Testing in Go 101

About Me

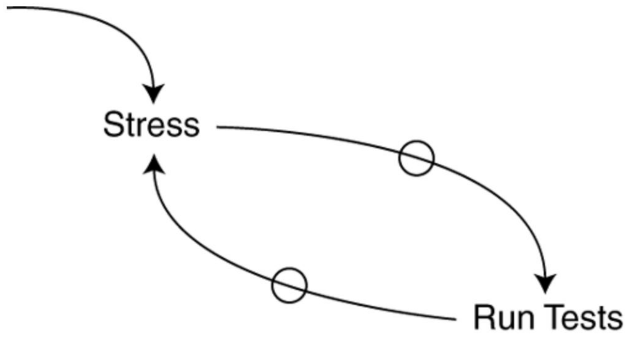
Francisco Daines (Dino)

- Principal Sw. Engineer at Walmart Chile
- Experience in Java, C, Javascript
- Using Go since 2020
- Maintainer of Arch-go

fdaines
fdaines
fdaines
fdaines@gmail.com

1.
A brief introduction to Application Testing

Why is testing important?



The "no time for testing" death spiral
Kent Beck, Test-Driven Development: By Example

Infinite loop

More Stress => Less testing

Less testing => More errors (then stress)

Breaking the loop

- As testing reduces uncertainty then, if the stress is increased, then improve tests to reduce uncertainty and stress.

“

Rather than apply minutes of suspect reasoning, we can just ask the computer by making the change and running the tests.

Kent Beck, Test-Driven Development: By Example

Benefits of Application Testing

Reduces Bugs

If we continuously test different use cases for our code, then we can find bugs and fix them before publishing our software artifacts

Improves Security

Testing different use cases reduces vulnerabilities of our code

Increases Software Quality

Since we can catch some bugs as part of our development cycle, our software products will be of higher quality.

The ones that really matters...

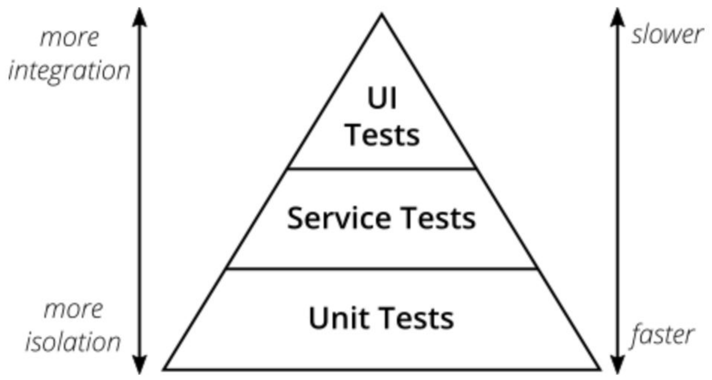
Saves Money

Increasing software quality reduces production errors and debugging time to find their causes, so that we can focus our time in adding value to business.

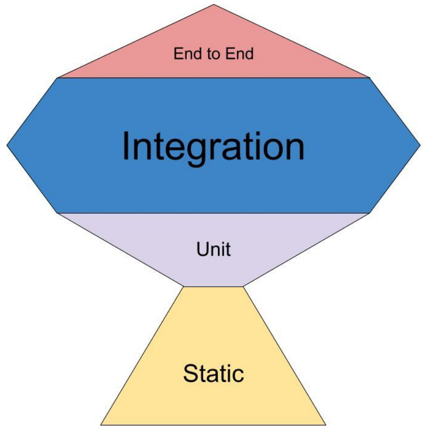
Improves Customer Satisfaction

Increasing deployment frequency and releasing higher quality products should improve our customers satisfaction and loyalty.

Different testing approaches



The testing pyramid
Mike Cohn



The testing trophy
Kent C. Dodds

There is no rule of thumb to select the “right” testing approach

This talk is about unit testing, unless you select to follow a testing pyramid or trophy

2.
Test in Go - First steps

First Steps

Go testing tool

Is a standard tool to automates the testing for desired packages.

```
> go test ./...
```

The example above will run tests for all the nested packages

For more information:

```
> go help test
```

Testing Package

Provides features to create unit tests, benchmarks and fuzzy tests.

In this talk, we'll focus only on unit tests.

For more information:

https://pkg.go.dev/testing

My very first test in Go

How can I test multiple cases?

Most of the time we need to execute our functions using different input values, and checking the returned values with the expected results.

A (wrong) Approach

Copy/Paste the test, changing the input and expected values.

Code duplication is an issue we never want to deal with.

```go
func TestAddNumbersCase1(t *testing.T) {
  expected := 101
  got := AddNumbers(1,100)
  if got != expected {
     t.Errorf("Expected: %v, got: %v", expected, got)
  }
}

func TestAddNumbersCase2(t *testing.T) {
  expected := 0
  got := AddNumbers()
  if got != expected {
     t.Errorf("Expected: %v, got: %v", expected, got)
  }
}
```

Table Driven Tests

The main idea is to define a table with input and expected results, then execute the same test for each tuple in the table, avoiding code duplication.

Input	Expected
1, 2, 3	6
1, 100	101
201, 403	604
111, 222, 333, 444	1110
(empty)	0

```go
var addNumbersTestCases = []struct {
  numbers  []int
  expected int
}{
  {[]int{1,2,3}, 6},
  {[]int{1,100}, 101},
  {[]int{201,403}, 604},
  {[]int{111,222,333,444}, 1110},
  {[]int{}, 0},
}

func TestAddNumbersTableDriven(t *testing.T) {
  for idx, test := range addNumbersTestCases {
     t.Run(fmt.Sprintf("Scenario %d", idx), func(t *testing.T) {
        got := AddNumbers(test.numbers...)
        if got != test.expected {
           t.Errorf("TestCase(%d). Expected: %v, got: %v", idx, test.expected, got)
        }
     })
  }
}
```

Table Driven Tests

Code Examples

Examples are code snippets that are displayed as package documentation and also are verified by running them as tests.

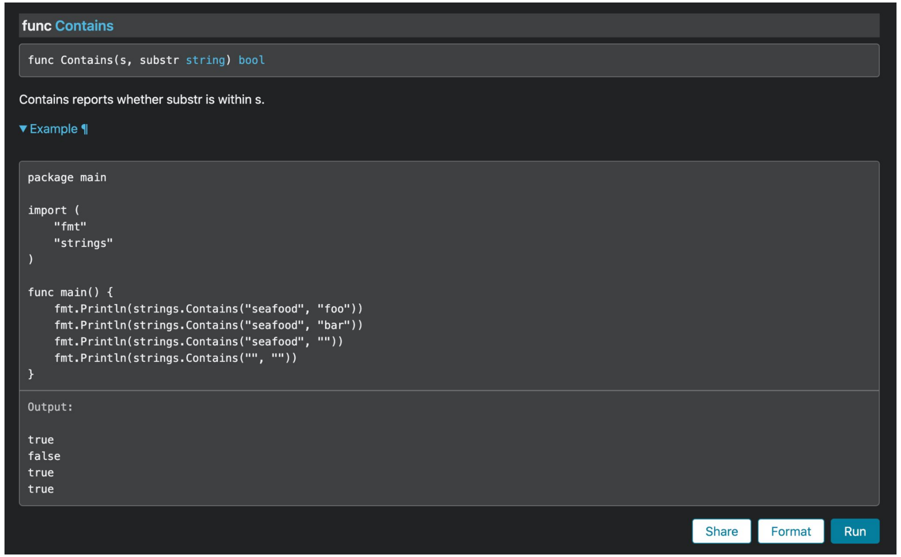
They can also be run by a user visiting the godoc web page for de package.

```go
package section2

import (
  "fmt"
)

func ExampleAddNumbers() {
  fmt.Println(AddNumbers(4, 6))
  // Output: 10
}
```

```go
// Naming convention for Example Functions
func Example() { ... }
func ExampleFunction() { ... }
func ExampleType() { ... }
func ExampleType_Method() { ... }
```

```go
// Example for Compare function (package: strings)
func ExampleCompare()
// Example for Writer type (package: bufio)
func ExampleWriter() { ... }
// Example for lines method in Scanner type (package: bufio)
func ExampleScanner_lines() { ... }
```

Code Examples



func Contains

```
func Contains(s, substr string) bool
```

Contains reports whether substr is within s.

▼Example ¶

```
package main

import (
    "fmt"
    "strings"
)

func main() {
    fmt.Println(strings.Contains("seafood", "foo"))
    fmt.Println(strings.Contains("seafood", "bar"))
    fmt.Println(strings.Contains("seafood", ""))
    fmt.Println(strings.Contains("", ""))
}
```

```
Output:

true
false
true
true
```

Code Examples will be published to godoc web page for the package.

This example shows the ***strings*** package documentation at https://pkg.go.dev/strings#Contains

3.
Assertions

Assertions

Assertions are sentences that must be true, in any other case our tests will fail.

There are a lot of assertion libraries, in this case we will use ***testify package***.

```go
package section3

import (
  "github.com/stretchr/testify/assert"
  "testing"
)

func TestAssertions(t *testing.T) {
  var zeroValue int
  assert.Zero(t, zeroValue, "Expected to have zero value")

  slice := []string{"foo", "bar", "test", "example"}
  assert.NotNil(t, slice, "Expected to have a value")

  assert.Contains(t, slice, "bar", "Expected to contain 'bar'")

  myValue := 10
  assert.Equal(t, 10, myValue, "Expected value is 10")

  list1 := []int{1,2,3,4,5}
  list2 := []int{3,2,5,1,4}
  assert.ElementsMatch(t, list1, list2, "List contains different elements")
}
```

Interesting Assertions

Equal, NotEqual	Zero, NotZero
Nil, NotNil	ElementsMatch
Contains, NotContains	Filesystem assertions
Error, NoError	HTTP assertions
ErrorContains	JSONEq, YAMLEq

Assertions

What happens if an assertion fails

```go
func TestAssertions(t *testing.T) {
  slice := []string{"foo", "bar", "test", "example"}
  assert.Contains(t, slice, "barX", "Expected to contain 'barX'")
}
```

More readable output

```
> go test -run ^TestAssertionsFailing$ ./... -v
=== RUN   TestAssertionsFailing
    assertions_failing_test.go:10:
                Error Trace:    assertions_failing_test.go:10
                Error:          []string{"foo", "bar", "test", "example"} does not contain "barX"
                Test:           TestAssertionsFailing
                Messages:       Expected to contains 'barX'
--- FAIL: TestAssertionsFailing (0.00s)
FAIL
FAIL    github.com/fdaines/testing-101/section3 0.178s
FAIL
```

Assertions

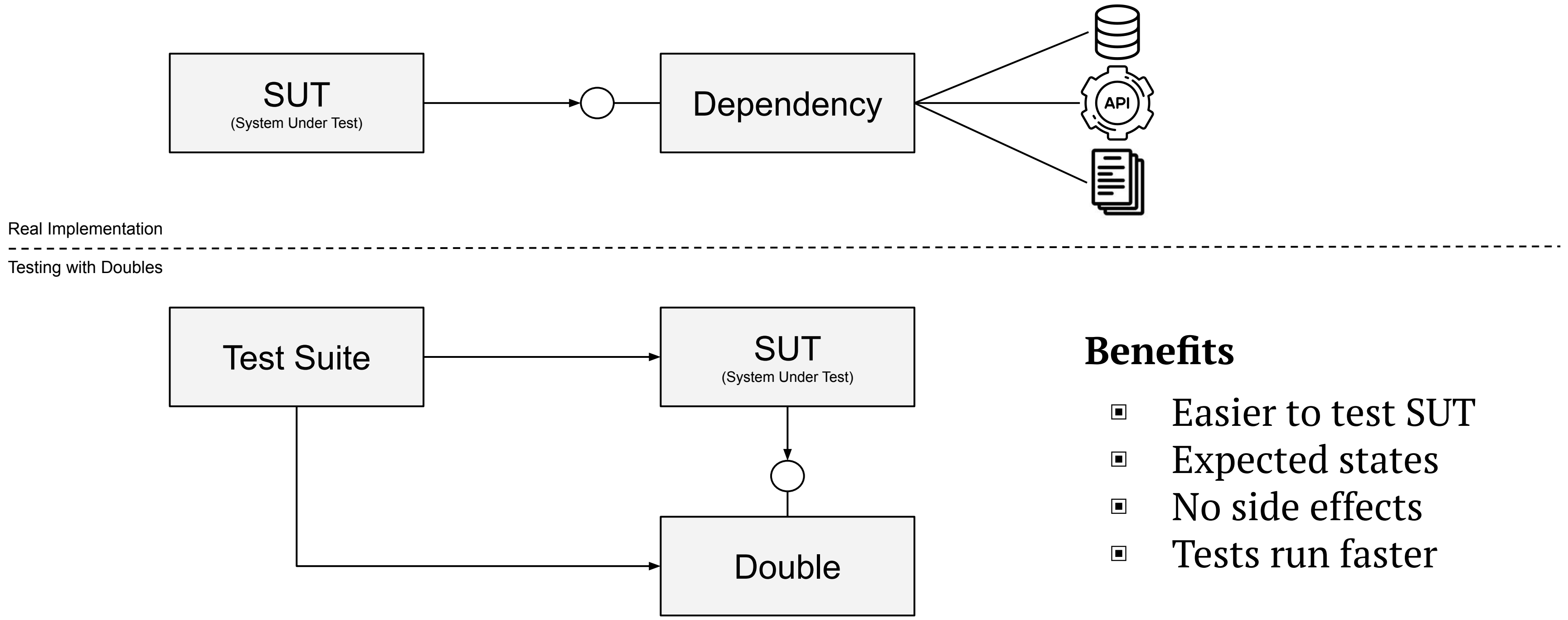
Assertions Libraries

- https://github.com/stretchr/testify
- https://github.com/onsi/gomega
- https://github.com/smartystreets/assertions
- https://github.com/pellared/fluentassert

- For more options: https://pkg.go.dev/search?q=assert

4.
Test Doubles: Mocks and Stubs

Context



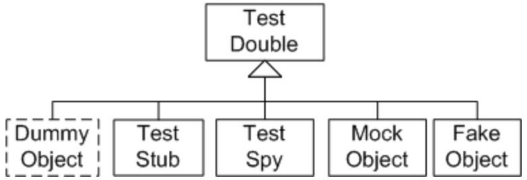
Benefits

- Easier to test SUT
- Expected states
- No side effects
- Tests run faster

Test Doubles

Test Doubles replaces some dependencies to ease testing.

Test Doubles merely has to provide the same API as the real dependency so that our system thinks it is the real one.



Sketch Types Of Test Doubles embedded from Types Of Test Doubles
Gerard Meszaros - xUnit Patterns

Variation	Description
Dummy	Objects that are passed around to functions but never actually used.
Fake	It has a real implementation of dependency API, but usually take some shortcut which makes them not suitable for production (an InMemoryTestDatabase is a good example).
Stub	Provide canned answers to calls made during the test, usually not responding at all to anything outside what's programmed in for the test.
Spies	Are stubs that also record some information based on how they were called. One form of this might be a logging service that records how many messages were logged.
Mock	They're similar to stubs but including expectations which form a specification of the calls they are expected to receive. Usually they throw an exception if either receive a call they don't expect or doesn't receive all the calls they were expecting.

Using Stubs

The easiest way to stub dependency components is to implement an object that complies with the required interface. This is easy if we're following clean code principles, where we have an interface for every infrastructure component.

```go
type ConcreteDependency struct {
}
func (m ConcreteDependency) DoStuff() {
  // Do some stuff (call external service)
}

type MyBusinessType struct {
  innerDependency ConcreteDependency
}
func (m MyBusinessType) SomeBusinessLogicMethod() {
  m.innerDependency.DoStuff()
}
```

Poor design, because *MyBusinessType* depends directly on a concrete type.

Hard to test if concrete type calls external services or triggers side effects

```go
type MyInterface interface {
  DoStuff()
}

type MyBusinessType struct {
  innerDependency MyInterface
}
func NewBusinessType(dependency MyInterface) MyBusinessType{
  return MyBusinessType{dependency}
}
func (m MyBusinessType) SomeBusinessLogicMethod() {
  m.innerDependency.DoStuff()
}
```

We can create many implementations of *MyInterface* to test different behaviors.

Using Stubs

Using Stubs

The ***gostub*** packages allows us to stub static variables and functions, and rolling back to their original values/behavior after the test is finished.

```go
package section4

var configFile = "config.json"
var myFunction = MyOriginalBehavior

func MyOriginalBehavior() string {
  message := "Hello world"
  // message = callToExternalService()
  return message
}
```

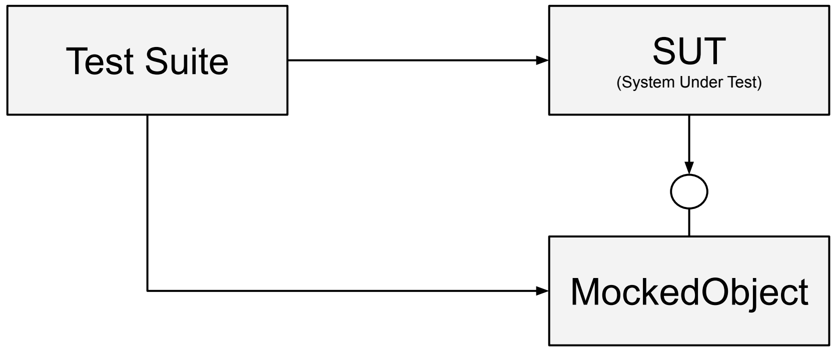
https://github.com/prashantv/gostub

```go
package section4

import (
  "testing"
  "github.com/prashantv/gostub"
  "github.com/stretchr/testify/assert"
)

func TestForUnexistentConfigFile(t *testing.T) {
  stub := gostub.Stub(&configFile, "/unexistent_path")
  defer stub.Reset()

  content, err := readConfiguration()
  assert.Error(t, err, "Expect to receive an error")
}

func TestWithFixedReturnValue(t *testing.T) {
  stub := gostub.Stub(&myFunction, func () string {
     return "Bye World"
  })
  defer stub.Reset()

  message := myFunction()
  assert.Equal(t, "Bye World", message, "Unexpected message")
}
```

Using Mocks

A mock is like a stub but the test will also verify that the object under test calls the mock as expected.

Part of the test is verifying that the mock was used correctly.



Links a set of Calling parameters with a specific behavior.
The test would fail when:

- One of the configured behavior was not called
- The test calls the functions with a non configured set of parameters

Using Mocks with Testify

Testify offers some useful features

- `mockedObject.On("MyMethod", arg1, arg2).Return(returnValues…).Once()`
- `mockedObject.On("MyMethod", arg1, arg2).Return(returnValues…).Twice()`
- `mockedObject.On("MyMethod", arg1, arg2).Return(returnValues…).Times(5)`
- `mockedObject.On("MyMethod", arg1, arg2).After(time.Second)`

Example: The first time the method is called with certain arguments, return "Hello", all the next calls will return "Bye".

```
var mockedService = new(TestExampleImplementation)
mockedService.On("MyMethod", 1, 2).Return("Hello").Once()
mockedService.On("MyMethod", 1, 2).Return("Bye")
```

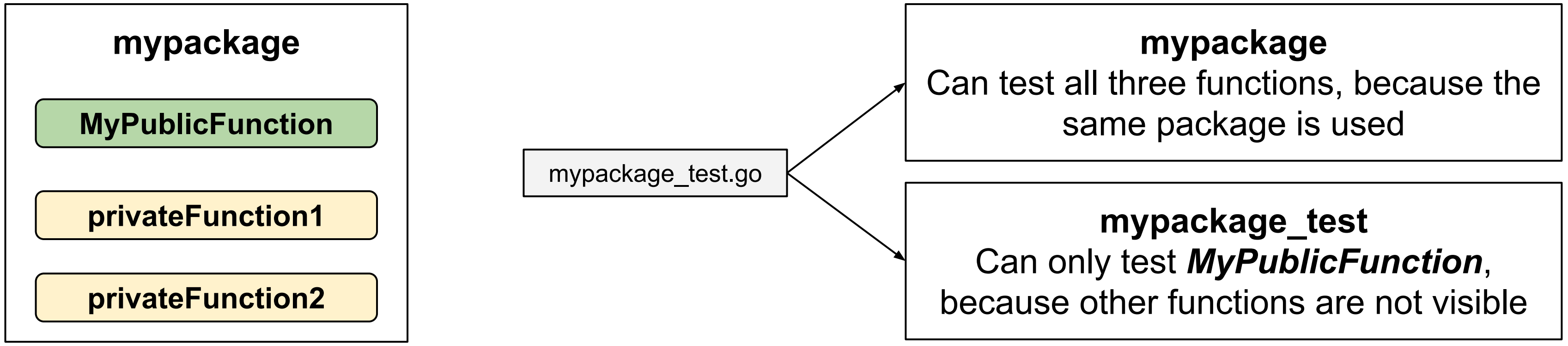
Using mocks with Testify

5.
Where to put tests?

Where should I put my Tests?

By convention, Go testing files should be always located in the same folder/package where the code they are testing resides.

The Go compiler will exclude all the testing files when building our application, so we don’t need to worry about having testing code into deployed artifacts.


mypackage
MyPublicFunction
privateFunction1
privateFunction2
mypackage_test.go
mypackage
Can test all three functions, because the same package is used
mypackage_test
Can only test *MyPublicFunction*, because other functions are not visible

Where should I put my Tests?

5.
Coverage Reports

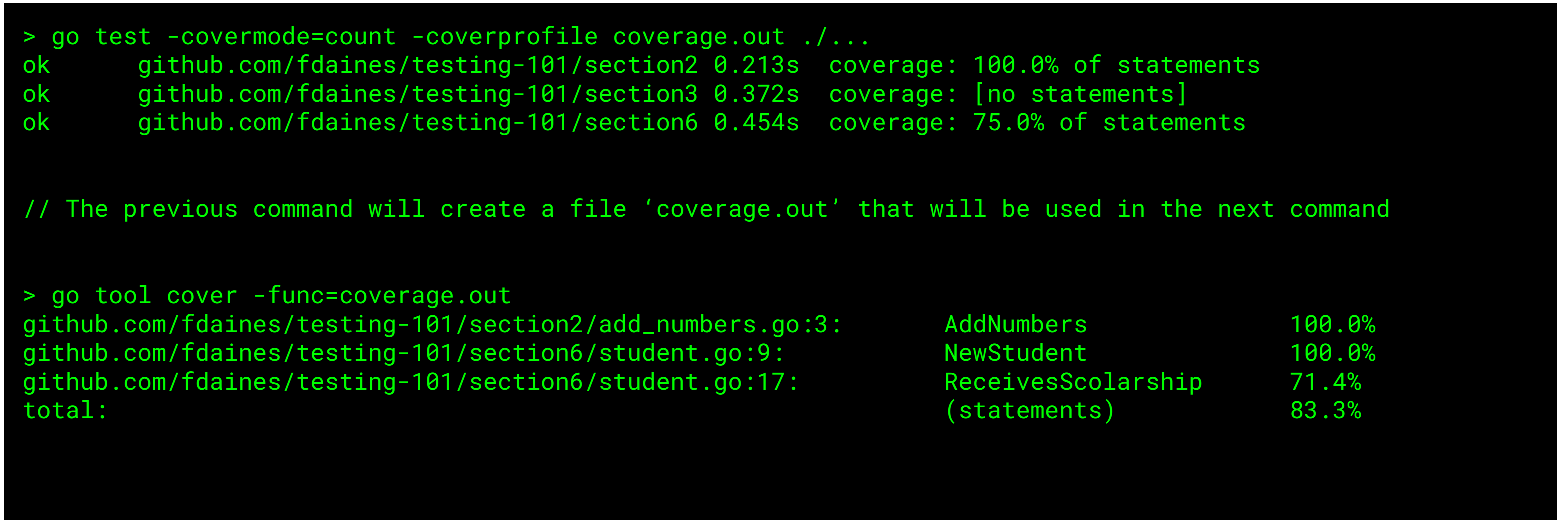
Coverage Reports

Using go testing tool

```
> go test ./... -cover
ok      github.com/fdaines/testing-101/section2 0.172s  coverage: 100.0% of statements
ok      github.com/fdaines/testing-101/section3 0.229s  coverage: [no statements]
ok      github.com/fdaines/testing-101/section6 0.275s  coverage: 75.0% of statements
```

Coverage Reports

Using go tool cover

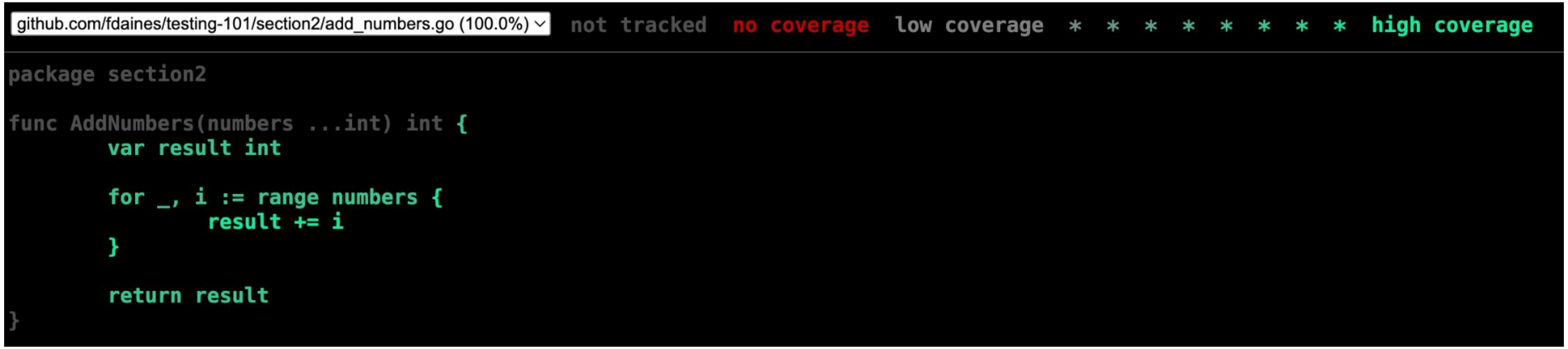


```
> go test -covermode=count -coverprofile coverage.out ./...
ok      github.com/fdaines/testing-101/section2 0.213s  coverage: 100.0% of statements
ok      github.com/fdaines/testing-101/section3 0.372s  coverage: [no statements]
ok      github.com/fdaines/testing-101/section6 0.454s  coverage: 75.0% of statements


// The previous command will create a file ‘coverage.out’ that will be used in the next command


> go tool cover -func=coverage.out
github.com/fdaines/testing-101/section2/add_numbers.go:3:       AddNumbers              100.0%
github.com/fdaines/testing-101/section6/student.go:9:           NewStudent              100.0%
github.com/fdaines/testing-101/section6/student.go:17:          ReceivesScolarship      71.4%
total:                                                          (statements)            83.3%
```

Coverage Reports

Using go tool cover to generate an HTML report

```
// Using the same ‘coverage.out’ file generated with go test.
> go tool cover -html=coverage.out
```

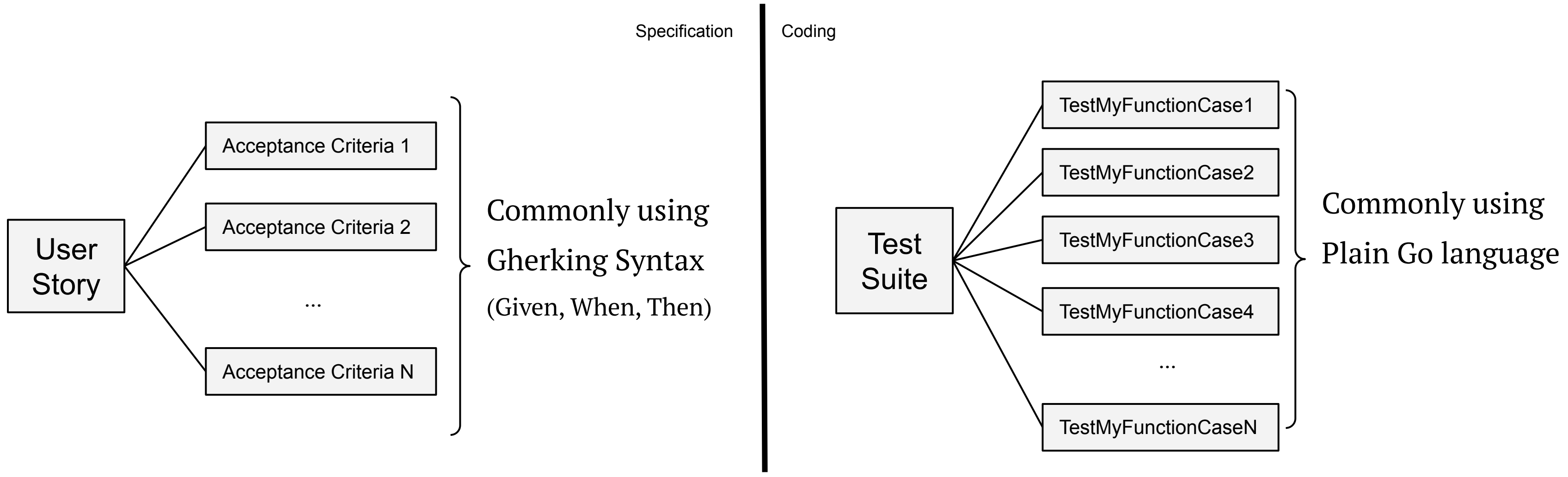


github.com/fdaines/testing-101/section2/add_numbers.go (100.0%) not tracked no coverage low coverage * * * * * * * * high coverage

```
package section2

func AddNumbers(numbers ...int) int {
	var result int

	for _, i := range numbers {
		result += i
	}

	return result
}
```

6.
BDD: Behavior Driven Development

Motivation



Behavior Driven Development uses a simple DSL to convert structured human language (like the one used in acceptance criteria) into executable tests.

BDD-style testing frameworks

- https://github.com/onsi/ginkgo
- https://github.com/smartystreets/goconvey
- https://github.com/franela/goblin
- https://github.com/azer/mao
- https://github.com/pranavraja/zen

BDD in Go using Ginkgo and Gomega

User Story

As a Dean
I want to know if a student is elegible for a scholarship
So that I'll have enough information to prepare the scholarships budget

Acceptance Criteria

Given a Student
When his/her program is medicine
  And he/she has a GPA >= 80
Then he/she receives a scholarship

Given a Student
When his/her program is engineering
  And he/she has a GPA >= 75
Then he/she receives a scholarship

Given a Student
When his/her program is literature
  And he/she has a GPA >= 60
Then he/she receives a scholarship

Every other case: he/she doesn't receive scholarship

⇨

Ginkgo + Gomega

Ginkgo offers a BDD style testing framework

- Functions like ***Describe***, ***Context***, ***When***, ***It*** and others allow us to create tests based on acceptance criteria with little effort.

Gomega is a matcher/assertion library with an API that help us to represent expectations from acceptance criteria.

Example using Ginkgo and Gomega

```go
package section6

import (
  . "github.com/onsi/ginkgo/v2"
  . "github.com/onsi/gomega"
  "testing"
)

func TestSection6(t *testing.T) {
  // connects Ginkgo to Gomega. When a matcher fails the Fail handler (from Ginkgo) is called.
  RegisterFailHandler(Fail)
  // is the entry point for the Ginkgo spec runner.
  RunSpecs(t, "Testing package section6")
}

var _ = Describe("Checking scholarship elegibility", func() {
  When("the program is medicine", func() {
     Context("and the GPA >= 80", func() {
        student := NewStudent(18, "medicine", 80)
        It("receives the scholarship", func() {
           Expect(student.ReceivesScolarship()).To(BeTrue())
        })
     })
     Context("and the GPA < 80", func() {
        student := NewStudent(18, "medicine", 79)
        It("doesn't receive the scholarship", func() {
           Expect(student.ReceivesScolarship()).To(BeFalse())
        })
     })
  })
})
```

References

Test-Driven Development: By Example, Kent Beck, 2002.

http://xunitpatterns.com/

https://github.com/stretchr/testify

https://github.com/onsi/ginkgo

https://github.com/onsi/gomega

thanks!

Any questions?

You can find me at

@fdaines

fdaines@gmail.com