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

- like other talks, but my experience
- YAGNI
- same appreciation for Go

Covering the basics

What is TDD?

- Test-Driven Development
- development process
- writing tests first
- **red**-**green**-**refactor** cycle
- it is **NOT** a new thing (Alan Perlis - 1968)

How to use TDD?

1. design, write and run a test

   make sure the test fails.

2. write the minimal amount

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

3. **refactor**, when possible.

then repeat...

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?)

Backing out of a driveway

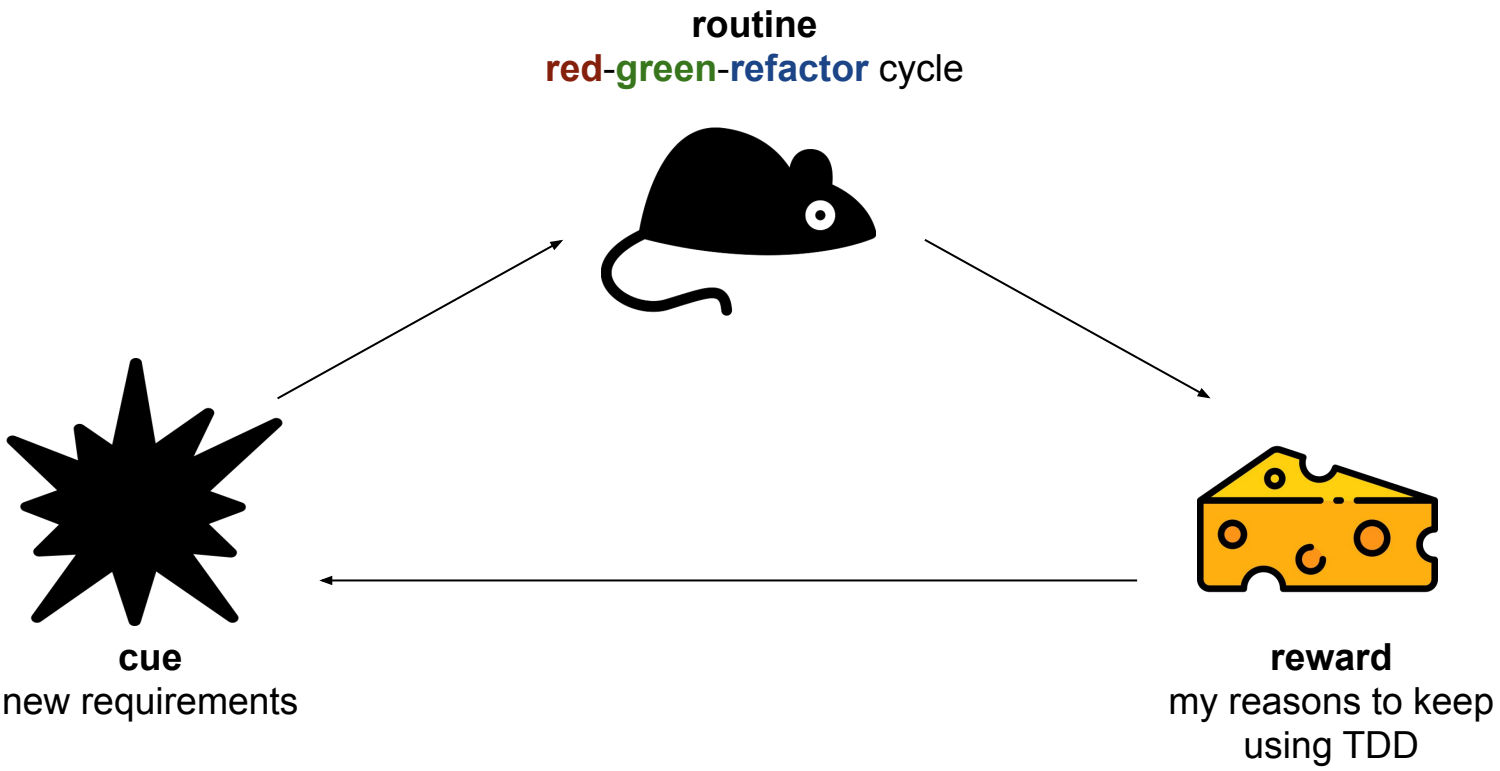
- your brain goes into this preprogrammed mode
- semi-aware of key factors
- instead of overwhelmed by all the data


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

What I learned about habit

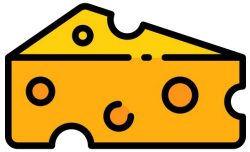
- **Cue** - trigger to react
- **Routine** - sequence of actions
- **Reward** - motivation to keep doing it

My habit of TDD



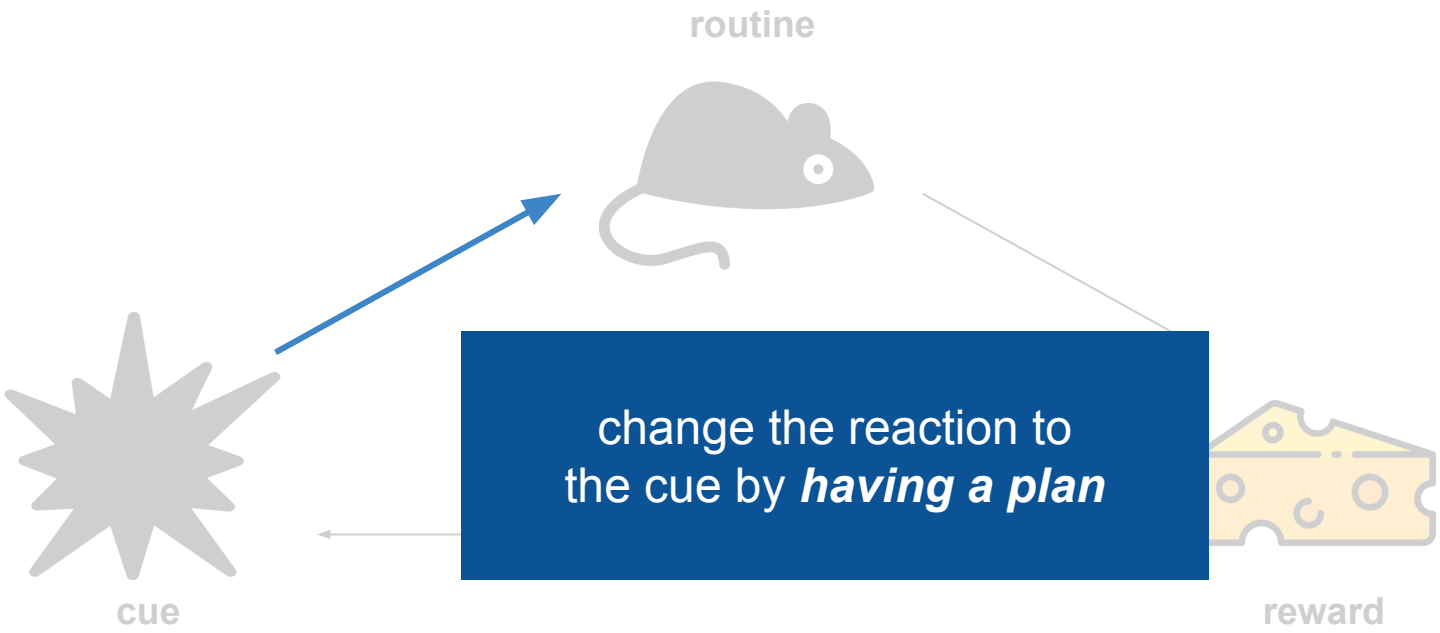
My Reward

- 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

- new task?
- do not think too much about
- write boilerplate test 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) {

        })
    }
}
```

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:

Dead simple task, a function to say **"Hello"** that

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

- ✦ motivate to give TDD a fair try
- ✦ you know what is more valuable
- ✦ keep searching for your reasons

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)