AMSTERDAM, JAN 28 2020

THE HABIT OF TDD

Victor Alves
Senior Developer @ Sentia
github.com/vhugo

Before we start...

- Who have heard about TDD before?
- Who thinks it is a good idea?
- Who uses it on the daily basis?

Yet another talk about TDD

There are many talks, articles and books about TDD.

They often reflect the author's point of view.

This talk is no different.

Yet another talk about TDD

You might never need to use TDD, but if you're anything like me, you're constantly looking for ways to improve the quality and productivity of your work.

Yet another talk about TDD

The same way I am passionate about Go, I am also passionate about TDD because it helps me improve the quality and productivity of my work.

Covering the basics

What is TDD?

- stands for Test-Driven Development
- development process that consist of writing tests first
- **red**-**green**-**refactor** cycle
- it is **NOT** a new thing

How to use TDD?

1. design, write and run a test for a part of the requirements

   make sure the test fails with a clear error

2. write the minimal amount

   of code to **make the test pass**.

3. **refactor**, when possible.

then repeat...

WHY use TDD?

- reasons are usually about the effect on quality and productivity
- different reasons in almost every talk, article or book
- even research studies have failed to produce a conclusive result
- the best reason is the one you find that has the most value to you
- I will share my reasons later in this talk…

What I learned about habit

- preprogrammed mode of the brain
- energy saver - free our mind for other activities
- habits are great for processes (Do you see what I'm getting at?)

What I learned about habit

Imagine backing out of a driveway,

once you're used to do it, your brain goes into this

preprogrammed mode, semi-aware of a few key factors

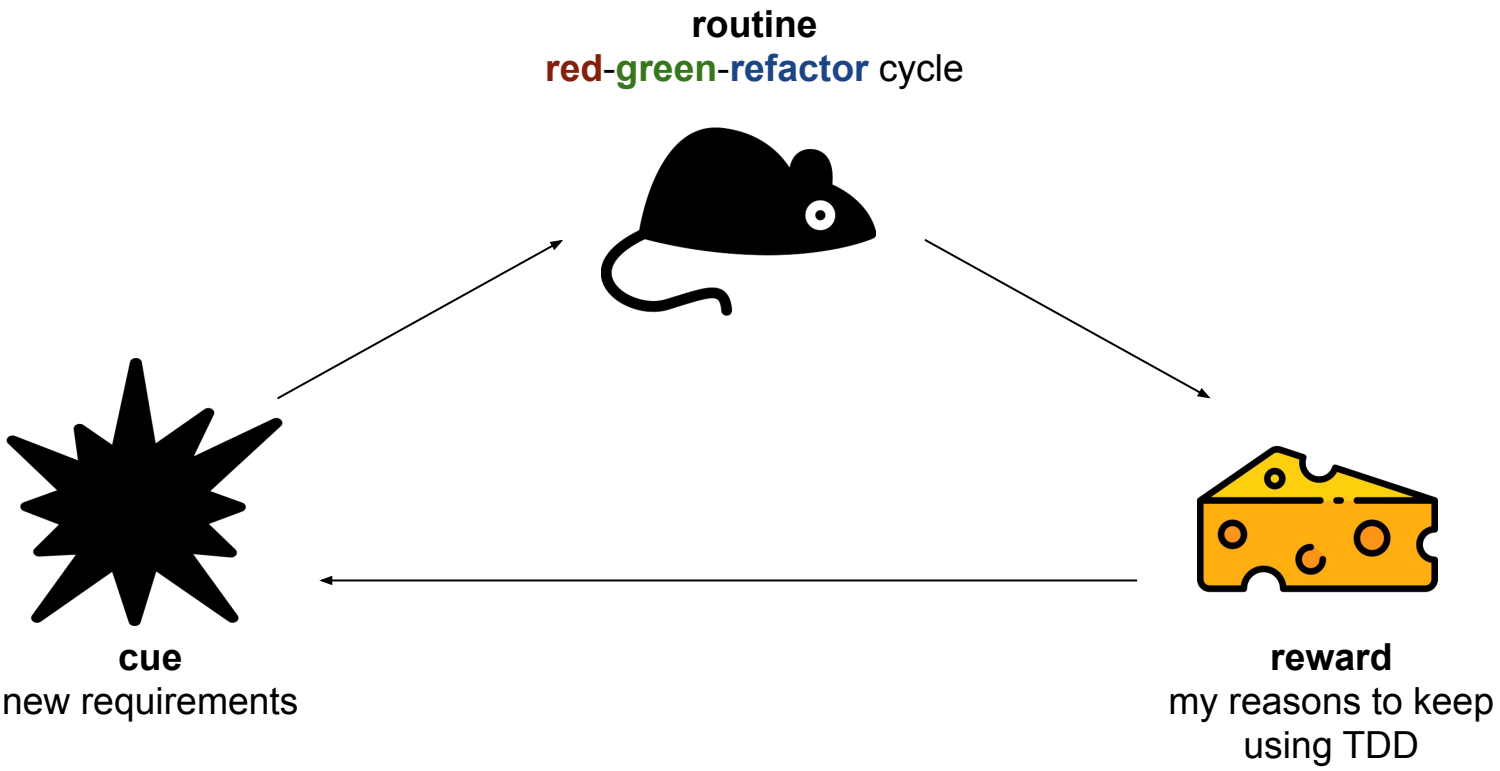
instead of being overwhelmed by all the data.


this is
just like TDD
but I
don't like TDD
imgflip.com

What I learned about habit

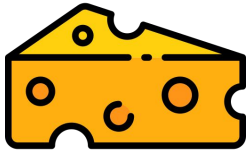
- **Cue** - trigger to react
- **Routine** - sequence of actions
- **Reward** - motivation to keep doing it

My habit of TDD



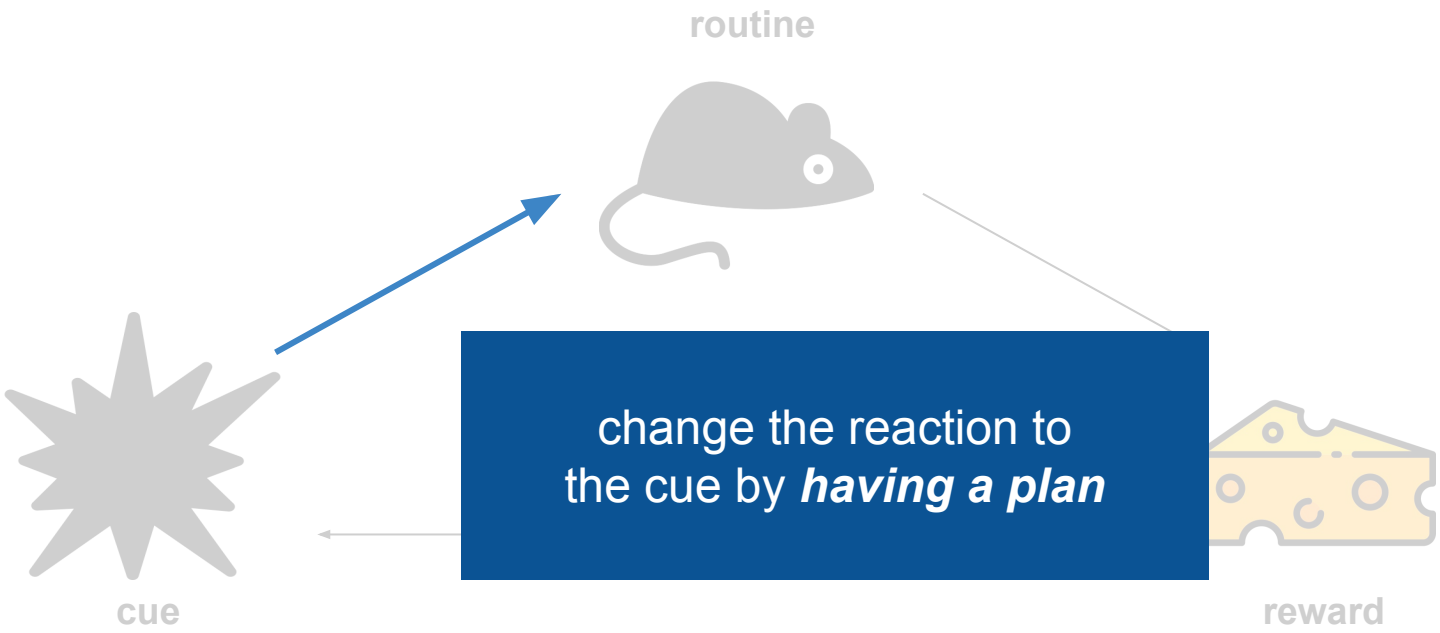
My Reward

- not having to write test afterwards
- clear implementation
- review before peer review
- getting things done (gamification)
- fixing bugs for good



reward
my reasons to keep using TDD

Changing an old habit



My plan

Whenever I start working on a new task, without thinking too much about it, I am going to write a boilerplate code for my test.

hello/hello_test.go

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

    for _, tc := range []struct {
        name string
    }{
        {
            name: "empty",
        },
    }{
        t.Run(tc.name, func(t *testing.T) {

        })
    }
}
```

my boilerplate code

hello/hello_test.go

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

    for _, tc := range []struct {
        name string
    }{
        {
            name: "empty",
        },
    }{
        t.Run(tc.name, func(t *testing.T) {

        })
    }
}
```

Table Driven Test - Data structure for my test cases, starting with a name property

hello/hello_test.go

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

    for _, tc := range []struct {
        name string
    }{
        {
            name: "empty",
        },
    }{
        t.Run(tc.name, func(t *testing.T) {

        })
    }
}
```

My very first test case

hello/hello_test.go

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

    for _, tc := range []struct {
        name string
    }{
        {
            name: "empty",
        },
    }{
        t.Run(tc.name, func(t *testing.T) {

        })
    }
}
```

inside this subtest, I design how my solution is going to be used and test that unit for each test case.

this is my canvas...

hello/hello_test.go

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

    for _, tc := range []struct {
        name string
    }{
        {
            name: "empty",
        },
    }{
        t.Run(tc.name, func(t *testing.T) {

        })
    }
}
```

```
> go test -v
```

TEST PASS

```
=== RUN   TestHello
=== RUN   TestHello/empty
--- PASS: TestHello (0.00s)
    --- PASS: TestHello/empty (0.00s)
PASS
ok      github.com/vhugo/hello  0.006s
```

In Practice:

Let's work on a dead simple task, create a function to say "Hello" with the following requirements:

- ✦ it returns "Hello, World!" by default
- ✦ when receive an argument it returns "Hello, "+ argument +"!"

Requirement 1

it returns "Hello, World!"

by default

hello/hello_test.go

```go
func TestHello(t *testing.T) {
    for _, tc := range []struct {
        name string
        want string
    }{
        {
            name: "default",
            want: "Hello, World!",
        },
    } {
        t.Run(tc.name, func(t *testing.T) {
            got := hello.Hello()

            if tc.want != got {
                t.Fatalf("want %q, got %v", tc.want, got)
            }
        })
    }
}
```

hello/hello_test.go

```go
func TestHello(t *testing.T) {
    for _, tc := range []struct {
        name string
        want string
    }{
        {
            name: "default",
            want: "Hello, World!",
        },
    } {
        t.Run(tc.name, func(t *testing.T) {
            got := hello.Hello()

            if tc.want != got {
                t.Fatalf("want %q, got %v", tc.want, got)
            }
        })
    }
}
```

update my test case data structure

hello/hello_test.go

```go
func TestHello(t *testing.T) {
    for _, tc := range []struct {
        name string
        want string
    }{
        {
            name: "default",
            want: "Hello, World!",
        },
    } {
        t.Run(tc.name, func(t *testing.T) {
            got := hello.Hello()

            if tc.want != got {
                t.Fatalf("want %q, got %v", tc.want, got)
            }
        })
    }
}
```

update my test case

hello/hello_test.go

```go
func TestHello(t *testing.T) {
    for _, tc := range []struct {
        name string
        want string
    }{
        {
            name: "default",
            want: "Hello, World!",
        },
    } {
        t.Run(tc.name, func(t *testing.T) {
            got := hello.Hello()

            if tc.want != got {
                t.Fatalf("want %q, got %v", tc.want, got)
            }
        })
    }
}
```

This is how I designed my unit to be used

hello/hello_test.go

```go
func TestHello(t *testing.T) {
    for _, tc := range []struct {
        name string
        want string
    }{
        {
            name: "default",
            want: "Hello, World!",
        },
    } {
        t.Run(tc.name, func(t *testing.T) {
            got := hello.Hello()

            if tc.want != got {
                t.Fatalf("want %q, got %v", tc.want, got)
            }
        })
    }
}
```

```
> go test -v
```

BUILD FAIL

```
# github.com/vhugo/hello_test [github.com/vhugo/hello.test]
./hello_test.go:20:11: undefined: hello.Hello
FAIL    github.com/vhugo/hello [build failed]
```

hello/hello.go

```go
package hello

func Hello() string {
    return "Hello, World!"
}
```

```
> go test -v
```

TEST PASS

```
=== RUN   TestHello
=== RUN   TestHello/default
--- PASS: TestHello (0.00s)
    --- PASS: TestHello/default (0.00s)
PASS
ok      github.com/vhugo/hello  0.006s
```

Requirement 2

when receive an argument it returns "Hello, "+ argument +"!"

hello/hello_test.go

```go
func TestHello(t *testing.T) {
    for _, tc := range []struct {
        name string
        arg, want string
    }{
        {
            name: "default",
            want: "Hello, World!",
        },
        {
            name: "with argument",
            arg: "Gophers",
            want: "Hello, Gophers!",
        },
    } {
        t.Run(tc.name, func(t *testing.T) {
            got := hello.Hello(tc.arg)
            ...
```

update my test case data structure again

hello/hello_test.go

```go
func TestHello(t *testing.T) {
    for _, tc := range []struct {
        name string
        arg, want string
    }{
        {
            name: "default",
            want: "Hello, World!",
        },
        {
            name: "with argument",
            arg: "Gophers",
            want: "Hello, Gophers!",
        },
    } {
        t.Run(tc.name, func(t *testing.T) {
            got := hello.Hello(tc.arg)
            ...
```

next test case

hello/hello_test.go

```go
func TestHello(t *testing.T) {
    for _, tc := range []struct {
        name string
        arg, want string
    }{
        {
            name: "default",
            want: "Hello, World!",
        },
        {
            name: "with argument",
            arg: "Gophers",
            want: "Hello, Gophers!",
        },
    } {
        t.Run(tc.name, func(t *testing.T) {
            got := hello.Hello(tc.arg)
            ...
```

update the design to receive the argument

hello/hello.go

```go
package hello

func Hello(s string) string {
    return "Hello, World!"
}
```

update the implementation and avoid a build failure

hello/hello.go

```go
package hello

func Hello(s string) string {
    return "Hello, World!"
}
```

```
> go test -v
```

TEST FAIL

```
=== RUN   TestHello
=== RUN   TestHello/default
=== RUN   TestHello/with_argument
--- FAIL: TestHello (0.00s)
    --- PASS: TestHello/default (0.00s)
    --- FAIL: TestHello/with_argument (0.00s)
        hello_test.go:27: want "Hello, Gophers!", got "Hello,
World!"
FAIL
exit status 1
FAIL    github.com/vhugo/hello  0.007s
```

hello/hello.go

```go
package hello

func Hello(s string) string {
    if s != "" {
        return "Hello, " + s + "!"
    }
    return "Hello, World!"
}
```

update the implementation
to use the argument

hello/hello.go

```go
package hello

func Hello(s string) string {
    if s != "" {
        return "Hello, " + s + "!"
    }
    return "Hello, World!"
}
```

```
> go test -v
```

TEST PASS

```
=== RUN   TestHello
=== RUN   TestHello/default
=== RUN   TestHello/with_argument
--- PASS: TestHello (0.00s)
    --- PASS: TestHello/default (0.00s)
    --- PASS: TestHello/with_argument (0.00s)
PASS
ok      github.com/vhugo/hello  0.008s
```

Take away

The purpose of this talk is to motivate you to give TDD a fair try - even if you have tried before and it didn't work for you or if you never tried.

Take away

You are the one who knows what is more valuable to you, so if you're interested in TDD keep searching for reasons that make sense to you.

References

- Learn Go with tests (TDD) (great tutorial to get started with Go and TDD)
- The Effects of Test-Driven Development on External Quality and Productivity: A Meta-Analysis (paper reviewing other papers about TDD)
- Learning how to learn (learned about habits)
- Hello World (repo with code used in this presentation)

References (talks)

Interesting talks:

- TDD for those who don't need it - GopherCon SG 2017 (Chew Choon Keat)
- TDD, Where Did It All Go Wrong? - DevTernity 2017 (Ian Cooper)
- Absolute Unit (Test) - London Gophers 2019 (Dave Cheney)
- Advanced Testing with Go - GopherCon 2017 (Mitchell Hashimoto)