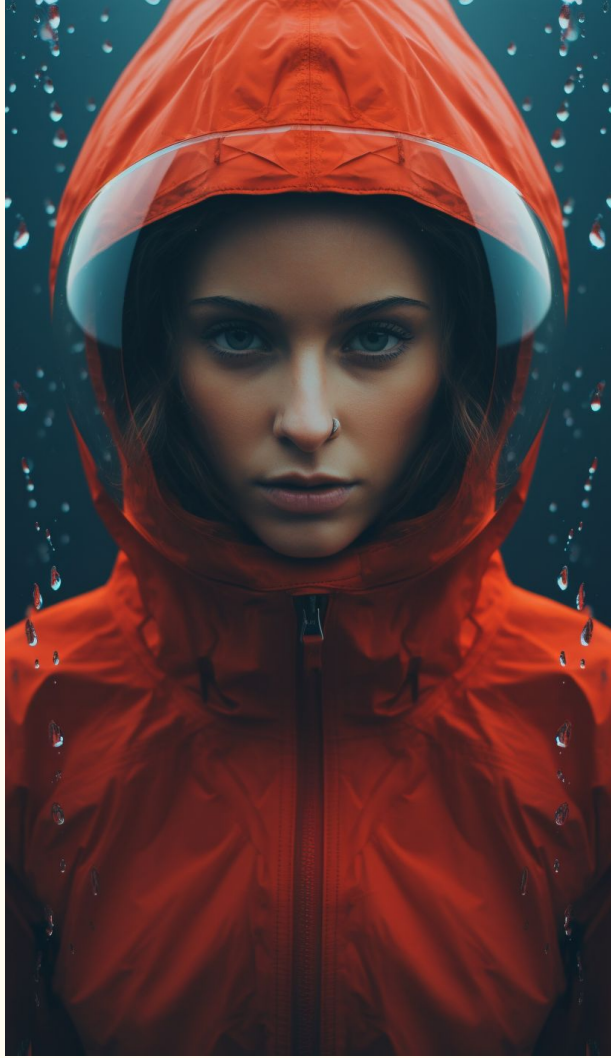
Lesson 5 - Testing and Quality Assurance

—

World of Testing and Quality Assurance

Unit, Integration, and System Testing

- Unit testing checks individual components in isolation.
- Integration testing verifies interactions between components.
- System testing validates the entire software system.



Test-Driven Development and Continuous Integration

- TDD involves writing tests before coding, ensuring functionality meets requirements.
- CI automates code integration and testing with every commit.

TDD using Go (1)

```go
// sum_test.go
package main

import "testing"

func TestSum(t *testing.T) {
    result := Sum(2, 3)
    expected := 5
    if result != expected {
        t.Errorf("Sum(2, 3) = %d; want %d", result, expected)
    }
}
```

Write the Test: In the sum_test.go file, we write a test function TestSum that calls the Sum function with inputs 2 and 3, and checks if the result matches the expected value (5).

TDD using Go (2)

```go
// main.go
package main

func Sum(a, b int) int {
    return a + b
}

func main() {
    // Main code (if needed)
}
```

Implement the Function: In the main.go file, we implement the Sum function that simply adds two integers and returns the result.

TDD using Go (3)

```
go test

PASS
ok      your/package/directory      0.001s
```

Run the Tests: In the terminal, we run the tests using go test. Go's testing framework executes the test function and compares the result to the expected value. If they match, the test passes; otherwise, it fails.

TDD using Python (1)

```python
# test_factorial.py
import unittest
from factorial import factorial

class TestFactorial(unittest.TestCase):

    def test_factorial_of_0(self):
        self.assertEqual(factorial(0), 1)

    def test_factorial_of_1(self):
        self.assertEqual(factorial(1), 1)

    def test_factorial_of_positive_number(self):
        self.assertEqual(factorial(5), 120)

if __name__ == '__main__':
    unittest.main()
```

Write the Test: In the test_factorial.py file, we write test cases for the factorial function. We define three test methods to check the factorial of 0, 1, and a positive number.

TDD using Python (2)

```python
# factorial.py
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

Implement the Function: In the factorial.py file, we implement the factorial function recursively according to the test cases.

TDD using Python (2)

```
python test_factorial.py

...
----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
```

Run the Tests: In the terminal, we run the tests using the unittest framework. The framework executes the test methods and compares the results to the expected outcomes.

Bug Tracking and Debugging Techniques

- Bug tracking tools log and manage reported issues.
- Debugging involves identifying and fixing issues in code.

How to Track Bug

- Reproduce the bug: Try to replicate the issue consistently. Understand the steps that lead to the problem.
- Log bug details: Write description, steps to produce, expected behavior, actual behavior, error messages and stack traces.
- Use a bug tracking system: Manage and track bug using tools (JIRA, GitHub Issues, etc.).
- Assign severity and priority: Assess the impact and urgency.
- Assign to developer: Ask relevant developers to address the issue.

Debugging Best Practices

- Isolate the problem.
- Check inputs and assumptions.
- Use logging.
- Inspect error messages.
- Break down the problem.
- Use debugging tools.
- Comment out code.
- Pair programming.
- Unit test.
- Version control.
- Google (Bing) and Stack Overflow.

Performance Testing and Optimization

- Performance testing assesses software speed, responsiveness, and scalability.
- Optimization involves improving code efficiency and minimizing resource usage.

Case Study: Tackling Performance Issues

A team is working on an online e-commerce platform. Users have reported slow loading times for product listings and checkout pages. The team's goal is to improve the application's performance to enhance user experience.

Issue Identification

1. **User Feedback**: Users complain about slow loading times, leading to abandoned carts and frustration.
2. **Performance Metrics**: Load time and page speed tests confirm that pages take longer to load than desired

Investigation

1. **Analyze Code and Database Queries**: The team reviews the codebase and identifies resource-intensive operations and database queries that slow down page rendering.
2. **Identify Bottlenecks**: Profiling tools show that multiple database queries are executed sequentially during page load, causing delays.

Performance Enhancements

1. **Caching Mechanism:** The team implements caching for frequently accessed data, such as product listings and user profiles, to reduce database queries.
2. **Optimized Database Queries:** Database queries are optimized using indexes, joins, and proper query design to minimize execution time.
3. **Lazy Loading:** Instead of loading all content at once, the team implements lazy loading for images and additional content, improving initial page load times.

Testing and Validation

1. **Load Testing**: Load testing tools simulate high user traffic to evaluate how the changes impact performance under heavy loads.
2. **User Acceptance Testing**: A subset of users participates in testing the improved application to provide feedback and confirm improved performance.

Outcome

1. **Improved Performance**: After implementing caching, optimizing queries, and incorporating lazy loading, the application's page load times are significantly reduced.
2. **Positive User Feedback**: Users report faster load times and improved user experience, leading to increased engagement and fewer abandoned carts.

Continuous Monitoring

1. **Performance Monitoring**: The team continues to monitor performance metrics, ensuring that improvements are sustained and identifying any new bottlenecks.
2. **Iterative Refinement**: As the application evolves, the team regularly reviews and refines the performance enhancements to adapt to changing usage patterns.

Continuous Improvement and Testing Metrics

- Continuous improvement involves learning from testing results and enhancing processes.
- Testing metrics track testing progress and effectiveness.

Ensuring Quality Across the Lifecycle

- Quality assurance spans the entire software development lifecycle.
- Ensuring quality requires collaboration, vigilance, and a proactive approach.

Conclusion

- Quality assurance is a multifaceted discipline critical for software success.
- Combining rigorous testing, optimization, and learning drives quality.

Question?