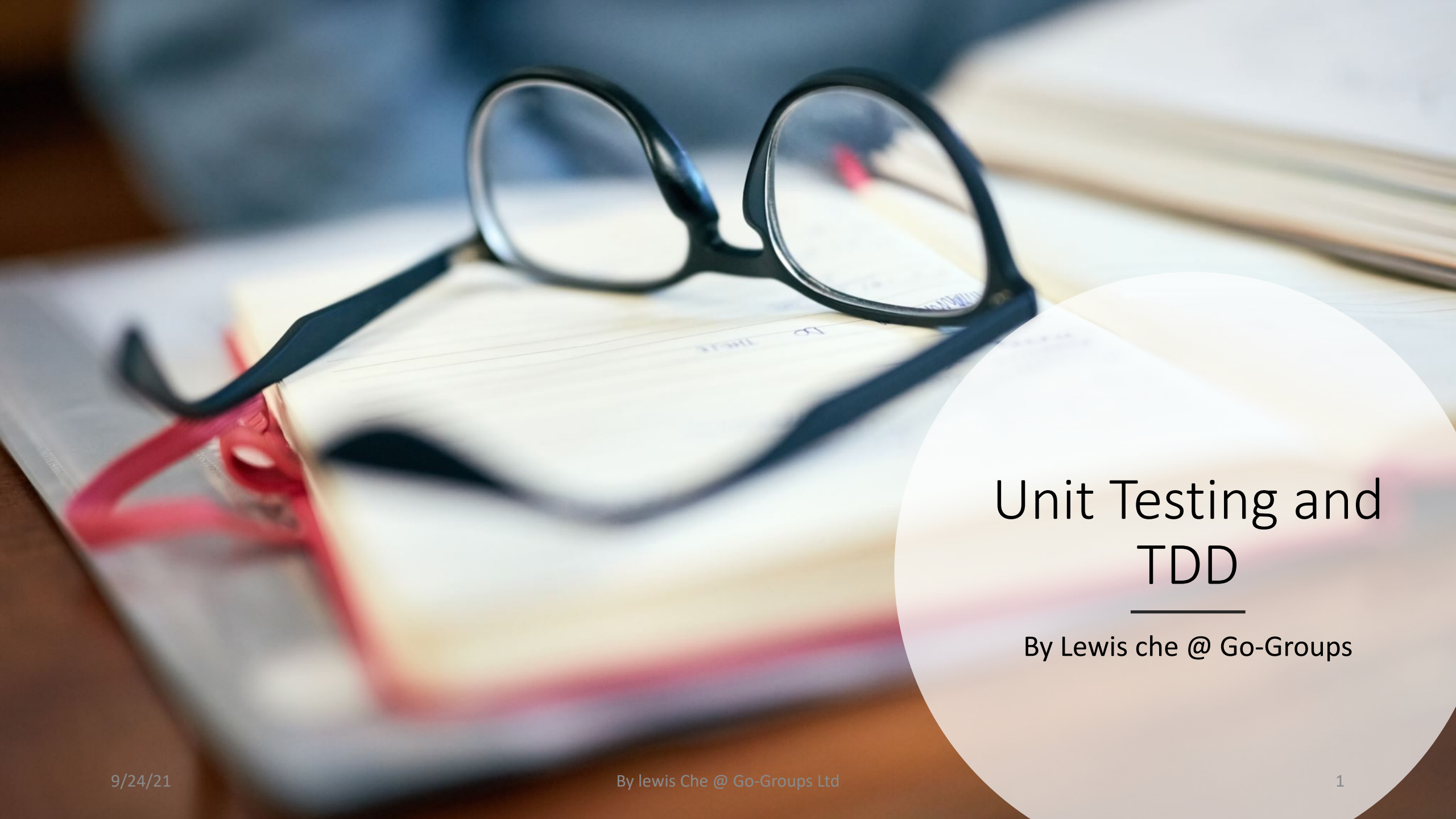
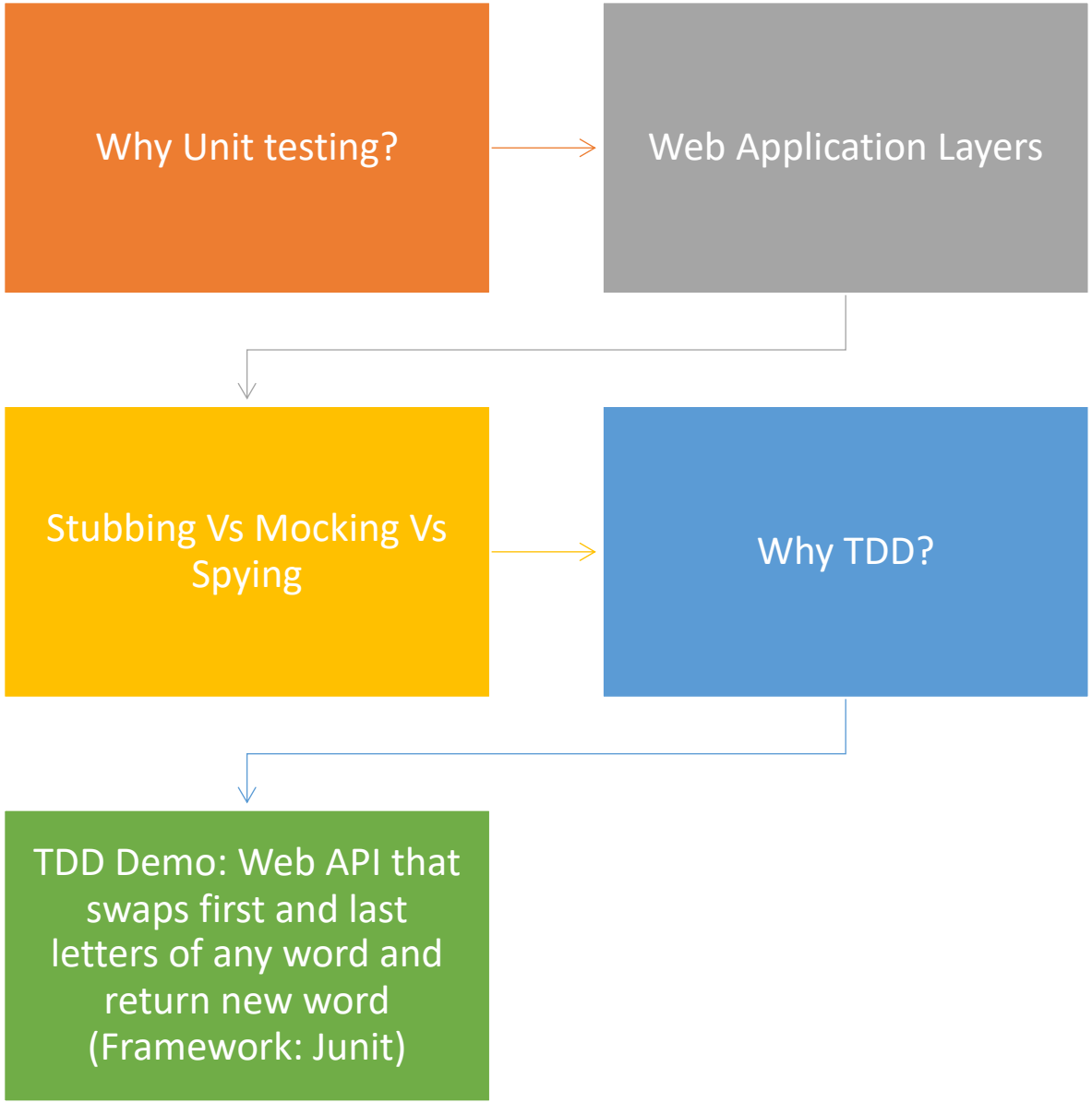


Unit Testing and TDD

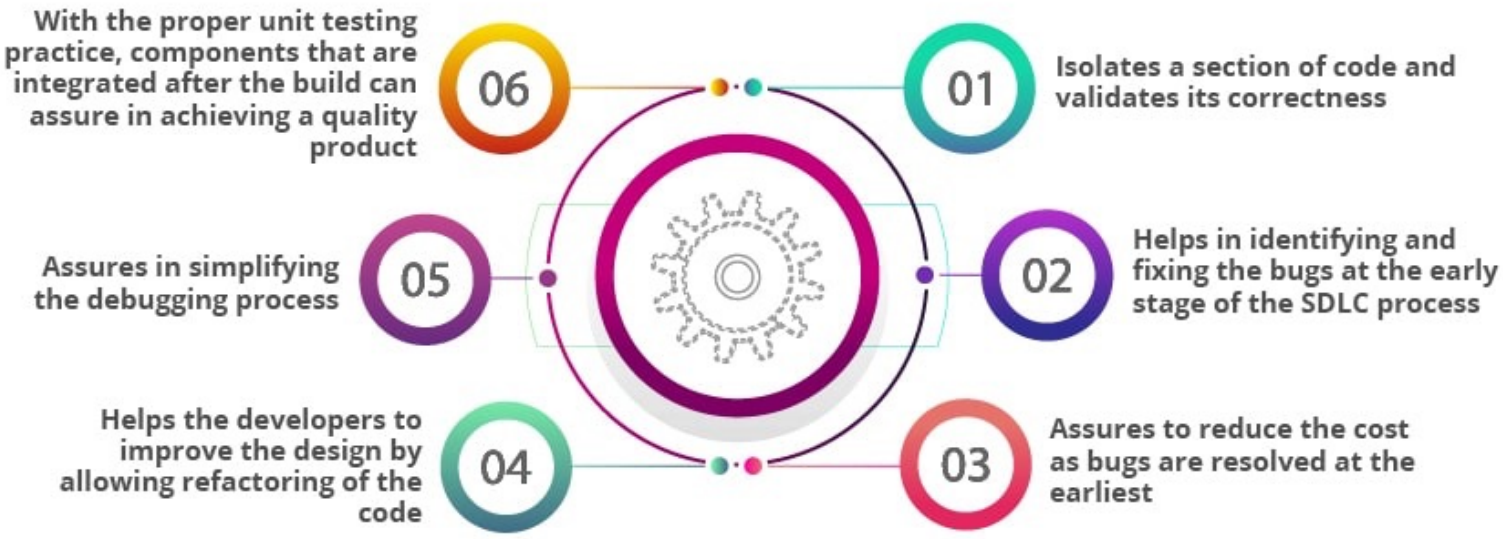
By Lewis che @ Go-Groups

Outline


Why Unit testing?
Web Application Layers
Stubbing Vs Mocking Vs Spying
Why TDD?
TDD Demo: Web API that swaps first and last letters of any word and return new word (Framework: Junit)

Why Unit Testing?

- *What is Unit testing*
  - Unit testing is testing UNITS of a software.
  - Smallest components – functionality
  - Part of the whole



Source: https://www.testingxperts.com/blog/unit-testing

Web Application Layers

A typical web application or microservice has three (3) layers

When testing a layer, you would like to limit your tests to that layer and eliminate any complexities that other layers might introduce.

Controller

- This is the Layer that receives user requests. It is the link between the application/service and the consumers

Business layer

- This where most of the business logic takes place.

Data Layer

- This Layer saves and/or gets data
- New architectures will further split this layer into Data Service and Data Access layer.
- Data Service layer will ensure security and isolation of data from business layer.
- Data Access layer does the actual interaction with database or storage. That is where action CRUD happens.

Stubbing Vs Mocking Vs Spying

Stubbing

- Process of creating dummy class with predefined results
- Cannot be changed in real time – need a new stub if new results expected
- Hard to maintain ☹

Mocking

- Process of creating controllable dummy object.
- You can define the expected results prior to testing – real time changes
- Does not keep track of its passed state. Is not real

Spying

- Process of creating a controllable dummy object that also keeps track of its state
- Part real and part fake!!

Why Test-Driven Development (TDD)?

- TDD is a practice or approach to software development where the test drives the development of the software.
- **Pros**
  - Encourages small steps
  - Gives developer better understanding of their code
  - Early bug identification and fixing
  - Easier maintenance and refactoring
  - Improves team collaboration as tests will indicate undesired changes
  - Enforces good software development principles and good architecture
  - Above all test coverage is done. No need to come back for tests later which might be complex!!
- **Cons**
  - Kinder slows development earlier on
  - It is an art and good TDD comes with time and practice
  - Difficult to apply in Legacy Code.
  - Some unit tests might not be good and developing based on such reduces code quality

BONUS: Pair Programming

Pair Programming is:

- A software development technique
- Two developers working together on one task on same machine
- One developer writes the code – the **Driver**
- The other developer reviews the code as it is being written – the **Navigator**
- The Navigator thinks ahead identifying possible future errors and brings new ideas for improvement
- Driver and Navigator frequently switch roles

Source: https://en.wikipedia.org/wiki/Pair_programming

TDD Demo: Web API that swaps first and last letters of any word and return new word (Framework: Junit)

Thanks!!

Index

Spring Boot projects with versions >= 2.2.0 use JUnit 5 by default.

Description	JUnit 4	JUnit 5

Test Annotation Changes	`@Before`
	`@After`
	`@BeforeClass`
	`@AfterClass`
	`@Ignore`
		`@BeforeEach`
		`@AfterEach`
		`@BeforeAll`
		`@AfterAll`
		`@Disabled`

Use `@ExtendWith` instead of `@RunWith`	`@RunWith(SpringJUnit4ClassRunner.class)` `@RunWith(MockitoJUnitRunner.class)`	`@ExtendWith(SpringExtension.class)` `@ExtendWith(MockitoExtension.class)`
Package changes to `org.junit.jupiter`	`org.junit.Test;` `org.junit.Assert.*;`	`org.junit.jupiter.api.Test;` `org.junit.jupiter.api.Assertions.*;`
`@RunWith` is NOT needed with `@SpringBootTest`, `@WebMvcTest`, `@DataJpaTest`	`@RunWith(SpringRunner.class)` `@SpringBootTest(classes = DemoApplication.class)`	`@SpringBootTest(classes = DemoApplication.class)`