Sprint 3: Testing and Final Review

Lecture 5 Notes

School of Computing Communication and Media Studies

Masoud Hamad

CS2113 – Software Development Project
Academic Year 2025

Contents

1 Introduction to Sprint 3

Sprint 3 is the final sprint of the course. Teams work on new user stories while focusing heavily on **testing practices**. Additionally, each team member must complete:

- **Peer Review:** Assessing themselves and teammates
- **Final Report:** Individual reflection on the course

Warning

Both peer review and final report are **required to pass the course**. Submitting either after the Sprint deadline will decrease the personal grade.

2 Why Testing Matters

Testing ensures that code works as intended. Without automated tests, developers must manually verify all features after every change—an impractical approach at scale.

2.1 The Problem: Regression Bugs

Regression bugs occur when features that previously worked break after code changes. Manual testing becomes impractical because:

- Every new feature requires re-testing all existing features
- Time required grows exponentially with project size
- Human testers make mistakes and miss edge cases

2.2 The Solution: Automated Tests

Automated tests:

- Run quickly and consistently
- Catch regressions immediately
- Document expected behavior
- Enable confident refactoring

3 The Test Pyramid

Martin Fowler's Test Pyramid categorizes tests into three levels:

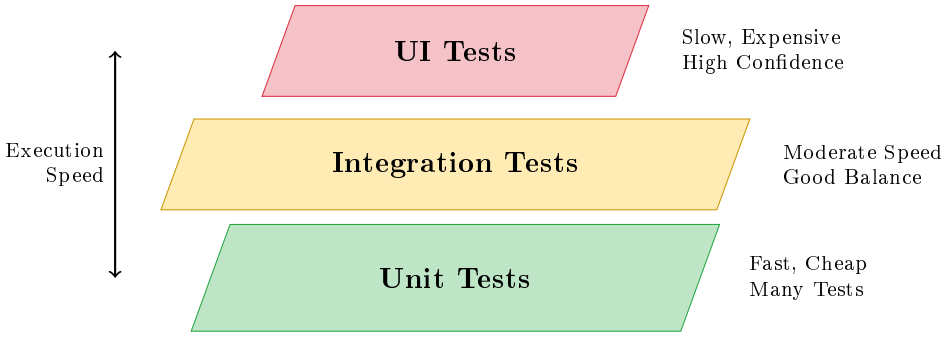

Figure 1: The Test Pyramid

3.1 Trade-offs

Moving up the pyramid:

- \+ Increases reliability/confidence
- – Tests become harder to maintain
- – Tests become slower to execute
- – Tests become more expensive to implement

4 Types of Tests

4.1 Unit Tests

Unit Tests test individual methods in isolation. They never integrate with other code parts like databases.

```java
@Test
void calculateWordsReturnsSingleWord() {
    String message = "Hello";
    assertEquals(1, MessageUtils.calculateWords(message));
}

@Test
void calculateWordsReturnsManyWords() {
    String message = "Hello world";
    assertEquals(2, MessageUtils.calculateWords(message));
}

@Test
void calculateWordsReturnsZeroForEmpty() {
    String message = "";
    assertEquals(0, MessageUtils.calculateWords(message));
}
```

Pros:

- Simple to implement and maintain
- Fast execution

Cons:

- Limited reliability about overall application functionality

4.2 Integration Tests

Integration Tests test that different application parts work together when integrated. Commonly test database operations and REST API endpoints.

Key Insight

"Write tests. Not too many. Mostly integration." — Guillermo Rauch, CEO of Vercel
Integration tests provide the best "bang for your buck" in confidence that applications work as intended.

4.3 UI Tests (End-to-End)

UI Tests simulate real user interactions through the browser: navigating pages, filling forms, clicking buttons, verifying content.

Pros:

- High reliability that application works end-to-end

Cons:

- Difficult to implement
- Laborious to maintain
- Slow execution time

5 Testing REST API Endpoints

5.1 Test Class Structure

```
@SpringBootTest
@AutoConfigureMockMvc
public class QuizRestControllerTest {

    @Autowired
    private QuizRepository quizRepository;

    @Autowired
```

```
9       private MockMvc mockMvc;
10
11      private ObjectMapper mapper = new ObjectMapper();
12
13      @BeforeEach
14      void setUp() {
15          quizRepository.deleteAll();
16      }
17  }
```

5.2 Key Annotations

- `@SpringBootTest` – Loads Spring context for tests
- `@AutoConfigureMockMvc` – Enables MockMvc injection
- `@BeforeEach` – Runs before each test (clean database)
- `@Test` – Marks a test method

5.3 Test Independence

Warning

Each test should start with an **empty database** to ensure test independence. Tests should not depend on other tests' data or execution order.

6 Arrange-Act-Assert Pattern

Structure each test in three phases:

1. **ARRANGE** – Set up test inputs, objects, database state
2. **ACT** – Execute the behavior being tested
3. **ASSERT** – Verify expected outcomes

Figure 2: Arrange-Act-Assert Pattern

7 GET Request Tests

```java
@Test
public void getAllQuizzesReturnsEmptyList() throws Exception {
    // Act & Assert
    this.mockMvc.perform(get("/api/quizzes"))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$", hasSize(0)));
}

@Test
public void getQuizByIdReturnsQuiz() throws Exception {
    // Arrange
    Quiz quiz = new Quiz("Test Quiz", "Description");
    quizRepository.save(quiz);

    // Act & Assert
    this.mockMvc.perform(get("/api/quizzes/" + quiz.getId()))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.name").value("Test Quiz"))
        .andExpect(jsonPath("$.id").value(quiz.getId()));
}

@Test
public void getQuizByIdReturnsNotFound() throws Exception {
    // Act & Assert
    this.mockMvc.perform(get("/api/quizzes/999"))
        .andExpect(status().isNotFound());
}
```

8 POST Request Tests

```java
@Test
public void createQuizSavesValidQuiz() throws Exception {
    // Arrange
    CreateQuizDto quiz = new CreateQuizDto("New Quiz");
    String requestBody = mapper.writeValueAsString(quiz);

    // Act
    this.mockMvc.perform(post("/api/quizzes")
            .contentType(MediaType.APPLICATION_JSON)
            .content(requestBody))
        // Assert
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.name").value("New Quiz"));
```

```
14
15      // Verify database
16      List<Quiz> quizzes = quizRepository.findAll();
17      assertEquals(1, quizzes.size());
18      assertEquals("New Quiz", quizzes.get(0).getName());
19  }
20
21  @Test
22  public void createQuizRejectInvalidQuiz() throws Exception {
23      // Arrange
24      CreateQuizDto quiz = new CreateQuizDto(""); // Empty
            name
25      String requestBody = mapper.writeValueAsString(quiz);
26
27      // Act & Assert
28      this.mockMvc.perform(post("/api/quizzes")
29              .contentType(MediaType.APPLICATION_JSON)
30              .content(requestBody))
31          .andExpect(status().isBadRequest());
32
33      // Verify nothing saved
34      assertEquals(0, quizRepository.findAll().size());
35  }
```

9 Test Configuration

9.1 Separate Test Database

Create `src/test/resources/application.properties`:

```
spring.datasource.url=jdbc:h2:mem:quizzer-test;
    DB_CLOSE_ON_EXIT=FALSE
spring.jpa.hibernate.ddl-auto=create-drop
```

9.2 Required Dependencies

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>com.jayway.jsonpath</groupId>
    <artifactId>json-path</artifactId>
```

```
    <version>2.8.0</version>
    <scope>test</scope>
</dependency>
```

9.3 Running Tests

```
# Run all tests
./mvnw test

# Run specific test class
./mvnw test -Dtest=QuizRestControllerTest

# Run with verbose output
./mvnw test -X
```

10 Test Naming Conventions

Use descriptive names that explain:

- What is being tested
- Under what conditions
- Expected outcome

Good Examples:

- `getAllQuizzesReturnsEmptyListWhenNoQuizzesExist`
- `getQuizByIdReturnsNotFoundWhenQuizDoesNotExist`
- `createAnswerDoesNotSaveForUnpublishedQuiz`

11 Peer Review

Peer Review assesses each team member's contributions. Personal grades derive from peer reviews and teacher observations.

11.1 Evaluation Criteria

Each team member is graded (0–5) on:

1. **Activity in Team Work**
    - Meeting attendance
    - Active presence and engagement

- Communication outside meetings

2. **Technical Contributions**
   - Amount of working code written
   - Active participation in coding (pair programming)
   - Code quality
3. **Project Management & Documentation**
   - Backlog management
   - Process improvement (retrospectives)
   - Documentation writing

Tip

Peer reviews are **anonymous**—team members don't see each other's reviews. Be honest and constructive!

12 Final Report

The final report answers five questions:

12.1 Question 1: Scrum Events

- List the four Scrum events during a Sprint
- Describe the purpose of each event
- Assess your team's success in fulfilling each event's purpose

12.2 Question 2: Team Assessment

- Identify areas where your team succeeded
- Identify areas needing improvement

12.3 Question 3: Personal Assessment

- Identify your personal strengths
- Identify areas for personal improvement

12.4 Question 4: Advice for New Teams

- Provide three important recommendations
- Justify based on your course experiences

12.5 Question 5: Learning Reflection

- Describe what you learned during the course
- Identify topics for deeper exploration

13 Final Checklist

Sprint 3 Final Checklist

- ☐ All user stories implemented and tested
- ☐ Integration tests for REST endpoints
- ☐ Test configuration with separate database
- ☐ README.md with test instructions
- ☐ LICENSE file added to repository
- ☐ Applications deployed to production
- ☐ Project documentation up to date
- ☐ GitHub Release "Sprint 3" created
- ☐ Peer review submitted
- ☐ Final report submitted as PDF

Figure 3: Sprint 3 Final Checklist

14 Exercises

Sprint 3 Deadline

All work must be pushed to GitHub repository before the final deadline. Peer review and final report are **required to pass**.

14.1 Exercise 1: Retrospective

Conduct a Mad-Sad-Glad retrospective for Sprint 2. Compare with Sprint 1 retrospective to identify recurring issues.

14.2 Exercise 2: Select Scrum Master

Choose a new Scrum Master for Sprint 3 (different from Sprint 2).

14.3 Exercise 3: Close Sprint 2 Issues

Close completed Sprint 2 issues and create Sprint 3 milestone.

14.4 Exercise 4: Create User Story Issues

Create issues for Sprint 3 user stories (Review feature).

14.5 Exercises 5–8: Task Planning

Break down each Sprint 3 user story into technical tasks.

14.6 Exercise 9: Test Configuration

Add test-specific configuration file with separate database.

14.7 Exercise 10: Test All Quizzes Endpoint

Implement tests:

- Returns empty list when no quizzes exist
- Returns list when published quizzes exist
- Does not return unpublished quizzes

14.8 Exercise 11: Test Quiz Questions Endpoint

Implement tests:

- Returns empty list when quiz has no questions
- Returns list with questions and answer options
- Returns error when quiz doesn't exist

14.9 Exercise 12: Test Create Answer Endpoint

Implement tests:

- Saves answer for published quiz
- Rejects answer without answer option
- Rejects answer for non-existing option
- Rejects answer for unpublished quiz

14.10 Exercise 13: Additional Endpoint Tests

Implement tests for at least two additional endpoints of your choice.

14.11 Exercise 14: Update Documentation

Add test running instructions to README.md Developer Guide.

14.12 Exercise 15: Add License

- Choose a license (MIT recommended)
- Create LICENSE file in repository root
- Add license info to README.md

14.13 Exercise 16: Deploy Applications

Deploy final versions of backend and frontend to production.

14.14 Exercise 17: Update Project Documentation

Ensure all documentation is current:

- Project description
- Data model
- Developer guide
- Swagger documentation

14.15 Exercise 18: Create Release

Create GitHub release "Sprint 3" with feature description.

14.16 Exercise 19: Peer Review

Complete peer review for all team members including yourself.

14.17 Exercise 20: Final Report

Write and submit final report answering all five questions as PDF.
