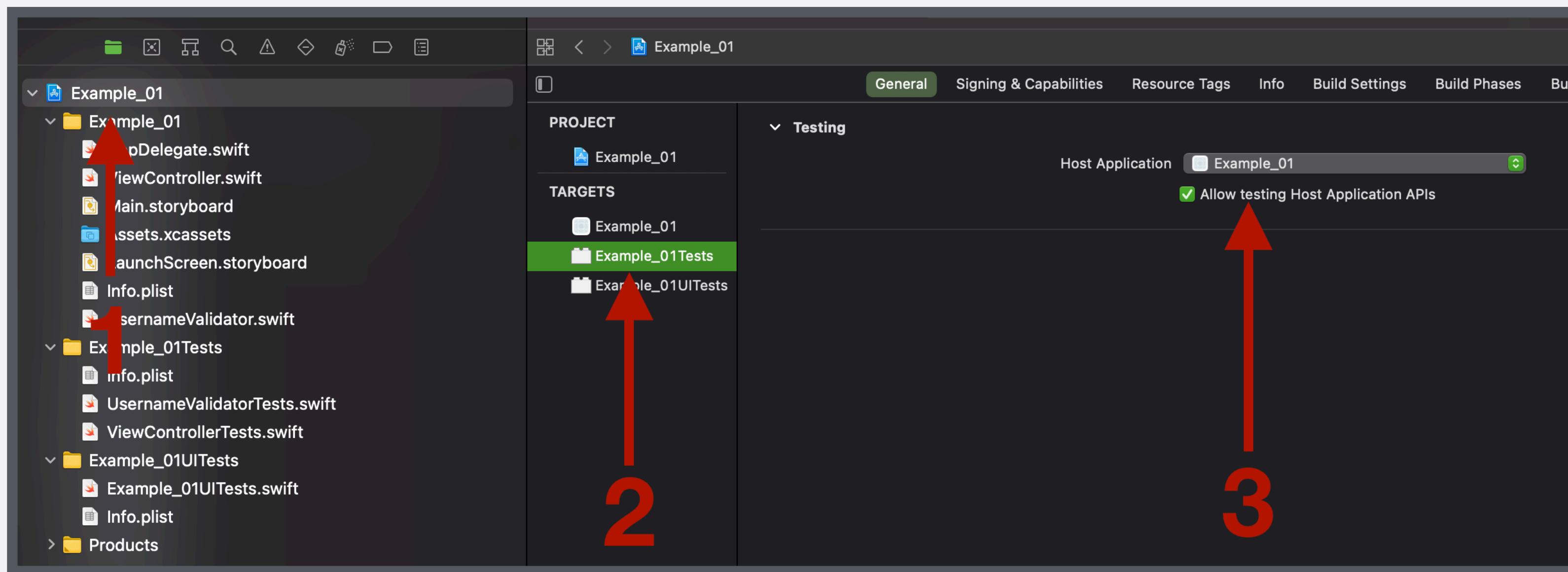
# Требования

- Скорость
- Надежность



# Домашнее задание

Не включая **HostApplication**, добиться прохождения модульных тестов в тестовом проекте



## СРОК - до следующей лекции