A Gentle Introduction

TDD IN GO

Practice, libraries and tools for Test-Driven Development in the Go language

Luciano Ramalho | @standupdev | @ramalhoorg

Repo with examples and slides: https://tgo.li/tddgo
Sometimes you need a blank template.

LUCIANO RAMALHO

Technical Principal

@ramalhoorg
luciano.ramalho@thoughtworks.com
**FLUENT PYTHON, MY FIRST BOOK**

Fluent Python *(O’Reilly, 2015)*

Python Fluente *(Novatec, 2015)*

Python к вершинам мастерства *(DMK, 2015)*

流暢的 Python *(Gotop, 2016)*

also in Simplified Chinese, Polish, Korean...

🌟🌟🌟🌟🌟 87
4.6 out of 5 stars

More than 40,000 copies sold as of May, 2018
VISION FOR THIS TUTORIAL

Make the most of our time together
Lots of live coding, together
Simple yet complete app built from tests
Discussion of TDD styles and variations
Brief overview of tools and libraries
Lots of references for further study

Repo with examples and slides: https://tgo.li/tddgo
INTRODUCTION

What TDD is all about
TRADITIONAL TESTING

implement implement implement implement test

Feedback about correctness and class design

Source: Test-Driven Development, by Hugo Corbucci and Mauricio Aniche
Feedback about correctness and class design

Source: *Test-Driven Development*, by Hugo Corbucci and Mauricio Aniche
TDD CYCLE

1. Write the simplest test for next functionality

2. Watch it fail

3. Implement the simplest solution to satisfy the test

4. Refactor to remove data and code duplication

Source: Test-Driven Development, by Hugo Corbucci and Mauricio Aniche
TDD BEST PRACTICE: BABY STEPS

Work on small increments.

Time between red/green states measured in minutes, not hours.

At first: practice with the smallest increments you can think.

Like 4L on 4WD: be willing and ready to engage reduced gear when the going gets tough.
TDD BEST PRACTICE: CALL SHOT

Before running test, call out expected outcome.

When pairing, co-pilot should call outcome.

— Test will error out because there’s no method named `parseLine`.

— Test will fail because function returns 1, but the expected result is 42.

— Test will pass.

Source: *Test-Driven Development*, by Hugo Corbucci and Mauricio Aniche
TDD BEST PRACTICE: IMPROVE FAILING REPORTS

Source: Growing Object-Oriented Software, Guided by Tests
by Steve Freeman, Nat Pryce
Let's practice
Rotating pairs of pilot and co-pilot.

After 7 minutes, call volunteer for co-pilot.

When tests are green, audience can make suggestions for refactoring or next test.

When a test is red, audience should only offer suggestions when requested by pair.
OUR GOAL

$\texttt{runes\ cat\ eyes}$

U+1F638 😻 GRINNING CAT FACE WITH SMILING EYES
U+1F63B 😍 SMILING CAT FACE WITH HEART-SHAPED EYES
U+1F63D 😻 KISSING CAT FACE WITH CLOSED EYES

$\texttt{}$
An Example() function in a *_test file is a test.

```go
func Example() {
    main()
    // Output:
    // Please provide one or more words to search.
}
```

Program output is compared to Output: comment.
TESTING PACKAGE: T TYPE API

type PB
  ○ func (pb *PB) Next() bool

type T
  ○ func (c *T) Error(args ...interface{})
  ○ func (c *T) Errorf(format string, args ...interface{})
  ○ func (c *T) Fail()
  ○ func (c *T) FailNow()
  ○ func (c *T) Failed() bool
  ○ func (c *T) Fatal(args ...interface{})
  ○ func (c *T) Fatalf(format string, args ...interface{})
  ○ func (c *T) Helper()
  ○ func (c *T) Log(args ...interface{})
  ○ func (c *T) Logf(format string, args ...interface{})
  ○ func (c *T) Name() string
  ○ func (t *T) Parallel()
  ○ func (t *T) Run(name string, f func(t *T)) bool
  ○ func (c *T) Skip(args ...interface{})
  ○ func (c *T) SkipNow()
  ○ func (c *T) Skipf(format string, args ...interface{})
  ○ func (c *T) Skipped() bool

type TB
func TestMake(t *testing.T) {
    testCases := []struct {
        elems  []string
        wantLen int
    }{
        {[]string{}, 0},
        {[]string{"a"}, 1},
        {[]string{"a", "b"}, 2},
        {[]string{"a", "b", "a"}, 2},
    }
    for _, tc := range testCases {
        t.Run(fmt.Sprintf("%v gets %d", tc.elems, tc.wantLen), func(t *testing.T) {
            s := Make(tc.elems...)
            assert.Equal(t, tc.wantLen, s.Len())
        })
    }
}
Index

Variables

func CallerInfo() [string
func Condition(t TestingT, comp Comparison, msgAndArgs ...interface{}) bool
func Condition(t TestingT, comp Comparison, msgAndArgs ...interface{}) bool
func Contains(t TestingT, s, contains interface{}, msgAndArgs ...interface{}) bool
func Contains(t TestingT, s interface{}, contains interface{}, msgAndArgs ...interface{}) bool
func DirExists(t TestingT, path string, msgAndArgs ...interface{}) bool
func DirExists(t TestingT, path string, msgAndArgs ...interface{}) bool
func ElementsMatch(t TestingT, listA, listB interface{}, msgAndArgs ...interface{}) (ok bool)
func ElementsMatch(t TestingT, listA interface{}, listB interface{}, msgAndArgs ...interface{}) bool
func Empty(t TestingT, object interface{}, msgAndArgs ...interface{}) bool
func Empty(t TestingT, object interface{}, msgAndArgs ...interface{}) bool
func Equal(t TestingT, expected, actual interface{}, msgAndArgs ...interface{}) bool
func Equal(t TestingT, expected, actual interface{}, msgAndArgs ...interface{}) bool
func EqualError(t TestingT, theError error, errString string, msgAndArgs ...interface{}) bool
func EqualError(t TestingT, theError error, errString string, msgAndArgs ...interface{}) bool
func EqualValues(t TestingT, expected, actual interface{}, msgAndArgs ...interface{}) bool
func EqualValues(t TestingT, expected, actual interface{}, msgAndArgs ...interface{}) bool
func Equalf(t TestingT, expected interface{}, actual interface{}, msgAndArgs ...interface{}) bool
func Error(t TestingT, err error, msgAndArgs ...interface{}) bool
func Error(t TestingT, err error, msgAndArgs ...interface{}) bool
func Exactly(t TestingT, expected, actual interface{}, msgAndArgs ...interface{}) bool

93 functions!
func Example_2WordQuery() {  // 1
    oldArgs := os.Args // 2
    defer func() { os.Args = oldArgs }()
    os.Args = []string{"", "cat", "smiling"}
    main() // 3

    // Output:
    // U+1F638 😷 GRINNING CAT FACE WITH SMILING EYES
    // U+1F63A 😷 SMILING CAT FACE WITH OPEN MOUTH
    // U+1F63B 😷 SMILING CAT FACE WITH HEART-SHAPED EYES
}
TOOLS

Libraries and utilities for TDD in Go
### Go testing libraries with most Github stars*

<table>
<thead>
<tr>
<th>Stars</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>5251</td>
<td>stretchr/testify</td>
</tr>
<tr>
<td>3685</td>
<td>smartystreets/goconvey</td>
</tr>
<tr>
<td>2166</td>
<td>onsi/ginkgo</td>
</tr>
<tr>
<td>1452</td>
<td>golang/mock</td>
</tr>
<tr>
<td>902</td>
<td>DATA-DOG/go-sqlmock</td>
</tr>
<tr>
<td>884</td>
<td>gavv/httpexpect</td>
</tr>
<tr>
<td>709</td>
<td>onsi/gomega</td>
</tr>
<tr>
<td>575</td>
<td>google/go-cmp</td>
</tr>
<tr>
<td>512</td>
<td>franela/goblin</td>
</tr>
<tr>
<td>502</td>
<td>h2non/baloo</td>
</tr>
<tr>
<td>496</td>
<td>h2non/gock</td>
</tr>
<tr>
<td>404</td>
<td>DATA-DOG/godog</td>
</tr>
<tr>
<td>387</td>
<td>go-check/check</td>
</tr>
</tbody>
</table>

*Data collected 2018-07-05
TECHNIQUES

Coding tips
func ExampleMake() {
    w := []string{"beta", "alpha", "gamma", "beta"}
    s := Make(w...)
    fmt.Println(s)
    // Output: Set{alpha beta gamma}
}
func ExampleSet_Channel() {
    set := MakeFromText("beta alpha delta gamma")
    // iteration order over underlying map is undefined
    for elem := range set.Channel() {
        fmt.Println(elem)
    }
    // Unordered output:
    // alpha
    // beta
    // delta
    // gamma
}
func restore(nameVar, value string, existed bool) {
    if existed {
        os.Setenv(nameVar, value)
    } else {
        os.Unsetenv(nameVar)
    }
}

func TestGetUCDPath_isSet(t *testing.T) {
    pathBefore, existed := os.LookupEnv("UCD_PATH")
    defer restore("UCD_PATH", pathBefore, existed)
    ucdPath := fmt.Sprintf("./TEST%d-UnicodeData.txt", time.Now().UnixNano())
    os.Setenv("UCD_PATH", ucdPath)
    got := getUCDPath()
    if got != ucdPath {
        t.Errorf("getUCDPath() [set]\nwant: %q; got: %q", ucdPath, got)
    }
}
const dataStr = \`\003C;LESS–THAN SIGN;Sm;0;ON;;;;Y;;;;;
\003D;EQUALS SIGN;Sm;0;ON;;;;N;;;;;
\003E;GREATER–THAN SIGN;Sm;0;ON;;;;Y;;;;;
\003F;QUESTION MARK;Po;0;ON;;;;N;;;;;
\0040;COMMERCIAL AT;Po;0;ON;;;;N;;;;;
\0041;LATIN CAPITAL LETTER A;Lu;0;L;;;;;N;;;;0061;
\0042;LATIN CAPITAL LETTER B;Lu;0;L;;;;;N;;;;0062;
\`

func TestFilter(t *testing.T) {
    query := "sign"
    data := strings.NewReader(dataStr)
    got := Filter(data, query)
    want := []string{
        "U+003C\t<\tLESS–THAN SIGN",
        "U+003D\t=\tEQUALS SIGN",
        "U+003E\t>\tGREATER–THAN SIGN",
    }
    assert.Equal(t, want, got)
}
Instead of specific types, make your functions accept common, narrow interfaces like io.Reader

```go
func Filter(data io.Reader, query string) []string {  
    queryTerms := strset.MakeFromText(strings.ToUpper(query))  
    scanner := bufio.NewScanner(data)  
    result := []string{}  
    for scanner.Scan() {  
        name, code := parseLine(scanner.Text())  
        if match(queryTerms, name) {  
            line := fmt.Sprintf("U+%04X\t%c\t%s", code, code, name)  
            result = append(result, line)  
        }  
    }  
    return result
}
```
The httptest package in the standard libraries provides doubles for testing HTTP clients and servers
func TestOpenUCD_remote(t *testing.T) {
    if testing.Short() { // ①
        t.Skip("skipped test \[-test.short option\]") // ②
    }
    ucdPath := fmt.Sprintf("./TEST%d-UnicodeData.txt", time.Now().UnixNano())
    ucd, err := openUCD(ucdPath)
    if err != nil {
        t.Errorf("openUCD(%q):
%v", ucdPath, err)
    }
    ucd.Close()
    os.Remove(ucdPath)
}
VARIATIONS

Beyond the classic TDD cycle
**TDD CYCLE: REFACTOR AFTER TEST**

Source: *Growing Object-Oriented Software, Guided by Tests* by Steve Freeman, Nat Pryce
TDD CYCLES: MOCKIST STYLE

Source: Growing Object-Oriented Software, Guided by Tests by Steve Freeman, Nat Pryce
TDD STYLES

Chicago style, a.k.a. “classic”
Mostly inside-out: from unit tests to acceptance tests

London style, a.k.a. “mockist”
Mostly outside-in: from acceptance tests to unit tests
“Go eschews mocks and fakes in favour of writing code that takes broad interfaces.”

“That’s generally how we get around dependency injection frameworks and large mocking frameworks: just by writing code that uses small interfaces. Then we have small fakes like the ResponseRecorder — small fakes that allow us to inspect how they were used. There are frameworks that generate those kinds of fakes — one of them is called Go Mock [...]. They're fine, but I find that on balance the hand-written fakes tend to be easier to reason about, and clearer to see what is going on. That's my personal experience. But I am not an "enterprise" Go programmer so maybe people need that, I don't know. That's my advice.”

The term 'Mock Objects' has become a popular one to describe special case objects that mimic real objects for testing. Most language environments now have frameworks that make it easy to create mock objects. What's often not realized, however, is that mock objects are but one form of special case test object, one that enables a different style of testing. In this article I'll explain how mock objects work, how they encourage testing based on behavior verification, and how the community around them uses them to develop a different style of testing.

Contents
- Regular Tests
- Tests with Mock Objects
  - Using EasyMock
- The Difference Between Mocks and Stubs
- Classical and Mockist Testing
- Choosing Between the Differences
  - Driving TDD
  - Fixture Setup
  - Test Isolation
  - Coupling Tests to Implementations
  - Design Style
- So should I be a classicist or a mockist?
REFERENCES

Where to learn more
REFERENCES: BOOKS

Kent Beck: **Test Driven Development: By Example** [https://tgo.li/2NvBfcX](https://tgo.li/2NvBfcX)

Steve Freeman, Nat Pryce: **Growing Object-Oriented Software, Guided by Tests** [https://tgo.li/2tV8QoK](https://tgo.li/2tV8QoK)

Hugo Corbucci and Mauricio Aniche (book in Portuguese): **Test-Driven Development: Teste e design no mundo real com Ruby** [https://tgo.li/2zGSI4N](https://tgo.li/2zGSI4N)
REFERENCES: POSTS, VIDEOS

Andrew Gerrand [video]
**Go Testing Techniques (Google I/O 2014)**  [https://tgo.li/2upCkek](https://tgo.li/2upCkek)

Francesc Campoy [video]
**Unit Testing HTTP Servers (justforfunc #16)**  [https://tgo.li/2NSEGdZ](https://tgo.li/2NSEGdZ)

Martin Angers [post]
**Lesser-known Features of Go-Test**  [https://tgo.li/2m7ta1E](https://tgo.li/2m7ta1E)

Martin Fowler [post]
**Mocks Aren't Stubs**  [https://tgo.li/2lUqTXv](https://tgo.li/2lUqTXv)

Martin Fowler, Kent Beck, David Heinemeier Hansson [post + videos]
**Is TDD Dead?**  [https://tgo.li/2lWOAYn](https://tgo.li/2lWOAYn)

Michael Feathers , Steve Freeman [video]
**Test Driven Development: Ten Years Later**  [https://tgo.li/2KD2Gnm](https://tgo.li/2KD2Gnm)
THANK YOU

Let’s connect!

Luciano Ramalho
@standupdev | @ramalhoorg
luciano.ramalho@thoughtworks.com