Unit and Integration Testing

Testing Essentials, Testing Levels, Unit Testing, Mocking

SoftUni Team

Technical Trainers

Software University

Software University

https://softuni.bg

SoftUni

Table of Contents

Have a Question?

sli.do

#java-web

Attention Please!

Testing

Testing (1)

- **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 (2)

Software University

- **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**
  - 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**
  - Often isolated in order to ensure individual testing

Mocking

- 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**

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 code is modular and reusable (necessary for Unit testing)

Simple Demonstration

Unit Testing a Web Application

Unit Testing

- **Unit Testing** for web apps is similar to the unit tests we've done
  - 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

Unit Testing (1)

- 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 (2)

- 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 class UserServiceImpl implements UserService {
    ...
    public User getUserByUsername(String username) {
        return this.userRepository.findByUsername(username);
    }
}
```

Unit Testing (3)

- 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");

        ...
    }
}
```

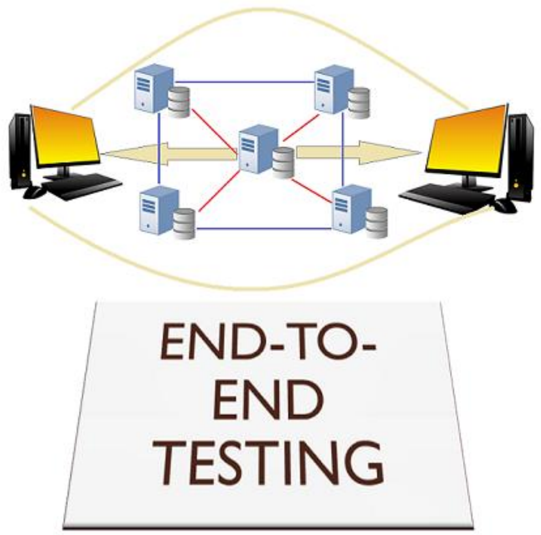
Unit Testing (Assert)

- 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 (1)

- **Web applications** also need testing for:
  - Controllers
  - Services
  - Custom Components etc.



Testing (2)

- 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 the Web Layer

Testing Controller

- UserController example

```
@Controller
@RequestMapping("/users")
public class UserController {
        // Inject UserService in constructor
    @GetMapping("/{id}")
    public ModelAndView getById(@PathVariable("id") Long id, ModelAndView
modelAndView) {
        modelAndView.addObject("user", this.userService.findById(id));
        modelAndView.setViewName("one");
        return modelAndView;
    }
    @GetMapping("/all")
    public ModelAndView findAll(ModelAndView modelAndView){
        modelAndView.addObject("users", this.userService.findAll());
        modelAndView.setViewName("all");
        return modelAndView;
    }
}
```

MockMvcResultMatchers Methods (1)

- **request()**
  - Access to request-related assertions
- **handler()**
  - Access to assertions for the handler that handled the request
- **model()**
  - Access to model-related assertions
- **view()**
  - Access to assertions on the selected view

MockMvcResultMatchers Methods (2)

- **flash()**
  - Access to flash attribute assertions
- **status()**
  - Access to response status assertions
- **header()**
  - Access to response header assertions
- **content()**
  - Access to response body assertions

Simple test examples (1)

```
@SpringBootTest
@AutoConfigureMockMvc
public class UserControllerTests {
    @Autowired
    private MockMvc mockMvc;
    @Test
    public void when_getOneStudents_returnFirst() throws Exception {
        mockMvc
                .perform(MockMvcRequestBuilders
                        .get("/users/1"))
                .andExpect(status().isOk())
                .andExpect(view().name("one"))
                .andExpect(model().attributeExists("user"));
}
```

Simple test examples (2)

```
@SpringBootTest
@AutoConfigureMockMvc
public class AuthorsControllerTest {
        // @Autowired MockMvc and AuthorRepository
  @BeforeEach
  public void setUp() { // Add two test authors in repository }
  @AfterEach
  public void tearDown() { authorRepository.deleteAll(); }
  @Test
  public void testGetAuthorsCorrect() throws Exception {
    this.mockMvc.perform(get("/authors")).
        andExpect(status().isOk()).
        andExpect(jsonPath("$", hasSize(2))).
        andExpect(jsonPath("$.[0].name", is(author1Name))).
        andExpect(jsonPath("$.[1].name", is(author2Name)));  }
```

Simple test examples (3)

- Testing with MockUser

```
@Test
@WithMockUser("customUsername")
public void getMessageWithMockUserCustomUsername() {
        String message = messageService.getMessage();
...
}
```

- Specific Roles

```
@Test
@WithMockUser(username="admin",roles={"USER","ADMIN"})
public void getMessageWithMockUserCustomUser() {
        String message = messageService.getMessage();
        ...
}
```

Testing (1)

- There are also different concepts and practices of test development
  - **Code-first** approach (The usual Development)
  - **Test-first** approach (Test-Driven Development)

Testing (2)

- 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**"

Common levels of Software Testing (1)

- Some of the most common levels of Software Testing

Testing Level	Description
Unit Testing	Tests Individual components of code, independent from the infrastructure
Component Unit 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

Common levels of Software Testing (2)

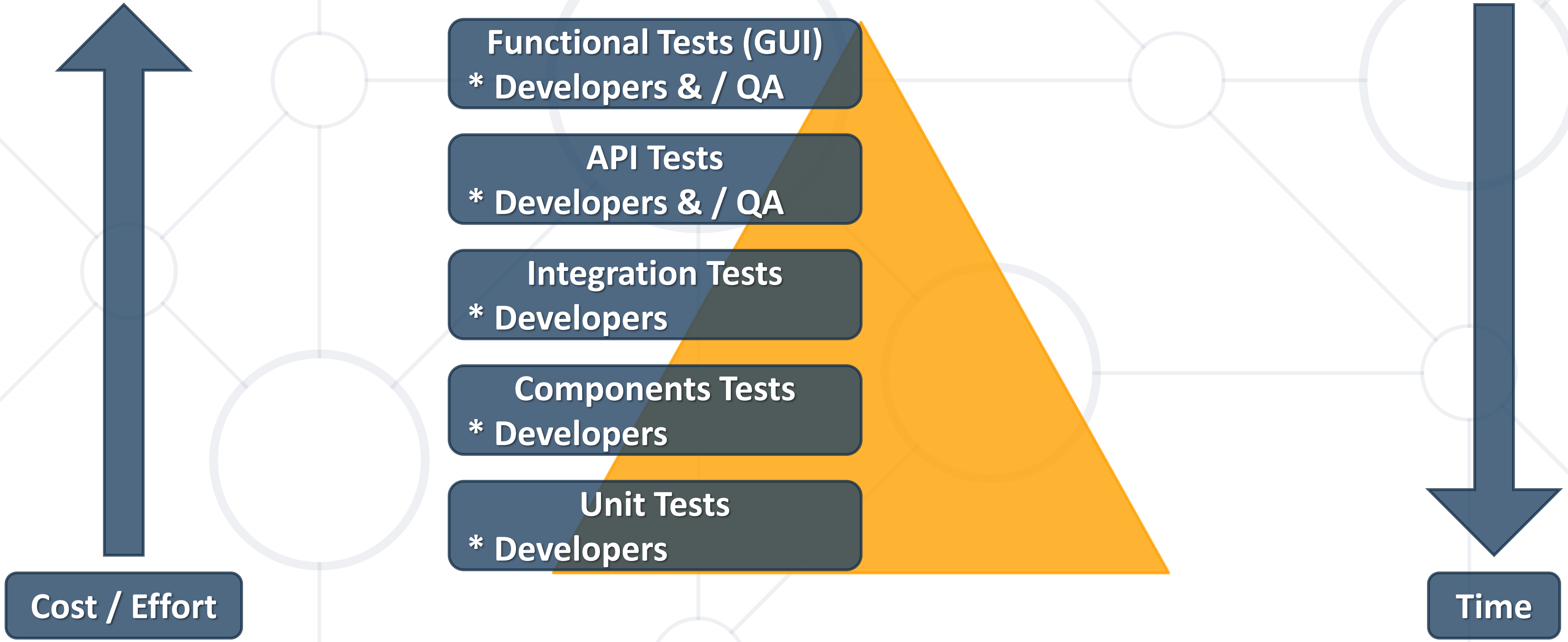
Testing Level	Description
Regression Testing	Testing that recent program or code change has not adversely affected existing features.
Acceptance Testing	Tests if the product meets the client's requirements. Purely done by QAs
Load / Stress Testing	Test the application's limits by attempting large data 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

- Unit testing ensures the correctness of a particular unit
  - Not testing all components 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

Different Testing levels

- Different Testing levels require different time and resources



Live Demonstration

Testing

Summary

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

Software University

Questions?

SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

Finance Academy

SoftUni Diamond Partners

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

Flutter International

INDEAVR Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

BOSCH

Postbank Решения за твоето утре

PHAR VISION

SmartIT

DXC TECHNOLOGY

createX

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License