Test Driven Development & Go

Conf 42, Golang - 2023

whoami

Mohammad Quanit

- Product Engineering Manager - timegram.io
- AWS Community builder
- Tech Content Writer & public speaker
- Love doing gym, travelling, sports

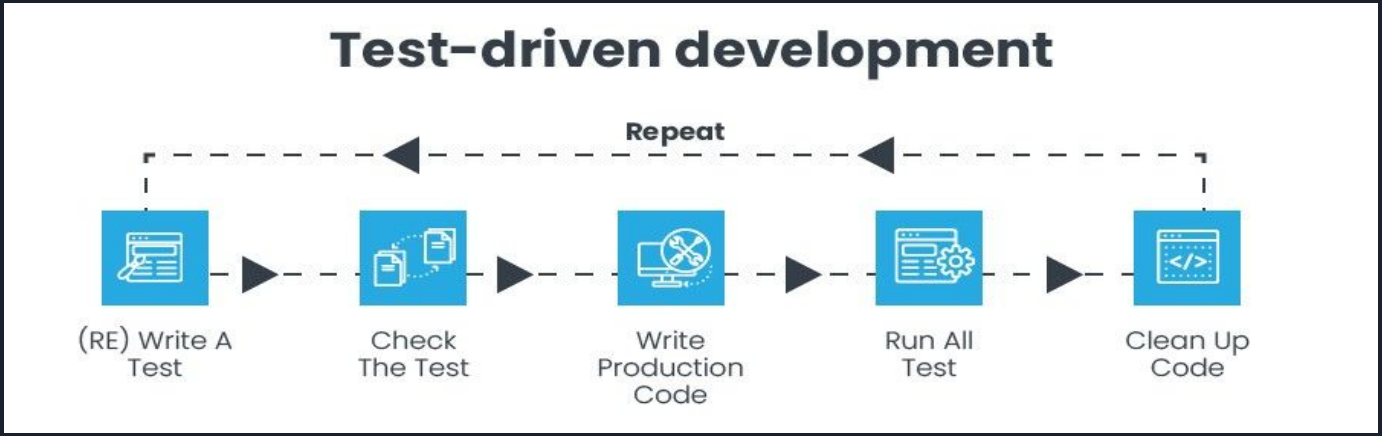
Agenda

- Test Driven Development - TDD
- Go Testing Package
- HTTP Testing (Rest API)
- Table Driven Testing
- Other Open Source Testing Frameworks
- Best Practices

Test Driven Development

- TDD is a software development process that involves repeatedly writing test cases first and then write actual code.
- It forces developers to think in terms of implementers or users.
- By writing tests first, you can catch errors early in the development process, and ensure that your code is easy to test, maintain, and refactor.
- TDD helps engineers to write better code, reduce time on debugging, and leads to more predictable and reliable software.
- TDD is not just a part of some merely testing mechanism but a method of designing software (Mental model).

TDD Stages


Test-driven development
Repeat
(RE) Write A Test
Check The Test
Write Production Code
Run All Test
Clean Up Code

Motivations for TDD

- Shortens the programming feedback loop
- Encourage engineers to write modular, testable & maintainable code
- Helps catch errors early, reduce debugging time
- Reduces the cost of change
- Boosts confidence with sense of continuous reliability and success
- If you don’t have tests, how do you know your code is doing the right thing?

Go Testing Package

Conf 42, Golang - 2023

Go Testing

Go comes with a built-in command line testing tool that automates the process of running tests called “go test”

Test functions with a specific signature, must start with Test that takes a pointer ***Testing.t**

Test Coverage tool can generate a coverage report, which shows how much of your code is covered by tests

Go testing package include supports Benchmarks, that can be used to measure performance of your code

Go testing package provides supports for several flags that can be used to control the behaviour of your tests

Go Testing Package

Tests are written on files ending with “_test.go” & every test function starts with **Test** keyword which takes a testing parameter

HTTP Testing

HTTP Testing

Testing an HTTP server in Go involves sending HTTP requests to the server and verifying the responses it returns.

Table Driven Testing

Conf 42, Golang - 2023

Table Driven Testing

Table-driven tests allow you to test a function with multiple inputs and expected outputs.

Testing Frameworks by GO Community

- Gomega - Matcher/Assertion lib (https://github.com/onsi/gomega)
- GoCheck - Featured rich testing lib (https://github.com/go-check/check)
- Testify - Toolkit for mocks, assertions (https://github.com/stretchr/testify)
- GoMock - A dedicated mocking framework (https://github.com/golang/mock)
- Ginkgo - A BDD testing framework for expressive specs (https://github.com/onsi/ginkgo)

Best Practices to Follow

- Write test case before actual code

Best Practices to Follow

- Write test case before actual code
- Write small, focused tests

Best Practices to Follow

- Write test case before actual code
- Write small, focused tests
- Use **go test** command to test case along with **-v** flag for verbose logs

Best Practices to Follow

- Write test case before actual code
- Write small, focused tests
- Use **go test** command to test case along with **-v** flag for verbose logs
- Use Mock dependencies to simulate actual behavior of the feature

Best Practices to Follow

- Write test case before actual code
- Write small, focused tests
- Use **go test** command to test case along with **-v** flag for verbose logs
- Use Mock dependencies to simulate actual behavior of the feature
- Utilize code coverage tools that like **go test -cover**

Best Practices to Follow

- Write test case before actual code
- Write small, focused tests
- Use **go test** command to test case along with **-v** flag for verbose logs
- Use Mock dependencies to simulate actual behavior of the feature
- Utilize code coverage tools that like **go test -cover**
- Automate and refactor your test cases e.g using CI tools

Best Practices to Follow

- Write test case before actual code
- Write small, focused tests
- Use **go test** command to test case along with **-v** flag for verbose logs
- Use Mock dependencies to simulate actual behavior of the feature
- Utilize code coverage tools that like **go test -cover**
- Automate and refactor your test cases e.g using CI tools
- Always keeps your test cases up to date

Thank You

Conf 42, Golang Team

Follow me on:

Twitter
Github
LinkedIn