How to test in Go?

with few examples

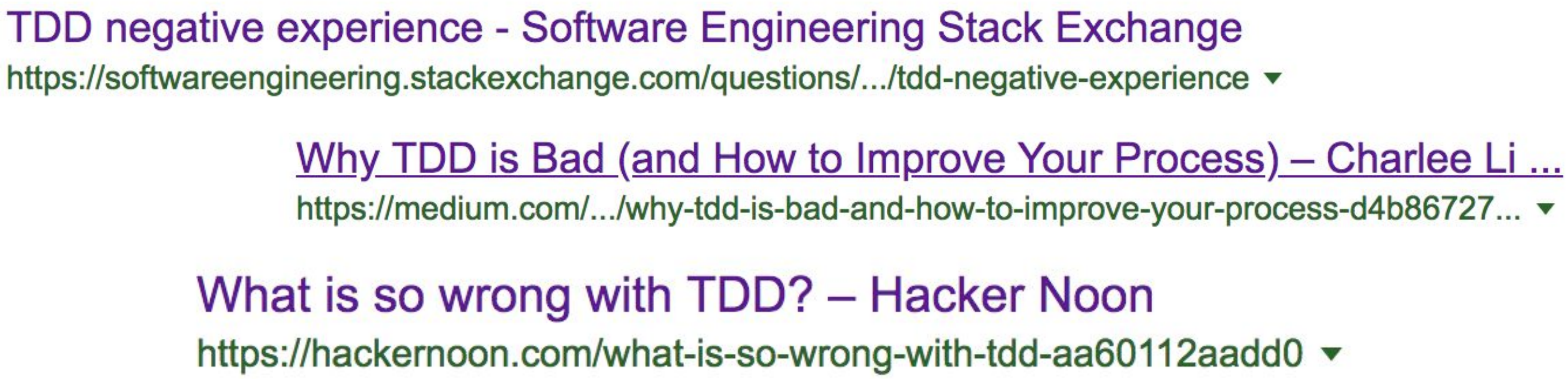
Development is complicated

Divide and Conquer

- Tests always fail for a reason
- Tests build developer’s confidence
- Tests speed up feedback loop
- Tests are fun to write
- Tests save time for manual clicking
- Tests allow easy refactoring
- Tests...

Is TDD bad?

People criticized the approach and now very few of us writes them :(



TDD negative experience - Software Engineering Stack Exchange
https://softwareengineering.stackexchange.com/questions/.../tdd-negative-experience ▾

Why TDD is Bad (and How to Improve Your Process) – Charlee Li ...
https://medium.com/.../why-tdd-is-bad-and-how-to-improve-your-process-d4b86727... ▾

What is so wrong with TDD? – Hacker Noon
https://hackernoon.com/what-is-so-wrong-with-tdd-aa60112aadd0 ▾

Design first!

“A bad design with a complete test suite is still a bad design” - Rich Hickey

Myth #1

“it takes too much time to write tests”

Myth #1

“it takes too much time to write tests”

- saves time for deployments

Myth #1

“it takes too much time to write tests”

- saves time for deployments
- saves time for debugging and bug fixing

Myth #1

“it takes too much time to write tests”

- saves time for deployments
- saves time for debugging and bug fixing
- it makes your work more structured and less chaotic

Myth #1

“it takes too much time to write tests”

- saves time for deployments
- saves time for debugging and bug fixing
- it makes your work more structured and less chaotic
- it makes returning to project after a long break way easier and less stressful

Myth #2

“writing tests is difficult”

- it may be when you write tests after implementation

Myth #2

“writing tests is difficult”

- it may be when you write tests after implementation
- mindset change to write tests first is difficult indeed

Myth #3

“100% coverage is unsustainable”

~~Myth #3~~

~~“100% coverage is unsustainable”~~

~~Myth #3~~

~~“100% coverage is unsustainable”~~

- it doesn’t mean near 0% code coverage is good

Developer’s Bliss

- Good Design
- Valuable Tests
- Risk Management

Testing

Libraries for testing codebases and generating test data.

- Testing Frameworks
  - assert - Basic Assertion Library used along side native go testing, with building blocks for custom assertions.
  - badio - Extensions to Go's `testing/iotest` package.
  - baloo - Expressive and versatile end-to-end HTTP API testing made easy.
  - biff - Bifurcation testing framework, BDD compatible.
  - bro - Watch files in directory and run tests for them.
  - charlatan - Tool to generate fake interface implementations for tests.
  - cupaloy - Simple snapshot testing addon for your test framework.
  - dbcleaner - Clean database for testing purpose, inspired by `database_cleaner` in Ruby.
  - dsunit - Datastore testing for SQL, NoSQL, structured files.
  - endly - Declarative end to end functional testing.
  - frisby - REST API testing framework.
  - ginkgo - BDD Testing Framework for Go.
  - go-carpet - Tool for viewing test coverage in terminal.
  - go-cmp - Package for comparing Go values in tests.
  - go-mutesting - Mutation testing for Go source code.
  - go-testdeep - Extremely flexible golang deep comparison, extends the go testing package.
  - go-vcr - Record and replay your HTTP interactions for fast, deterministic and accurate tests.
  - goblin - Mocha like testing framework fo Go.
  - gocheck - More advanced testing framework alternative to gotest.
  - GoConvey - BDD-style framework with web UI and live reload.
  - gocrest - Composable hamcrest-like matchers for Go assertions.
  - godog - Cucumber or Behat like BDD framework for Go.
  - gofight - API Handler Testing for Golang Router framework.
  - gogiven - YATSPEC-like BDD testing framework for Go.
  - gomatch - library created for testing JSON against patterns.
  - gomega - Rspec like matcher/assertion library.
  - GoSpec - BDD-style testing framework for the Go programming language.
  - gospecify - This provides a BDD syntax for testing your Go code. It should be familiar to anybody who has used libraries such as rspec.
  - gosuite - Brings lightweight test suites with setup/teardown facilities to `testing` by leveraging Go1.7's Subtests.
  - gotest.tools - A collection of packages to augment the go testing package and support common patterns.
  - Hamcrest - fluent framework for declarative Matcher objects that, when applied to input values, produce self-describing results.
  - httpexpect - Concise, declarative, and easy to use end-to-end HTTP and REST API testing.
  - jsonassert - Package for verifying that your JSON payloads are serialized correctly.
  - restit - Go micro framework to help writing RESTful API integration test.
  - testfixtures - A helper for Rails' like test fixtures to test database applications.
  - Testify - Sacred extension to the standard go testing package.
  - testsql - Generate test data from SQL files before testing and clear it after finished.
  - Tt - Simple and colorful test tools.
  - wstest - Websocket client for unit-testing a websocket http.Handler.
- Mock
  - counterfeiter - Tool for generating self-contained mock objects.
  - go-sqlmock - Mock SQL driver for testing database interactions.
  - go-txdb - Single transaction based database driver mainly for testing purposes.
  - gock - Versatile HTTP mocking made easy.
  - gomock - Mocking framework for the Go programming language.
  - govcr - HTTP mock for Golang: record and replay HTTP interactions for offline testing.
  - hoverfly - HTTP(S) proxy for recording and simulating REST/SOAP APIs with extensible middleware and easy-to-use CLI.
  - minimock - Mock generator for Go interfaces.
  - mockhttp - Mock object for Go http.ResponseWriter.
- Fuzzing and delta-debugging/reducing/shrinking.
  - go-fuzz - Randomized testing system.
  - gofuzz - Library for populating go objects with random values.
  - Tavor - Generic fuzzing and delta-debugging framework.
- Selenium and browser control tools.
  - cdp - Type-safe bindings for the Chrome Debugging Protocol that can be used with browsers or other debug targets that implement it.
  - chromedp - a way to drive/test Chrome, Safari, Edge, Android Webviews, and other browsers supporting the Chrome Debugging Protocol.
  - ggr - a lightweight server that routes and proxies Selenium Wedriver requests to multiple Selenium hubs.
  - selenoid - alternative Selenium hub server that launches browsers within containers.

No!

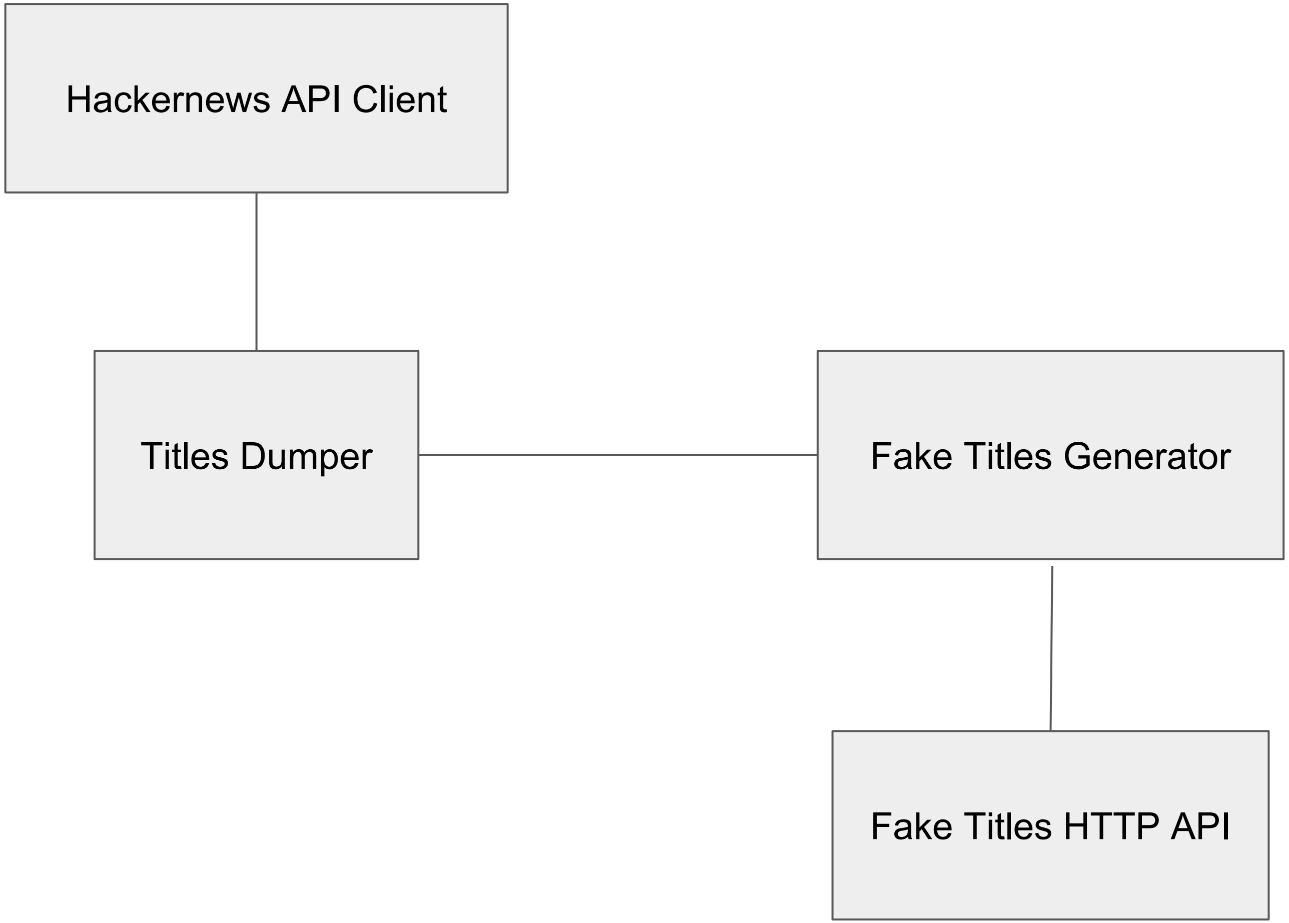
```
import (
     "github.com/onsi/ginkgo"
     "github.com/onsi/gomega"
     "github.com/smartystreets/goconvey"
     "gopkg.in/go-playground/assert.v1"
     "github.com/stretchr/testify/suite"
 )
```

Yes!

```
import (
     "testing"
     "github.com/stretchr/testify/assert"
     "github.com/stretchr/testify/require"
     "github.com/golang/mock/gomock"
     "net/http/httptest"
     "github.com/davecgh/go-spew/spew"
)
```

The Hackernews Turing Test

Building and testing a bot to create fake Hackernews headlines


Hackernews API Client
Titles Dumper
Fake Titles Generator
Fake Titles HTTP API

HTTP API in Go

```go
type Handler interface {
  ServeHTTP(ResponseWriter, *Request)
}

type HandlerFunc func(ResponseWriter, *Request)

func (f HandlerFunc) ServeHTTP(w ResponseWriter, r *Request) {
  f(w, r)
}
```

Hackernews API Client

```go
type Client interface {
  Top() []int
  Get(id int) Item
}
```

httptest server

```go
func TestTopStories(t *testing.T)  {
  handler := http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
     w.WriteHeader(500)
  })

  server := httptest.NewServer(handler)
  defer server.Close()

  client := &client{topUrl: func() string {
     return server.URL
  }}

  ids := client.Top()
  assert.Empty(t, ids)
}
```

gomock

```
//go:generate mockgen -package mock -destination ./hn_client.go goat/pkg/hn Client
package mock
```

gomock

```go
t.Run("writer error", func(t *testing.T) {
	ctrl := gomock.NewController(t)
	defer ctrl.Finish()

	client := mock.NewMockHnClient(ctrl)

	client.EXPECT().Top().Return([]int{1})
	client.EXPECT().Get(1).Return(hn.Item{Title: "A"})

	err := NewDumper(client).Dump(&errWriter{})
	assert.Error(t, err)
})
```

Interfaces

```go
func (d *Dumper) Dump(w io.Writer) error {
// ...
}

type Writer interface {
        Write(p []byte) (n int, err error)
}

type errWriter struct {
}

func (e errWriter) Write(p []byte) (int, error) {
        return 0, errors.New("boom")
}
```

Naive implementation

```go
func (d *Dumper) Dump(w io.Writer) error {
  ids := d.client.Top()
  for _, id := range ids {
     item := d.client.Get(id)
     _, err := w.Write([]byte(item.Title + "\n"))
     if err != nil {
        return err
     }
  }
  return nil
}
```

Table tests

```go
func TestSimpleGenerator(t *testing.T) {
  testcases := []struct {
     FilePath, ExpectedPrefix string
  }{
     {FilePath: "testdata/simple-1.txt", ExpectedPrefix: "Ala ma kota."},
     {FilePath: "testdata/simple-2.txt", ExpectedPrefix: "Ala ma"},
  }

  for _, tc := range testcases {
     buf, err := ioutil.ReadFile(tc.FilePath)
     if assert.NoError(t, err) {
        g := NewGenerator(bytes.NewReader(buf))
        assert.True(t, strings.HasPrefix(g.RandomTitle(), tc.ExpectedPrefix))
     }
  }
}
```

Brute-force tests

```go
func TestGeneratorBruteForce(t *testing.T) {
  buf, err := ioutil.ReadFile("testdata/simple-2.txt")
  require.NoError(t, err)

  g := NewGenerator(bytes.NewReader(buf))

  for i := 0; i < 100; i++ {
     title := g.RandomTitle()
     assert.True(t, strings.HasPrefix(title, "Ala ma "))
     assert.True(t, strings.HasSuffix(title, "psa.") || strings.HasSuffix(title, "kota."))
  }
}
```

Evolution of HTTP API #1

```
func fakenews(w http.ResponseWriter, r *http.Request) {
     w.WriteHeader(http.StatusOK)
     title := generator.RandomTitle()
     io.WriteString(w, title)
}
```

Evolution of HTTP API #1

```go
func fakenews(w http.ResponseWriter, r *http.Request) {
	w.WriteHeader(http.StatusOK)
	title := generator.RandomTitle()
	io.WriteString(w, title)
}

func TestFakenews(t *testing.T) {
	w := httptest.NewRecorder()
	req, err := http.NewRequest("GET", "/???", nil)
	assert.NoError(t, err)

	fakenews(w, req)

	assert.Equal(t, 200, w.Code)
	assert.Equal(t, "???", w.Body.String())
}
```

Evolution of HTTP API #2

```go
func TestFakenews(t *testing.T) {
	ctrl := gomock.NewController(t)
	defer ctrl.Finish()

	generator := mock.NewMockGenerator(ctrl)
	generator.EXPECT().RandomTitle().Return("foo!")

	w := httptest.NewRecorder()
	req, err := http.NewRequest("GET", "/fakenews", nil)
	require.NoError(t, err)

	NewApp(generator).ServeHTTP(w, req)

	assert.Equal(t, 200, w.Code)
	assert.Equal(t, "foo!", w.Body.String())
}
```

Evolution of HTTP API #3

```go
type app struct {
  generator ai.Generator
  router *http.ServeMux
}

func NewApp(generator ai.Generator) http.Handler {
  app := &app{
     generator: generator,
     router: http.NewServeMux(),
  }

  app.router.Handle("/fakenews", app.fakenews())

  return app.router
}
```

Evolution of HTTP API #4

```go
func (a *app) fakenews() http.HandlerFunc {
  return func (w http.ResponseWriter, r *http.Request) {
     w.WriteHeader(http.StatusOK)
     title := a.generator.RandomTitle()
     io.WriteString(w, title)
  }
}
```

Middleware

```
func (f http.HandlerFunc) http.HandlerFunc
```

Middleware

```go
app.router.Handle("/fakenews", app.logExecutionTime(app.fakenews()))


func (a *app) logExecutionTime(f http.HandlerFunc) http.HandlerFunc {
  // one time init...
  return func(w http.ResponseWriter, r *http.Request) {
     start := time.Now()
     f(w, r)
     println(time.Now().Sub(start).Nanoseconds())
  }
}
```

Hackernews or Fakenews?

- Ask HN: Facerank — A/B test your Tinder pics.
- Facebook Run Code Safely
- Autocomplete Using Markov Chains
- Building a Conspiracy Theory of Everything
- Jupyter Notebooks on a Startup by SoftBank
- Delta Chat App on Android
- Elon Musk Can’t Help Himself
- 'Drinkable' Potato Chips: The End of Cyber Security?
- Show HN: GitHub Code Review with Emacs
- A Git-Based Social Credit Card

What’s next?

“Learn Go with tests” by Chris James (github: @quii)

“Advanced testing with Go” by Mitchell Hashimoto

Thank you!

https://github.com/prozz/goat