Testing Practices Documentation
Unit Testing & Integration Testing

1. Introduction

Thisdocument presents a concise, end-to-end explanation of functional testing practices implemented in a Java Maven project. It focuses on Unit Testing and Integration Testing, including project structure, code references, execution flow, and best practices.

2. Project Overview & Stack

Functionality: Uppercase conversion,string reversal, length calculation, and combined processing.
Stack: Java, Maven, IntelliJ IDEA, JUnit 5, Surefire, Failsafe, JaCoCo.

3. Project Structure

```
src
▪▪▪ main/java/org/example
▪   ▪▪▪ StringService.java
▪   ▪▪▪ StringProcessor.java
▪▪▪ test/java/org/example
▪   ▪▪▪ StringServiceTest.java
▪▪▪ integration-test/java/org/example
    ▪▪▪ StringIntegrationIT.java
```

4. Unit Testing

Unit testingvalidates individual methods in isolation. Tests are fast, deterministic, and do not rely on other components. 4.1 Production Code –
StringService

```
public class StringService {
    public String toUpper(String input) {
        return input.toUpperCase();
    p u b l i c   S t r i n g   r e v e r s e ( S t r i n g
    i n p u t )   {
        return new StringBuilder(input).reverse().toString();
    }
    p u b l i c   i n t   l e n g t h ( S t r i n g   i n p u t )   {
        return input.length();
    }
}
```

4.2 Unit Test – StringServiceTest

```
class StringServiceTest {
    private final StringService service = new StringService();
    @Test void testToUpper() {

        assertEquals("HELLO", service.toUpper("hello"));

    }
    @Test void testReverse() {
        assertEquals("olleh", service.reverse("hello"));
    }
    @Test void testLength() {
        assertEquals(5, service.length("hello"));
    }
}
```

5. Integration Testing

Integration testingvalidatesinteraction between multiple real components. It ensures correct data flow and end-to-end behavior.

6.1 Integration Logic – StringProcessor

```
publicclass StringProcessor  {
    private final StringService service;
    public StringProcessor(StringService service) {
        this.service = service;
    }
    public String process(String input) {
        String upper = service.toUpper(input);
        String reversed = service.reverse(upper);
        return reversed + ":" + service.length(input);
    }
}
```

6.2 Integration Test – StringIntegrationIT

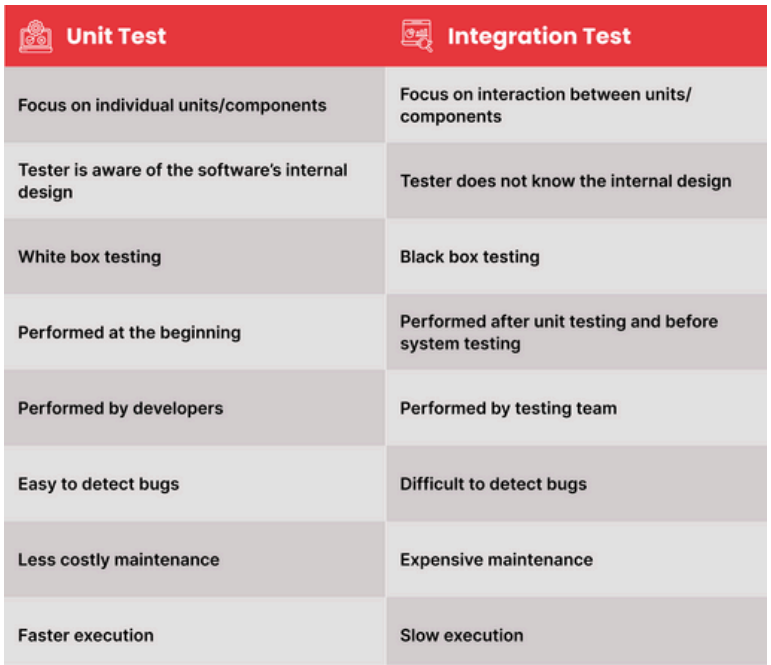
```
classStringIntegrationIT {
    @Test void testFullProcessingFlow() {
        StringProcessor processor =
            new StringProcessor(new StringService());
        assertEquals("OLLEH:5", processor.process("hello"));
    }
}
```

6. Unit vs Integration Testing

```
Unit Testing:
- Single class/method
- Very fast
- No dependencies
Integration Testing:
- Multiple classes
- Slower than unit tests
- Real dependencies
```

7. Code Coverage & Conclusion

Unit andintegrationteststogetherexecute all production code paths, achieving 100% method and line coverage. This layered approach ensures correctness, maintainability, and confidence in future changes.



Unit Test	Integration Test
Focus on individual units/components	Focus on interaction between units/ components
Tester is aware of the software's internal design	Tester does not know the internal design
White box testing	Black box testing
Performed at the beginning	Performed after unit testing and before system testing
Performed by developers	Performed by testing team
Easy to detect bugs	Difficult to detect bugs
Less costly maintenance	Expensive maintenance
Faster execution	Slow execution