Testing Go Code

lessons learnt

Jakub Jarosz

Dublin Go Meetup - November 2022

today we will

- talk about testing
- look at code examples
- refactor
- look at some Go test tooling
- improve communication through tests

concurrent tests: t.Parallel

```
func TestRiversClient_GetsLatestWaterLevelReadingsOnValidPath(t *testing.T) {
        t.Parallel()
        ts := newTestServer("/geojson/latest", "testdata/latest_short.json", t)
        client := rivers.NewClient()
        client.BaseURL = ts.URL
…
}

func TestRiversClient_GetsMonthWaterLevelOnValidPath(t *testing.T) {
        t.Parallel()
        ts := newTestServer("/data/month", "testdata/month_01041_0001.csv", t)
        client := rivers.NewClient()
        client.BaseURL = ts.URL
…
}
```

failures: t.Error and t.Errorf

— — —

```
func TestArgsSuppliesCommandLineArgumentsAsInputToPipeOnePerLine(t *testing.T) {
	t.Parallel()
	…
	cmd := exec.Command(os.Args[0], "hello", "world")
	cmd.Env = append(os.Environ(), "SCRIPT_TEST=args")
	got, err := cmd.Output()
	if err != nil {
		t.Fatal(err)
	}
	want := "hello\nworld\n"
	if string(got) != want {
		t.Errorf("want %q, got %q", want, string(got))
	}
}
```

Source: Bitfield Consulting

failures: t.Error and t.Errorf

```
func TestExecForEach_ErrorsOnInvalidTemplateSyntax(t *testing.T) {
        t.Parallel()

        p := script.Echo("a\nb\nc\n").ExecForEach("{{invalid template syntax}}")
        p.Wait()

        if p.Error() == nil {
                t.Error("want error with invalid template syntax")
        }
}
```

Source: [Bitfield Consulting]

abandoning tests with t.Fatal

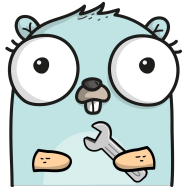
```
func TestArgsSuppliesCommandLineArgumentsAsInputToPipeOnePerLine(t *testing.T) {
	t.Parallel()
	…
	cmd := exec.Command(os.Args[0], "hello", "world")
	cmd.Env = append(os.Environ(), "SCRIPT_TEST=args")
	got, err := cmd.Output()
	if err != nil {
		t.Fatal(err)
	}
	want := "hello\nworld\n"
	if string(got) != want {
		t.Errorf("want %q, got %q", want, string(got))
	}
}
```

Source: Bitfield Consulting

writing debug output with t.Log

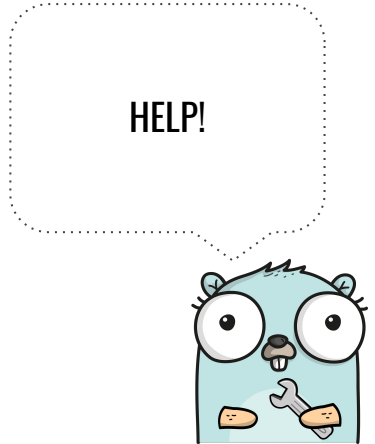
```
func TestLogLevelUSER(t *testing.T) {
	t.Log("Given the need to log DEV and USER messages.") {
		t.Log("\tWhen we set the logging level to USER.") {
			log.Init(&logdest, func() int { return log.USER }, log.Ldefault)
			resetLog()
			defer displayLog()
			...
			if logdest.String() == log1+log2 {
				t.Logf("\t\t%v : Should log the expected trace line.", Success)
			} else {
				t.Log("***>", logdest.String())
				t.Errorf("\t\t%v : Should log the expected trace line.", Failed)}
		}
	}
}
```

Source: [ardanlabs]



assistants: t.Helper

```
func newTestDB(stmtPopulateData string, t *testing.T) *sql.DB {
	t.Helper()

	db, err := sql.Open("sqlite3", ":memory:")
	if err != nil {
		t.Fatal(err)
	}
	…
	return db
}
```



assistants: t.Helper

```
func TestAdd_DoesNotAddDuplicateReadings(t *testing.T) {
	t.Parallel()

	db := newTestDB(stmtEmptyDB, t)
	readings := rivers.ReadingsRepo{
		Store: &rivers.SQLiteStore{
			DB: db,
		},
	}
	…
	if len(gotReadings) != 1 {
		t.Error("does not filter duplicates")
	}
}
```

cleanup resources: t.Cleanup

```
func newTestDB(stmtPopulateData string, t *testing.T) *sql.DB {
	t.Helper()
	db, err := sql.Open("sqlite3", ":memory:")
	if err != nil {
		t.Fatal(err)
	}
	...
	// Call the func to clean database after each test.
	// This way we don't need to pollute test logic with `defer`.
	t.Cleanup(func() {
		if _, err := db.Exec(`DROP TABLE waterlevel_readings`); err != nil {
			t.Fatalf("cleaning database: %v", err)
		}
	})
	return db
}
```

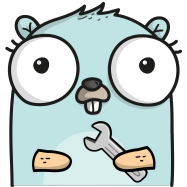
create and delete files: t.TempDir

```
func TestFileStore_AppendsDataToFile(t *testing.T) {
	t.Parallel()
	records := []rivers.StationWaterLevelReading{......}

	path := t.TempDir() + "/data_test.txt"

	s, _ := rivers.NewFileStore(path)
	s.Save(records)
	got, _ := os.ReadFile(path)
	want, _ := os.ReadFile("testdata/savedata_test.txt")
	if !cmp.Equal(want, got) {
		t.Error(cmp.Diff(want, got))
	}
```

source: rivers

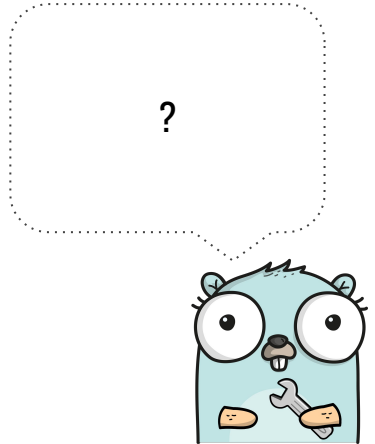
detecting useless implementations

> “Some tests almost seem determined to overlook any potential problems and merely confirm the prevailing supposition that the system works. I think this is the wrong attitude. **You want your tests to fail when the system is incorrect**, that’s the point.
>
> If a test can never fail, it’s not worth writing.”
>
> - John Arundel. “The Power of Go: Tests”



detecting useless implementations

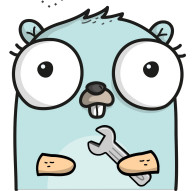
```
func GetMapKeyAsStringSlice(m map[string]string, key string, _ apiObject, delimiter string) ([]string, bool, error) {
	if str, exists := m[key]; exists {
		slice := strings.Split(str, delimiter)
		return slice, exists, nil
	}
	return nil, false, nil
}
...
if serverSnippets, exists, err := GetMapKeyAsStringSlice(a, b, c, d); exists {
	if err != nil {
		glog.Error(err)
	} else {
		cfgParams.ServerSnippets = serverSnippets
	}
}
```



detecting useless implementations

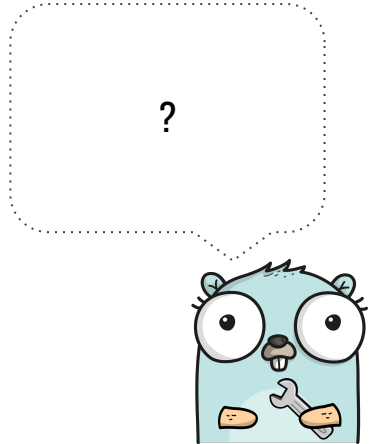
```
func TestGetMapKeyAsStringSlice(t *testing.T) {
        t.Parallel()
        slice, exists, err := GetMapKeyAsStringSlice(configMap.Data, "key", &configMap, ",")
        if err != nil {
                t.Errorf("Unexpected error: %v", err)
        }
        if !exists {
                t.Errorf("The key 'key' must exist in the configMap")
        }
        expected := []string{"1.String", "2.String", "3.String"}
        t.Log(expected)
        if !reflect.DeepEqual(expected, slice) {
                t.Errorf("Unexpected return value:\nGot: %#v\nExpected: %#v", slice, expected)
        }
}
```

?



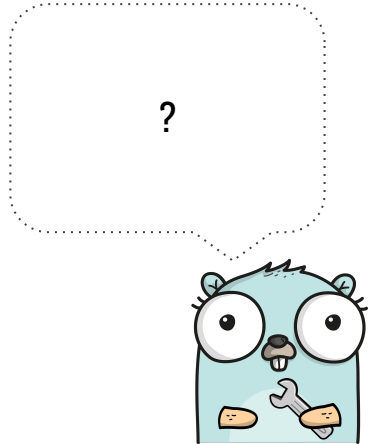
detecting useless implementations

```
func TestGetMapKeyAsStringSlice(t *testing.T) {
        t.Parallel()
        slice, exists, err := GetMapKeyAsStringSlice(configMap.Data, "key", &configMap, ",")
        if err != nil {
                t.Errorf("Unexpected error: %v", err)
        }
        if !exists {
                t.Errorf("The key 'key' must exist in the configMap")
        }
        expected := []string{"1.String", "2.String", "3.String"}
        t.Log(expected)
        if !reflect.DeepEqual(expected, slice) {
                t.Errorf("Unexpected return value:\nGot: %#v\nExpected: %#v", slice, expected)
        }
}
```



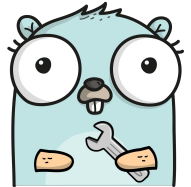
detecting useless implementations

```
func TestGetMapKeyAsStringSlice(t *testing.T) {
        t.Parallel()
        slice, exists, err := GetMapKeyAsStringSlice(configMap.Data, "key", &configMap, ",")
        if err != nil {
                t.Errorf("Unexpected error: %v", err)
        }
        if !exists {
                t.Errorf("The key 'key' must exist in the configMap")
        }
        expected := []string{"1.String", "2.String", "3.String"}
        t.Log(expected)
        if !reflect.DeepEqual(expected, slice) {
                t.Errorf("Unexpected return value:\nGot: %#v\nExpected: %#v", slice, expected)
        }
}
```



detecting useless implementations - refactor

```
func TestGetMapKeyAsStringSlice(t *testing.T) {
	t.Parallel()

	got, _ := GetMapKeyAsStringSlice(configMap.Data, "key", &configMap, ",")
	want := []string{"1.String", "2.String", "3.String"}
	if !cmp.Equal(want, got) {
		t.Errorf(cmp.Diff(want, got))
	}
}
```



comparisons: cmp.Equal and cmp.Diff

```
func TestGetMapKeyAsStringSlice(t *testing.T) {
        t.Parallel()
        …
        slice, exists, err := GetMapKeyAsStringSlice(configMap.Data, "key", &configMap, ",")
        if err != nil {
                t.Errorf("Unexpected error: %v", err)
        }
        want := []string{"1.String", "2.String", "3.String"}
        t.Log(want)
        if !reflect.DeepEqual(want, slice) {
                t.Errorf("Unexpected return value:\nGot: %#v\nExpected: %#v", slice, want)
        }
}
```

comparisons: cmp.Equal and cmp.Diff

```
func TestGetMapKeyAsStringSlice(t *testing.T) {
        t.Parallel()
        …
        slice, exists, err := GetMapKeyAsStringSlice(configMap.Data, "key", &configMap, ",")
        if err != nil {
                t.Errorf("Unexpected error: %v", err)
        }
        want := []string{"1.String", "2.String", "3.String"}
        t.Log(want)

        if !cmp.Equal(want, slice) {
                t.Errorf("Unexpected return value: \n%s", cmp.Diff(want, slice))
        }
}
```

comparisons: cmp.Equal and cmp.Diff - exclude fields

```
func TestGetPlace_RetrievesSingleGeoNameOnValidInput(t *testing.T) {
	client, _ := geonames.NewClient("DummyUser", geonames.WithBaseURL(ts.URL))
	got, _ := client.GetPlace(name, country, resultLimit)
	want := []geonames.Geoname{
		{
			Summary:   "Castlebar is the county town of County Mayo…",
			Position: geonames.Position{Lat:  53.8608, Long: -9.2988},
			CountryCode: "IE",
			Title:       "Castlebar",
		},
	}
	if !cmp.Equal(want, got, cmpopts.IgnoreFields(geonames.Geoname{}, "Summary", "Position")) {
		t.Errorf(cmp.Diff(want, got))
	}
}
```

Source: geonames

comparisons: cmp.Equal and cmp.Diff - test run

```
--- FAIL: TestGetPlace_RetrievesSingleGeoNameOnValidInput (0.00s)
    wikipedia_test.go:50:   []geonames.Geoname{
            {
                    Summary:   "Castlebar is the county town of County Mayo, Ireland. It is in t"...,
        -           Elevation: 42,
        +           Elevation: 41,
                    ... // 2 identical fields
        -           Title:     "Castlebarr",
        +           Title:     "Castlebar",
                    URL:       "en.wikipedia.org/wiki/Castlebar",
            },
          }
FAIL
exit status 1
FAIL   github.com/qba73/geonames 0.279s
```

comparisons: cmp.Equal and cmp.Diff - refactor

```
...
for _, test := range tests {
	test.expectedPolicyEx.Obj = test.policy

	policyEx, err := createAppProtectPolicyEx(test.policy)
	if (err != nil) != test.wantErr {
		t.Errorf("createAppProtectPolicyEx() returned %v, for the case of %s", err, test.msg)
	}

	if diff := cmp.Diff(test.expectedPolicyEx, policyEx); diff != "" {
		t.Errorf("createAppProtectPolicyEx() %q returned unexpected result (-want +got):\n%s", test.msg, diff)
	}
}
```

comparisons: cmp.Equal and cmp.Diff - refactor

```
...
for _, test := range tests {
	test.expectedPolicyEx.Obj = test.policy

	policyEx, err := createAppProtectPolicyEx(test.policy)
	if (err != nil) != test.wantErr {
		t.Errorf("createAppProtectPolicyEx() returned %v, for the case of %s", err, test.msg)
	}

	if !cmp.Equal(test.expectedPolicyEx, policyEx) {
		t.Errorf("createAppProtectPolicyEx() \n%s", cmp.Diff(test.expectedPolicyEx, policyEx))
	}
}
```

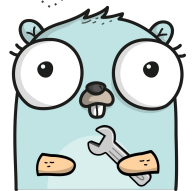
comparisons: cmp.Equal and cmp.Diff - refactor

```
...
for _, test := range tests {
	test.expectedPolicyEx.Obj = test.policy

	gotPolicy, err := createAppProtectPolicyEx(test.policy)
	if (err != nil) != test.wantErr {
		t.Errorf("createAppProtectPolicyEx() returned %v, for the case of %s", err, test.msg)
	}

	if !cmp.Equal(test.wantPolicy, gotPolicy) {
		t.Errorf(“createAppProtectPolicyEx() \n%s”, cmp.Diff(test.wantPolicy, gotPolicy))
	}
}
```

comparisons: cmp.Equal and cmp.Diff - refactor

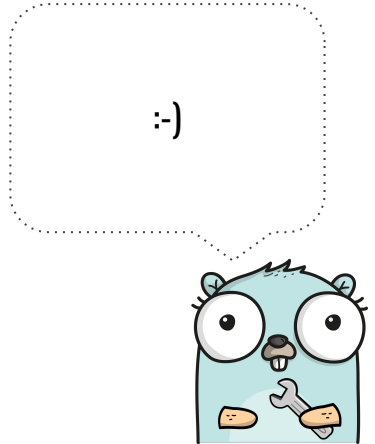
```
...
for _, test := range tests {
	test.expectedPolicyEx.Obj = test.policy

	gotPolicy, err := createAppProtectPolicyEx(test.policy)
	if (err != nil) != test.wantErr {
		t.Errorf("createAppProtectPolicyEx() returned %v, for the case of %s", err, test.msg)
	}

	if !cmp.Equal(test.wantPolicy, gotPolicy) {
		t.Errorf("createAppProtectPolicyEx() \n%s", cmp.Diff(test.wantPolicy, gotPolicy))
	}
}
```

?



comparisons: cmp.Equal and cmp.Diff - simplify, simplify...

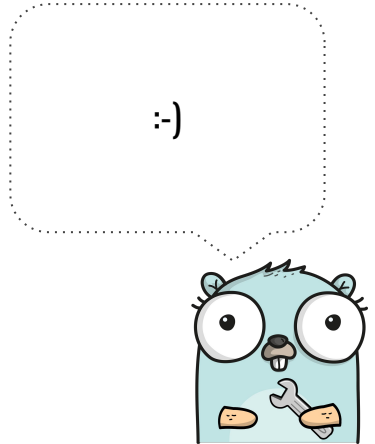
```
...
for _, tc := range tt {
	got, err := createAppProtectPolicyEx(tc.policy)
	if err != nil {
		t.Fatalf("createAppProtectPolicyEx() got error: %v", err)
	}
	if !cmp.Equal(tc.want, got) {
		t.Errorf(“createAppProtectPolicyEx() \n%s”, cmp.Diff(tc.want, got))
	}
}
```



comparisons: cmp.Equal and cmp.Diff - simplify, simplify...

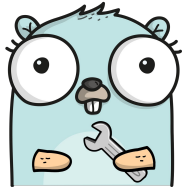
— — —

```
...
for _, tc := range tt {
	t.Run(tc.name, func(t *testing.T){
		got, err := createAppProtectPolicyEx(tc.policy)
		if err != nil {
			t.Fatal(err)
		}
		if !cmp.Equal(tc.want, got) {
			t.Error(cmp.Diff(tc.want, got))
		}
	)
}
```



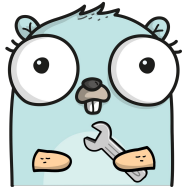
what are we really testing here?

```
func TestAddOrUpdateIngress(t *testing.T) {
	cnf, err := createTestConfigurator()
	if err != nil {
		t.Errorf("Failed to create a test configurator: %v", err)
	}
	ingress := createCafeIngressEx()
	warnings, err := cnf.AddOrUpdateIngress(&ingress)
	if err != nil {
		t.Errorf("AddOrUpdateIngress returned:  \n%v, but expected: \n%v", err, nil)
	}
	if len(warnings) != 0 {
		t.Errorf("AddOrUpdateIngress returned warnings: %v", warnings)
	}
	…
}
```



what are we really testing here?

```
func TestAddOrUpdateIngress(t *testing.T) {
	cnf, err := createTestConfigurator()
	if err != nil {
		t.Errorf("Failed to create a test configurator: %v", err)
	}
	ingress := createCafeIngressEx()
	warnings, err := cnf.AddOrUpdateIngress(&ingress)
	if err != nil {
		t.Errorf("AddOrUpdateIngress returned:  \n%v, but expected: \n%v", err, nil)
	}
	if len(warnings) != 0 {
		t.Errorf("AddOrUpdateIngress returned warnings: %v", warnings)
	}
	…
}
```



what are we really testing here? - t.Parallel

```
func TestAddOrUpdateIngress(t *testing.T) {
	t.Parallel()
	cnf, err := createTestConfigurator()
	if err != nil {
		t.Errorf("Failed to create a test configurator: %v", err)
	}
	ingress := createCafeIngressEx()
	warnings, err := cnf.AddOrUpdateIngress(&ingress)
	if err != nil {
		t.Errorf("AddOrUpdateIngress returned:  \n%v, but expected: \n%v", err, nil)
	}
	if len(warnings) != 0 {
		t.Errorf("AddOrUpdateIngress returned warnings: %v", warnings)
	}
	…
}
```

what are we really testing here? - t.Fatal

```
func TestAddOrUpdateIngress(t *testing.T) {
	t.Parallel()
	cnf, err := createTestConfigurator()
	if err != nil {
		t.Fatal(err)
	}
	ingress := createCafeIngressEx()
	warnings, err := cnf.AddOrUpdateIngress(&ingress)
	if err != nil {
		t.Errorf("AddOrUpdateIngress returned:  \n%v, but expected: \n%v", err, nil)
	}
	if len(warnings) != 0 {
		t.Errorf("AddOrUpdateIngress returned warnings: %v", warnings)
	}
	…
}
```

what are we really testing here?- t.Helper

```
func createTestConfigurator() (*Configurator, error) {
	templateExecutor, err := version1.NewTemplateExecutor("version1/abc.tmpl", "version1/def.tmpl")
	if err != nil {
		return nil, err
	}
	manager := nginx.NewFakeManager("/etc/nginx")
	cnf, err := NewConfigurator(manager, createTestStaticConfigParams(), NewDefaultConfigParams(false),
				templateExecutor, templateExecutorV2, false, false, nil, false, nil, false)
	if err != nil {
		return nil, err
	}
	cnf.isReloadsEnabled = true
	return cnf, nil
}
```

what are we really testing here? - t.Fatal

```
func createTestConfigurator(t *testing.T) (*Configurator, error) {
	templateExecutor, err := version1.NewTemplateExecutor("version1/abc.tmpl", "version1/def.tmpl")
	if err != nil {
		t.Fatal(err)
	}
	manager := nginx.NewFakeManager("/etc/nginx")
	cnf, err := NewConfigurator(manager, createTestStaticConfigParams(), NewDefaultConfigParams(false),
				templateExecutor, templateExecutorV2, false, false, nil, false, nil, false)
	if err != nil {
		t.Fatal(err)
	}
	cnf.isReloadsEnabled = true
	return cnf, nil
}
```

what are we really testing here? - t.Helper

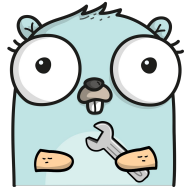
```
func createTestConfigurator(t *testing.T) *Configurator {
	t.Helper()
	templateExecutor, err := version1.NewTemplateExecutor("version1/abc.tmpl", "version1/def.tmpl")
	if err != nil {
		t.Fatal(err)
	}
	manager := nginx.NewFakeManager("/etc/nginx")
	cnf, err := NewConfigurator(manager, createTestStaticConfigParams(), NewDefaultConfigParams(false),
				templateExecutor, templateExecutorV2, false, false, nil, false, nil, false)
	if err != nil {
		t.Fatal(err)
	}
	cnf.isReloadsEnabled = true
	return cnf
}
```

what are we really testing here?- t.Helper

```
func TestAddOrUpdateIngress(t *testing.T) {
	t.Parallel()
	cnf := createTestConfigurator(t)

	ingress := createCafeIngressEx()
	warnings, err := cnf.AddOrUpdateIngress(&ingress)
	if err != nil {
		t.Errorf("AddOrUpdateIngress returned:  \n%v, but expected: \n%v", err, nil)
	}
	if len(warnings) != 0 {
		t.Errorf("AddOrUpdateIngress returned warnings: %v", warnings)
	}
	…
}
```

what are we really testing here? - t.Errorf

```
func TestAddOrUpdateIngress(t *testing.T) {
	t.Parallel()
	cnf := createTestConfigurator(t)

	ingress := createCafeIngressEx()
	warnings, err := cnf.AddOrUpdateIngress(&ingress)
	if err != nil {
		t.Errorf("AddOrUpdateIngress returned:  \n%v, but expected: \n%v", err, nil)
	}
	if len(warnings) != 0 {
		t.Errorf("AddOrUpdateIngress returned warnings: %v", warnings)
	}
	…
}
```



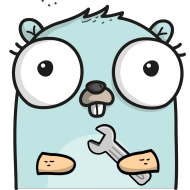
what are we really testing here? - t.Fatal

```
func TestAddOrUpdateIngress(t *testing.T) {
	t.Parallel()
	cnf := createTestConfigurator(t)

	ingress := createCafeIngressEx()
	warnings, err := cnf.AddOrUpdateIngress(&ingress)
	if err != nil {
		t.Fatal(err)
	}
	if len(warnings) != 0 {
		t.Errorf("AddOrUpdateIngress returned warnings: %v", warnings)
	}
	...
}
```

what are we really testing here? - naming tests

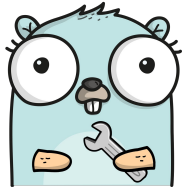
```
func TestAddOrUpdateIngress_ReturnsNoWarningsOnValidInput(t *testing.T) {
        t.Parallel()
        cnf := createTestConfigurator(t)
        ingress := createCafeIngressEx()
        warnings, err := cnf.AddOrUpdateIngress(&ingress)
        if err != nil {
                t.Fatal(err)
        }
        want, got := 0, len(warnings)
        if want != got {
                t.Errorf("want %d, got %d", want, got)
        }
}
```

APPROVED



what are we really testing here? API libraries

```
func NewNginxClientWithVersion(httpClient *http.Client, apiEndpoint string, version int) (*NginxClient, error) {
	if !versionSupported(version) {
		return nil, fmt.Errorf("API version %v is not supported by the client", version)
	}
	versions, err := getAPIVersions(httpClient, apiEndpoint)
	if err != nil {
		return nil, fmt.Errorf("error accessing the API: %w", err)
	}
	…
	return &NginxClient{
		apiEndpoint: apiEndpoint,
		httpClient:  httpClient,
		version:     version,
	}, nil
}
```

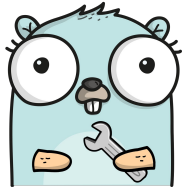


what are we really testing here? API libraries

```
func NewNginxClientWithVersion(httpClient *http.Client, apiEndpoint string, version int) (*NginxClient, error) {
	if !versionSupported(version) {
		return nil, fmt.Errorf("API version %v is not supported by the client", version)
	}
	versions, err := getAPIVersions(httpClient, apiEndpoint)
	if err != nil {
		return nil, fmt.Errorf("error accessing the API: %w", err)
	}
	...
	return &NginxClient{
		apiEndpoint: apiEndpoint,
		httpClient:  httpClient,
		version:     version,
	}, nil
}
```

what are we really testing here?

http://testing.gobanana.co.uk/?p=743

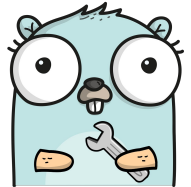


what are we really testing here?

```
func NewNginxClientWithVersion(httpClient *http.Client, apiEndpoint string, version int) (*NginxClient, error) {
	if !versionSupported(version) {
		return nil, fmt.Errorf("API version %v is not supported by the client", version)
	}
	versions, err := getAPIVersions(httpClient, apiEndpoint)
	if err != nil {
		return nil, fmt.Errorf("error accessing the API: %w", err)
	}
	...
	return &NginxClient{
		apiEndpoint: apiEndpoint,
		httpClient:  httpClient,
		version:     version,
	}, nil
}
```

what are we really testing here? - sensible defaults

```
package main
import “http”

func main() {
	resp, err := http.Get("http://www.google.com/robots.txt")
	if err != nil {
	// handle error
	}
}
```



sensible defaults - Go std library

```
package http

…
// To make a request with custom headers, use NewRequest and
// DefaultClient.Do.
//
// To make a request with a specified context.Context, use NewRequestWithContext
// and DefaultClient.Do.
func Get(url string) (resp *Response, err error) {
        return DefaultClient.Get(url)
}

…
```

sensible defaults - Go std library

```
package http

…
// DefaultClient is the default Client and is used by Get, Head, and Post.
var DefaultClient = &Client{}
…
```

sensible defaults - refactoring

```
func NewNginxClientWithVersion(httpClient *http.Client, apiEndpoint string, version int) (*NginxClient, error) {
	if !versionSupported(version) {
		return nil, fmt.Errorf("API version %v is not supported by the client", version)
	}
	versions, err := getAPIVersions(httpClient, apiEndpoint)
	if err != nil {
		return nil, fmt.Errorf("error accessing the API: %w", err)
	}
	...
	return &NginxClient{
		apiEndpoint: apiEndpoint,
		httpClient:  httpClient,
		version:     version,
	}, nil
}
```

sensible defaults - reduce paperwork

```
func NewNginxClientWithVersion(apiEndpoint string) *NginxClient {
        return &NginxClient{
                apiEndpoint: apiEndpoint,
                httpClient:  http.DefaulClient,
                version:     defaultVersion,
        }
}
```

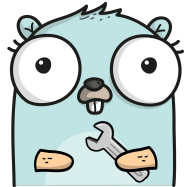
sensible defaults - functional options

```
type option func(*NginxClient) error

func WithHTTPClient(h *http.Client) option {
	return func(nc *NginxClient) error {
		if h == nil {
			return errors.New("nil http client")
		}
		nc.httpClient = h
		return nil
	}
}
```

functional options

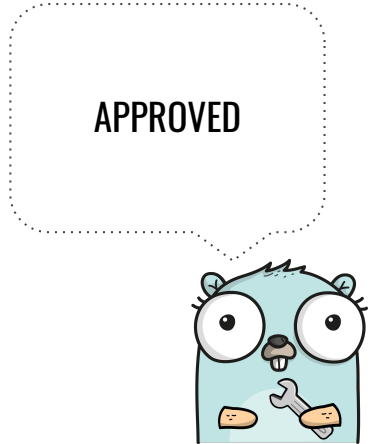
sensible defaults - functional options

```
type option func(*NginxClient) error

func WithAPIVersion(v int) option {
	return func(nc *NginxClient) error {
		switch v {
			case 5,6,7,8:
				nc.version = v
				return nil
			default:
		}
		return fmt.Errorf(“version %d not supported”, v)
	}
}
```



sensible defaults - functional options

```
func NewNginxClient(apiEndpoint string, opts ...option) (*NginxClient, error) {
	nc := NginxClient{
		apiEndpoint: apiEndpoint,
		httpClient:  http.DefaultClient,
		version:     defaultVersion,
	}
	for _, opt := range opts {
		if err := opt(&nc); err != nil {
			return nil, err
		}
	}
	return &nc, nil
}
```



sensible defaults - testing the client

```
package nginx_test

func TestNewNGINXClient_FailsOnInvalidAPIVersion(t *testing.T) {
        t.Parallel()
        _, err := nginx.NewNginxClient("http://localhost:8080", nginx.WithAPIVersion(10))
        if err == nil {
                t.Fatal("want error on invalid version, got nil")
        }
}
```

sensible defaults - table tests - negative cases

```
package nginx_test

func TestNewNGINXClient_FailsOnInvalidAPIVersions(t *testing.T) {
	t.Parallel()
	cases := []int{-1,0,1,4,9,10}
	for _, tc := range cases {
		_, err := nginx.NewNginxClient("http://localhost:8080", nginx.WithAPIVersion(tc))
		if err == nil {
			t.Fatalf("want error on invalid version %q, got nil", tc)
		}
	}
}
```

sensible defaults - table tests - positive cases

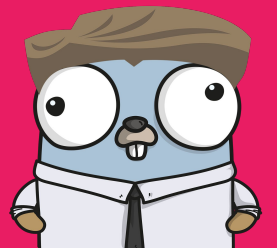
```
package nginx_test

func TestNewNGINXClient_SucceedOnValidAPIVersions(t *testing.T) {
	t.Parallel()
	cases := []int{5,6,7,8}
	for _, tc := range cases {
		_, err := nginx.NewNginxClient("http://localhost:8080", nginx.WithAPIVersion(tc))
		if err != nil {
			t.Fatalf("got error on valid API version %q", tc)
		}
	}
}
```

table tests - separate positive & negative cases

```
...
for _, tt := range tests {
	t.Run(tt.name, func(t *testing.T) {
		got, err := parsePositiveDuration(tt.testInput)
		if (err != nil) != tt.wantErr {
			t.Errorf("parsePositiveDuration() error = %v, wantErr %v", err, tt.wantErr)
			return
		}
		if !reflect.DeepEqual(got, tt.want) {
			t.Errorf("parsePositiveDuration() = %v, want %v", got, tt.want)
		}
	})
...
```

Source

gotestdox



```
go install github.com/bitfield/gotestdox/cmd/gotestdox@latest
```

gotestdox

> “What to call your test is easy: **it’s a sentence describing the next behaviour in which you are interested**. How much to test becomes moot: you can only describe so much behaviour in a single sentence.
>
> — Dan North, “Introducing BDD”

are we testing all important behaviours?

Test names should be ACE:

they should include **Action**, **Condition**, and **Expectation**.

gotestdox - are we testing all important behaviours?

Test names should be ACE:

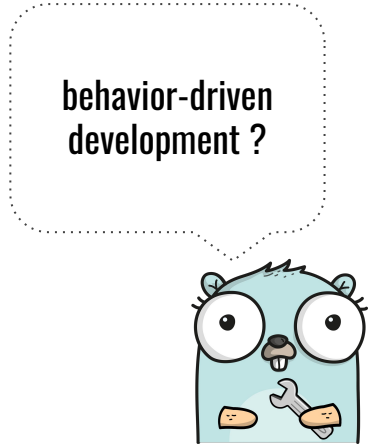
they should include Action, Condition, and Expectation.

func Valid()

Action: calling Valid

Condition: with valid input

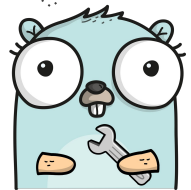
Expectation: returns true



gotestdox - don’t worry about long test names

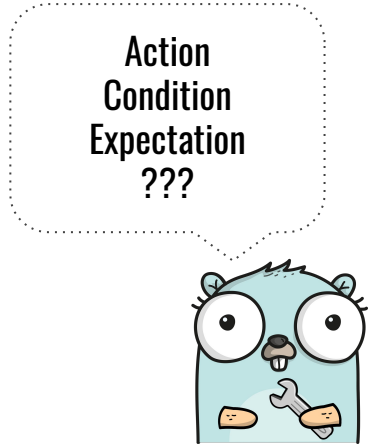
```
func TestAddOrUpdateIngress_ReturnsNoWarningsOnValidInput(t *testing.T) {
        t.Parallel()
        cnf := createTestConfigurator(t)
        ingress := createCafeIngressEx()
        warnings, err := cnf.AddOrUpdateIngress(&ingress)
        if err != nil {
              t.Fatal(err)
        }
        want, got := 0, len(warnings)
        if want != got {
              t.Errorf(“want %d, got %d”, want, got)
        }
        …
}
```

Action
Condition
Expectation



gotestdox - are we testing all important behaviours?

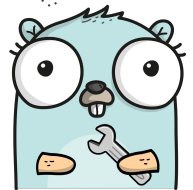
```
➜  client git:(main) gotestdox
github.com/nginxinc/nginx-plus-go-client/client:
 ✔ Add port to server (0.00s)
 ✔ Determine updates (0.00s)
 ✔ Have same parameters (0.00s)
 ✔ Have same parameters for stream (0.00s)
 ✔ Stream determine updates (0.00s)
```



gotestdox - are we testing all important behaviours?

```
➜  ngx git:(main) ✗ gotestdox
github.com/qba73/ngx:
 ✔ CheckServerUpdates is valid on valid input (0.00s)
 ✔ CheckStreamServerUpdates is valid on valid input (0.00s)
 ✔ GetNGINXInfo returns info about running NGINX instance (0.00s)
 ✔ GetNGINXStatus errors on invalid request params (0.00s)
 ✔ GetNGINXStatus returns status info on valid fields (0.00s)
 ✔ GetNGINXStatus uses valid request path on valid request params (0.00s)
 ✔ NewClient fails on invalid base URL (0.00s)
 ✔ NewClient fails on invalid version (0.00s)
 ✔ ServerAddress is valid on valid input with address and without port (0.00s)
 ✔ ServerAddress is valid on valid input with host and port (0.00s)
 ✔ ServerAddress is valid on valid input with unix socket (0.00s)
 ✔ ...
 ✔ UpstreamStreamServersConfig is valid on valid input (0.00s)
```

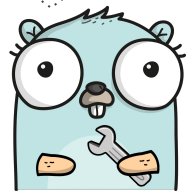
Action
Condition
Expectation



gotestdox - are we testing all important behaviours?

```
➜  meteo git:(master) ✗ gotestdox
github.com/qba73/meteo:
 ✔ Client formats weather info on valid input (0.00s)
 ✔ Client reads current weather on valid input (0.00s)
 ✔ Client requests weather with valid path and params (0.00s)
```

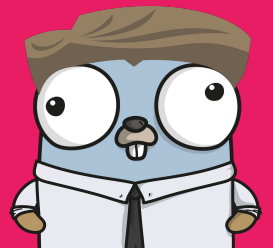
Action
Condition
Expectation



“gotestdox” summary

- **Forces to think**
- Helps with BDD approach
- Helps to build mental models
- Helps with design thinking
- Helps answering question(s):
  “What are we really testing?”

testscript



github.com/rogpeppe/go-internal/testscript

testscript - automate testing binaries, the right (Go) way!

http://blog.chriss-baumann.de/2010/07/26/because-life-is-too-short-for-manual-testing

testscript - repository structure

```
➜  tree
.
└── godublin
    ├── go.mod
    ├── go.sum
    ├── godub.go
    ├── godub_test.go
    └── testdata
        └── script
            ├── coverage.txtar
            └── hello.txtar
```

testscript examples:

https://github.com/qba73/dublin-go-meetup

“testscript” demo

- testscript DSL
- assertions
- stdout, stderr
- binary
- **cli tools**
- golden files

github.com/qba73/dublin-go-meetup

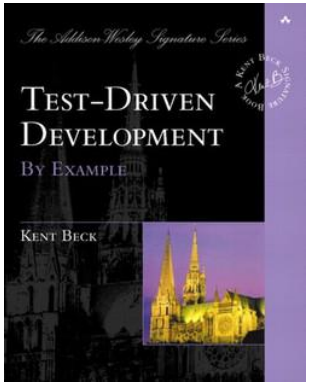
“testscript” summary

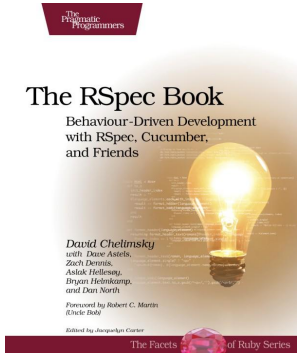
- Excellent for testing binaries
- Excellent for testing CLI tools
- Runs along other tests
- Saves time
- Builds confidence
- It’s derived directly from the code used to test Go tool itself!

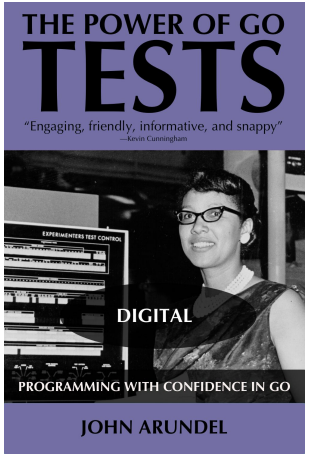
resources

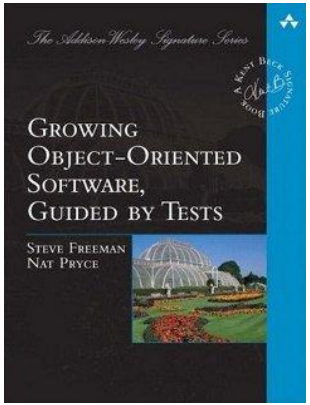
- bitfieldconsulting.com/books
- github.com/bitfield/gotestdox
- pkg.go.dev/github.com/rogpeppe/go-internal/testscript
- github.com/mvdan/go-internal
- pkg.go.dev/github.com/google/go-cmp/cmp

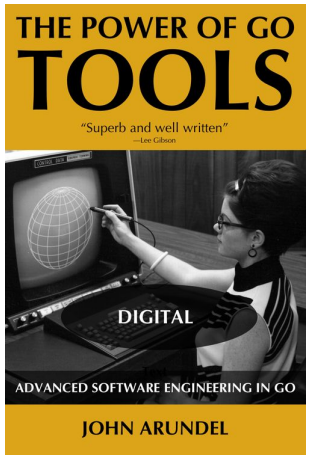
resources


The Addison Wesley Signature Series
Test-Driven Development
By Example
Kent Beck


The Pragmatic Programmers
The RSpec Book
Behaviour-Driven Development with RSpec, Cucumber, and Friends
David Chelimsky
with Dave Astels, Zach Dennis, Aslak Hellesøy, Bryan Helmkamp, and Dan North
Foreword by Robert C. Martin (Uncle Bob)
Edited by Jacquelyn Carter
The Facets of Ruby Series


THE POWER OF GO
TESTS
"Engaging, friendly, informative, and snappy"
—Kevin Cunningham
DIGITAL
PROGRAMMING WITH CONFIDENCE IN GO
JOHN ARUNDEL


The Addison Wesley Signature Series
Growing Object-Oriented Software, Guided by Tests
Steve Freeman
Nat Pryce


THE POWER OF GO
TOOLS
"Superb and well written"
—Lee Gibson
DIGITAL
ADVANCED SOFTWARE ENGINEERING IN GO
JOHN ARUNDEL

Thank You!

Jakub Jarosz

github, twitter @qba73 | linkedin.com/in/jakubjarosz

gopherize.me

Artwork by Ashley McNamara
inspired by Renee French
Web app by Mat Ryer