Quality Tests in Go

Mat Ryer
Gotham Go - New York City
November 18th 2016

Mat Ryer

⌨️ Coding Go since before v1

🔗 matryer.com

📚 Go Programming Blueprints: Second Edition

👷 graymeta.com - 100% Go backend

🖌️ Not Banksy

Wayne Manor

Wollaton Hall, Nottingham, England

Purpose of this talk

- ☞ Explain what testing is and why **the best projects in the world do it**
- ☞ Convince you to write tests **first**
- ☞ Look at some tips for writing quality tests

If you don't write test code,
you're either very brave or

What is a test?

```
// in batify.go
func Batify(item string) string {
    return "Bat " + item
}

// in batify_test.go
func TestBatify(t *testing.T) {
    actual := Batify("Car")
    if actual != "Bat Car" {
        t.Errorf("expected Bat Car but got %s", actual)
    }
}
```

Why write tests?

- ☞ Get computers to check our work
- ☞ Run them automatically (every time we change something)
- ☞ End up with a suite of tests describing the functionality
- ☞ Bold and robust refactoring
- ☞ We know if we break something by mistake
- ☞ We know **what** we've broken

Why write tests?

- ☞ Demonstrate the intent
- ☞ Code becomes documentation

Test driven development (TDD)

TDD is a development process where you write the test code first.

> “How can you test something that doesn't exist?”
>
> —Confused Person

Test driven development (TDD)

① Write a small piece of test code

② Run the test to see it fail

③ Make it pass

Test driven development (TDD)

- ☞ Evidence based programming
- ☞ Prove bugs or missing features
- ☞ Protect from regression
- ☞ Write tests for each other
- ☞ Get it out of your brain
- ☞ Be your own user first

Does Test driven development mean we don't have to do up-front design?

Does Test driven development mean we don't have to do up-front design?

Does Test driven development mean we don't have to do up-front design?

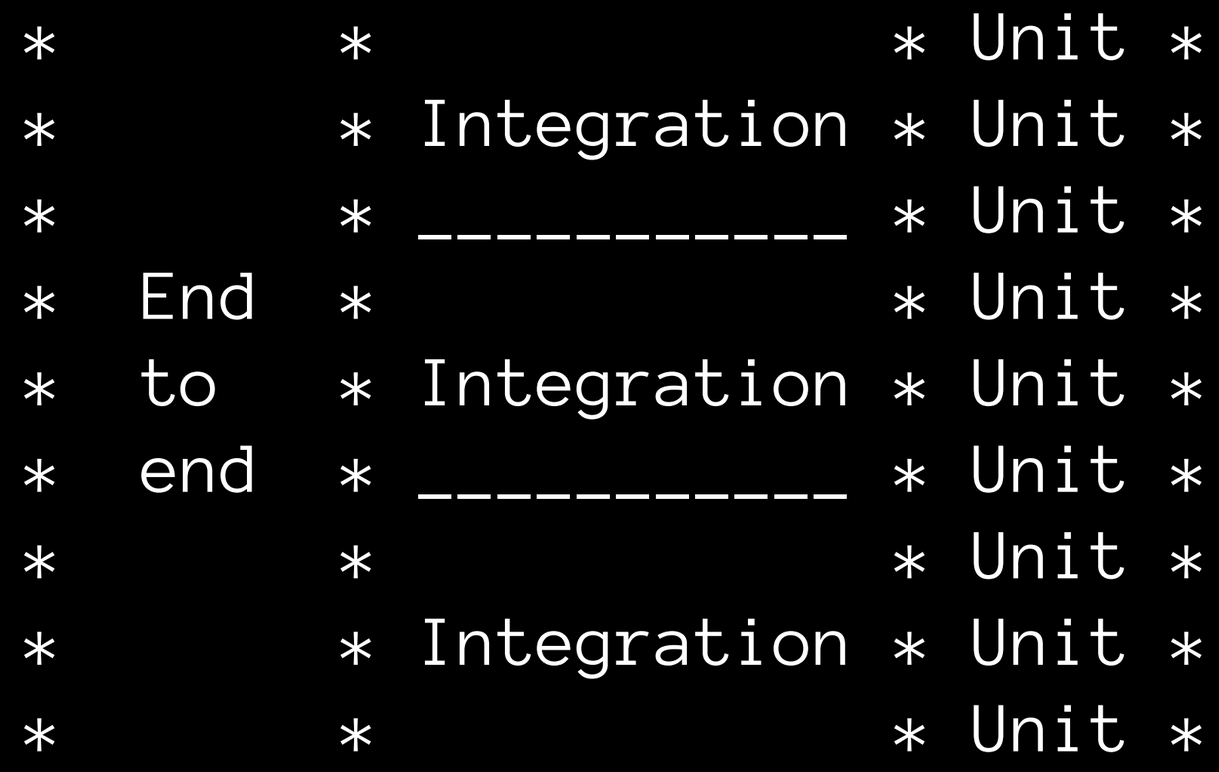
No

Don't skip the design step

TDD is an implementation time activity

Quality tests

Different levels of tests



I 💙 unit tests

"Can I create an account and send money to my friend?"

vs

"When I sign in, am I given an auth token?"

Unit tests :)

Unit tests are tiny, simple and laser focussed

☞ If something breaks, they point directly to it

End to end tests :(

If **anything** breaks, the end-to-end test will fail

I 💙 table tests

```
func TestBatify(t *testing.T) {
    tests := []struct {
        in, out string
    }{
        {"Car", "Bat Car"},
        {"Bike", "Bat Bike"},
        {"Backpack", "Bat Backpack"},
    }
    for _, test := range tests {
        t.Run(test.out, func(t *testing.T) {
            actual := Batify(test.in)
            if actual != test.out {
                t.Errorf("expected %s, but got %s", test.out, actual)
            }
        })
    }
}
```

I 💙 table tests

- ☞ Anonymous `struct` slice for each case
- ☞ One `for...range` loop iterating over each
- ☞ Run each case using `t.Run`
- ☞ Use a sensible value for the name, or add one (e.g. `map`)

Setup and teardown

```
func TestBatify(t *testing.T) {

    // TODO: setup
    // TODO: defer teardown

    for _, test := range tests {
        t.Run(test.out, func(t *testing.T) {

            // TODO: per-test setup
            // TODO: defer per test teardown

        })
    }
}
```

Shared setup and teardown

```
func setup(t *testing.T) (*db, func()) {
    db, err := dial(testDatabase)
    if err != nil {
        t.Errorf("dial: %s", err)
        return nil, func(){}
    }
    return db, func(){
        if err := db.Close(); err != nil {
            t.Errorf("db.Close: %s", err)
        }
    }
}
```

☞ Return a teardown func so setup and teardown code is kept together

```go
func TestStuff(t *testing.T) {

    db, teardown := setup(t)
    defer teardown()

    // use the db

}
```

Different package name

```
// in batify.go
package batify

func Batify(item string) string...

// meanwhile...
// in batify_test.go
package batify_test

import "batify"

func TestBatify(t *testing.T) {
    actual := batify.Batify("Car")
    //...
}
```

Different package name

- ☞ Can only test the public API
- ☞ IDEs will reveal the footprint of your package
- ☞ You will use your package name (avoid `tea.BrewTea` or `tea.TeaBrewer`)

“If you don't trust yourself to not use private methods in unit tests, then I cannot help you”

—Dave Cheney

Different package name

```
package batify

func Item(item string) string {
    return "Bat " + item
}

// now we can call batify.Item
// instead of batify.Batify
```

Interface tests[1]

```go
type Store interface {
    Get(id string) (Storable, error)
    Put(v Storable) (string, error)
}

func TestStore(t *testing.T, s Store) {
    obj := someTestObject{}
    id, err := s.Put(obj)
    if err != nil {
        t.Errorf("Put: %s", err)
    }
    obj2, err := s.Get(id)
    if err != nil {
        t.Errorf("Put: %s", err)
    }
    if obj.ID() != obj2.ID() {
        t.Errorf("object IDs don't match")
    }
}
```

[1] For a real example, see github.com/graymeta/stow - see `/test` folder

Test code is code

- ☞ Treat it as a first class concern
- ☞ Look after it
- ☞ Clear away old tests
- ☞ Know your test codebase intimately

“Don't be afraid to write your own test harness helpers”

—Eleanor McHugh

Top tips from the community...

I asked people on **Gophers Slack** for their top testing tips.

Btw, your invite is waiting at: https://invite.slack.golangbridge.org

TIP: Code coverage

“Shoot for 90% code coverage, 100% of the happy path”

—William Kennedy (Go in Action)

```
go test -cover
```

```
go test -covermode=count -coverprofile=count.out
go tool cover -html=count.out
```

btw, with test driven development, code coverage is automatically **very** high

TIP: Inject your dependencies

```
func TestSignInNotFound(t *testing.T) {
    users := &mockedUserService{
        GetUserFunc: func(email string) (*User, error) {
            return nil, ErrNotFound
        },
    }
    srv := Server{
        UserService: users,
    }
    token, err := srv.SignIn("nope@me.com", password)
    if err != ErrNotFound {
        t.Errorf("ErrNotFound error found")
    }
}
```

from Ernesto Jimenez

Mocking

```
type UserService interface {
    GetUser func(email string) (*User, error)
}

// meanwhile...

type MockedUserService struct {
    GetUserFunc func(email string) (*User, error)
}

func (m *MockedUserService) GetUser(email string) (*User, error) {
    return m.GetUserFunc(email)
}
```

Automagically turn interfaces into mocked structs with github.com/matryer/moq

TIP: Pass `testing.T` into helpers

```
func callBatifyAPI(t *testing.T, item string) string {
    // TODO: make a request
    resp, err := client.Do(req)
    if err != nil {
        t.Errorf("the request failed" %s", err)
    }
}
```

from Coleman McFarland

Should I use a testing framework?

Whatever makes you happy. Whatever makes you test.

☞ Always better to avoid dependencies if you can

☞ Don't give new team members something else to learn

Finally...

- ☞ Women who Go - womenwhogo.org
- ☞ Go Bridge - Building Bridges That Educate & Empower Underrepresented Communities golangbridge.org

You are welcome in the Go community; we want their loss to be our gain.

Thank you

Preorder now Go Programming Blueprints: Second Edition

Tweet me: `@matryer`

Find me after if you have any questions.