Unit Testing & Isolation

Testing Essentials, Testing Levels, Unit Testing, Mocking

SoftUni Team

Technical Trainers

Software University

SoftUni Foundation

Software University

http://softuni.bg

Table of Contents

1. Testing
2. Unit Testing
   - Mocking
   - Arrange
   - Act
   - Assert

Have a Question?

sli.do

#java-web

Testing

Attention please!

Testing

- **Testing** is an important part of the application lifecycle
  - In our ever-changing environment, testing is a necessity
  - New features need to be verified, before delivered to the clients
- **Testing** is a wide area of application development
  - There are several **levels** of testing
  - It does not affect only programmers
  - It has many **concepts** of development
  - There are **different types** of testing

Unit Testing

Unit Testing

- **Unit Testing** is:
  - A level of software testing where individual components are tested
    - The purpose is to validate that each unit performs as designed
  - The lowest level of software testing
  - Normally performed by software developers themselves
  - Often neglected, but it is in fact, the most important level of testing
- **Unit Testing** is often isolated in order to ensure individual testing
  - Testing Frameworks often provide *mocking* functionality

Unit Testing - Mocking

- ***Mocking*** *– something made as an imitation*
- **Mocking** is a software practice, primarily used in **Unit Testing**
  - An object under test may have **dependencies** on other objects
  - To **isolate** the behavior, the other objects are replaced
    - The replacements are **mocked objects**
    - The mocked objects **simulate** the behavior of the **real objects**
  - Useful if the real objects are impractical / incorporate to the unit test
- Basically, **Mocking** is creating objects that **simulate behavior**

Unit Testing - Benefits

- Unit testing increases confidence in changing / maintaining code
- Development is faster:
  - Verifying the correctness of new functionality is not manual
  - Localizing bugs, introduced in development is much faster
- The cost of fixing a defect detected during Unit Testing is lesser
  - Compared to the cost of the bug if it reaches the clients
- Debugging is easy
  - When a test fails, only the latest changes need to be checked
- The code is modular and reusable (necessary for Unit testing)

Unit Testing a Web Application

Simple Demonstration

Unit Testing

- **Unit Testing** web apps is pretty much like casual unit testing
  - Writing test methods to test classes and methods (functionalities)
    - Testing individual code components (**units**)
    - Independently from the **infrastructure**
  - You still use the same testing frameworks as in casual unit testing
- When using a web frameworks such as **Spring MVC**
  - Built-in logic does not need to be tested
    - It is already tested during the development of the framework itself
  - You still need to test your custom functionality

Unit Testing

- Testing a simple service with mocking in an **Spring MVC** app

```
@Entity
@Table(name = "users")
public class User {
    private String id;
    private String username;
    private String password;

    ...
}
```

```
@Repository
public interface UserRepository
extends JpaRepository<User, String> {
    User findByUsername(String username);
}
```

```
public interface UserService {
    User getUserByUsername(String username);
}
```

```
@Service
public interface UserServiceImpl {
    ...
    public User getUserByUsername(String username) {
        return this.userRepository.findByUsername(username);
    }
}
```

Unit Testing

- Testing a simple service with mocking in an **Spring MVC** app

```
public class UserServiceTests {
    private User testUser;
    private UserRepository mockedUserRepository;

    @Before
    public void init() {
        this.testUser = new User() {{
            setId("SOME_UUID");
            setUsername("Pesho");
            setPassword("123");
        }};

        this.mockedUserRepository = Mockito.mock(UserRepository.class);
    }

    ...
}
```

Unit Testing (Arrange)

- Testing a simple service with mocking in an **Spring MVC** app

```
public class UserServiceTests {
    ...
    @Test
    public void userService_GetUserWithCorrectUsername_ShouldReturnCorrect() {
        // Arrange
        Mockito.when(this.mockedUserRepository
                .findByUsername("Pesho"))
                .thenReturn(this.testUser);

        UserService userService = new UserServiceImpl(this.mockedUserRepository);

        User expected = this.testUser;
        ...
    }
}
```

Unit Testing (Act)

- Testing a simple service with mocking in an **Spring MVC** app

```
public class UserServiceTests {
    ...

    @Test
    public void userService_GetUserWithCorrectUsername_ShouldReturnCorrect() {
        ...

        // Act
        User actual = userService.getUserByUsername("Pesho");

        ...
    }
}
```

Unit Testing (Assert)

- Testing a simple service with mocking in an **Spring MVC** app

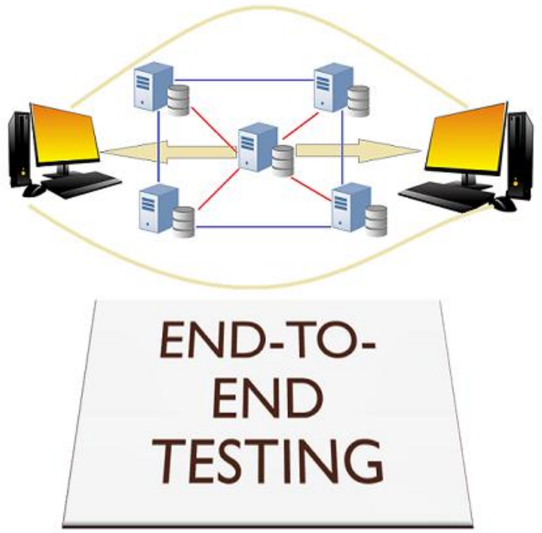
```
public class UserServiceTests {
    ...

    @Test
    public void userService_GetUserWithCorrectUsername_ShouldReturnCorrect() {
        ...

        // Assert
        Assert.assertEquals("Broken...", expected.getId(), actual.getId());
        Assert.assertEquals("Broken...", expected.getUsername(), actual.getUsername());
        Assert.assertEquals("Broken...", expected.getPassword(), actual.getPassword());
    }
}
```

Testing

- **Web applications** also need testing for "**unintentional features**"
  - Controllers, Services, Custom Components etc.
- Different **components** of the application are tested differently
  - They are tested on different levels
    - **Unit** testing
    - **Integration** testing
    - **End-to-End** testing
- Every component of the application must be tested



Testing

- There are also different concepts and practices of test development
  - **Code-first** approach (The usual Development)
  - **Test-first** approach (Test-Driven Development)
- Each has its own **advantages** and **disadvantages**
  - The **Code-first** approach ensures **flexibility** & **fast** development
  - The **Code-first** approach requires **additional refactoring**
  - The **Test-first** approach ensures **quality** and **edge case coverage**
  - The **Test-first** approach is **complicated** and is an "**initial delay**"

Testing

- Some of the most common levels of Software Testing

Testing Level	Description
Unit Testing	Tests **Individiual components** of code, independent from the infrastructure
Component Testing	Testing of **multiple functionalities** (a single component)
Integration Testing	Testing of all **integrated modules** to verify the combined functionality
System Testing	Tests the **system** as a **whole**, once all the components are integrated
Regression Testing	Ensures that a fixed bug **won't happen again**
Acceptance Testing	Tests if the **product** meets the **client**'s **requirements**. Purely done by QAs
Load / Stress Testing	Test the application's limits by attempting **large data processing** and introducting **abnormal circumstances** and **conditions** (edge cases)
Security Testing	Test if the application has any **security flaws** and **vulnerabilities**
Other Types of Testing	Manual, automation, UI, performance, black box, end-to-end testing, etc.

Testing

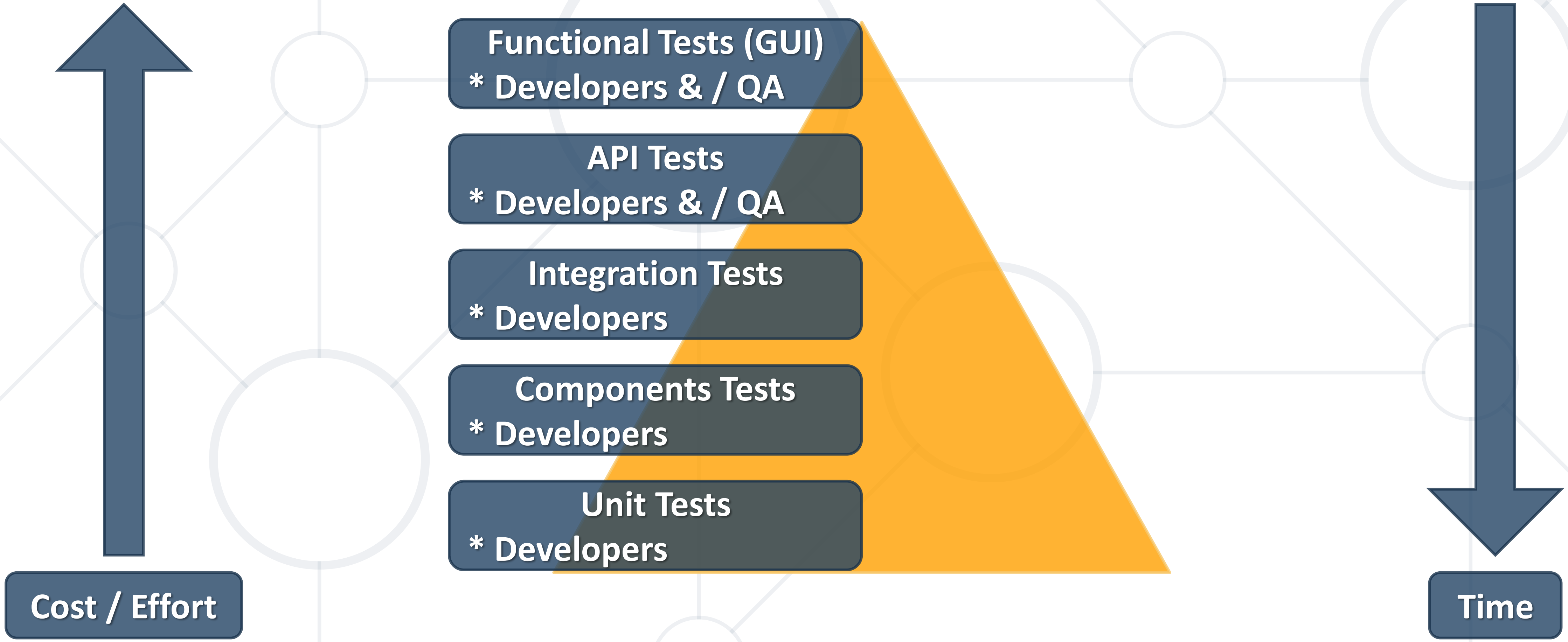
- Why should we bother testing an application on different levels?
  - Isn't Unit testing sufficient enough?
- Unit testing ensures the correctness of a particular unit
  - Not testing all components, as a whole, may lead to false results
    - A single unit may function correctly, independent of the infrastructure
  - Combining components and testing them collectively is necessary
  - Every level of testing is essential to an application's lifecycle
- The questions above are easily answered by this image.

Testing

- Why should we bother with Unit testing?
  - Can’t we just use Integration testing?..
- Unit testing is essential!
  - Helps reduce the scope, when searching for errors
  - They run faster, they fail faster.
    - Unit tests are fast – you are more likely to run them frequently
    - Frequent checks ensure correctness of the functionality
  - Provide unit documentation (in a way)

Testing

- Different Testing levels require different time and resources



Testing

Live Demonstration

Summary

- **Testing** is an important part of the application lifecycle
  - In our ever-changing environment, testing is a necessity
  - New features need to be verified, before delivered to the clients
- **Unit Testing** is:
  - A level of software testing where individual components are tested
    - The purpose is to validate that each unit performs as designed
  - The lowest level of software testing

Questions?

SoftUni

Software University

SoftUni Svetlina

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

SoftUni Diamond Partners

SoftUni Foundation

XSsoftware

SBTech
we know sports

telenor

SoftwareGroup
doing it right

NETPEAK

SmartIT

Postbank
Решения за твоето утре

SUPER HOSTING ®.BG

INDEAVR
Serving the high achievers

INFRAGISTICS®

LIEBHERR

æternity

codexio

SoftUni Organizational Partners

SoftUni Foundation

ИНФОРМАЦИОННО ОБСЛУЖВАНЕ

OneBit SOFTWARE

WORLD OF MYTHS

Lukanet.com

codexio

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education and Employment Opportunities
  - softuni.bg
- Software University Foundation
  - http://softuni.foundation/
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License