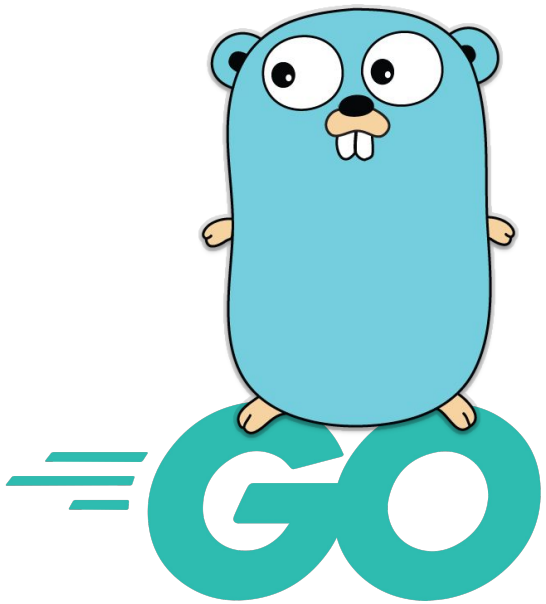
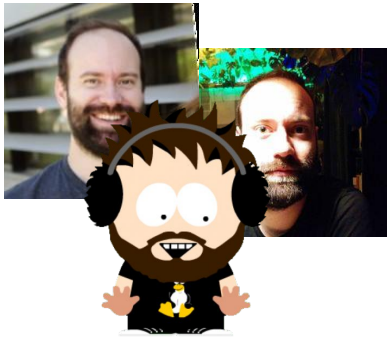
My “Decade” With Go

My experiences,
some ups & downs,
(maybe) some tips & tricks



Evelyn Haslinger eh@symflower.com
Markus Zimmermann mz@symflower.com

Who Am I?



- Markus Zimmermann https://github.com/zimmski
- Writing software since primary school
- Work (mainly “project consulting”)
  - Lots of “enterprise” applications, web services, tooling, software infrastructure, distributed apps and clustering
  - Now: Software testing and verification @Symflower
- Using Go since ...
  - @freetime: ~ public release (since Dec. holidays of 2009) (on & off)
  - @work (for every project, if possible): ~1.1 release (~2013) (fulltime)

Why Did I Start Using Go?

- Curiosity, I enjoy new languages
- Some Go “founders” are legendary
  - Ken Thompson and Rob Pike… Did you hear about UNIX? UTF-8? RegEx? (Robert Griesemer is really cool too)
- The announcement was really promising
  - Development speed of dynamic languages
  - Performance of compiled languages
  - Easy multi-processing, maintaining, <BIG words>
  - **Google is pushing it internally**

Projects Where I Am Using Go

- Open Source
  - go-mutesting, go-flags, go-clang, golint, errcheck, tavor, go-diff, zimmski/osutil, ...
- Work
  - Lots of rewrites (clarity, performance, parallelization), web/distributed services and “batch”jobs (easy)
  - **Now: Own startup “Symflower”**

What Is Symflower?

- **Generate perfect unit-tests fully-automatically**
- We are using Go for almost everything
- Distributed services with distinct components
- We support multiple languages
  - Multiple toolsets, parsers, frameworks, …
- We use a lot of math and …stuff… to find values
  - This is the most challenging part: also, mostly in Go

Example: Find the Problem!

```
func match(s1, s2 []byte) bool {
    for i := 0; i < len(s1); i++ {
        c1 := s1[i]
        c2 := s2[i]
        if c1 != c2 {
            c1 |= 'a' - 'A'
            c2 |= 'a' - 'A'
            if c1 != c2 || c1 < 'a' || c1 > 'z' {
                return false
            }
        }
    }
    return true
}
```

Code from https://golang.org/src/time/format.go

Found the Problem!

```
func match(s1, s2 []byte) bool {
    for i := 0; i < len(s1); i++ {
        c1 := s1[i]
        c2 := s2[i]
        if c1 != c2 {
            c1 |= 'a' - 'A'
            c2 |= 'a' - 'A'
            if c1 != c2 || c1 < 'a' || c1 > 'z' {
                return false
            }
        }
    }
    return true
}
```

Index out of range!

Generated Unit Tests

```go
func TestMatch117(t *testing.T) {
    var s1 []byte = []byte{'\x00'}
    var s2 []byte = nil

    match(s1, s2) // Panic!
}

func TestMatch119(t *testing.T) {
    var s1 []byte = []byte{'\x00'}
    var s2 []byte = []byte{'\x00'}

    actual := match(s1, s2)

    var expected bool = true
    assert.Equal(t, expected, actual)
}
```

```go
func TestMatch123(t *testing.T) {
    var s1 []byte = []byte{'c'}
    var s2 []byte = []byte{'C'}

    actual := match(s1, s2)

    var expected bool = true
    assert.Equal(t, expected, actual)
}
```

.....

Full MC/DC and problem coverage

If you find that interesting: talk to me or write to hello@symflower.com

My Experiences

Why I was and still am excited about Go

Why Am I Using Go Now?

- Consulting work: different … per project/**person**
  - Languages, code/API style, conventions, …
  - **Go: (almost) just one of everything**
  - No (performance) rewrites needed anymore
- **Really easy to learn / very small language**
- Enough high level/modern features
- Fast
- Maintainable and handles huge projects

Tools!, Tools!!, Tools!!!

- Everything that you need is usually built-in
  - Standard (STD) packages are amazing
  - $ go get
  - $ go install (build system is included)
  - $ go fmt (Picked up by almost all other languages)
  - $ go test (includes static analysis)
  - $ go … -race (Race detector)
  - $ go tool pprof / $ go tool trace

Problems With Using Go Fulltime

- Letting go of bias/conventions/… was **hard**
- Worked for almost everything I did
  - Services, Web, Clustering, Batch, … ALMOST NO PROBLEMS
  - Lots of things in STD but some missing (remember I used Go 1.1)
- Still: language wars with others continue…
  - Go is often not taken seriously by others
    “It is missing XYZ!!!!!!”, “ABC is better because DEF!”
  - Often those people never (really) used Go, so say
    “Its fine, I do it the Go way.”

```
if err != nil { … }
```

```
if err != nil { … }
…
if err != nil { … }
```

if err != nil { … } if err != nil { … } if err !=
nil { … } if err != nil { … } if err != nil { … }
if err != nil { … } if err != nil { … } if err !=
nil { … } if err != nil { … } if err != nil { … }
if err != nil { … } if err != nil { … } if err !=
nil { … } if err != nil { … } if err != nil { … }
if err != nil { … } if err != nil { … } if err !=
nil { … } if err != nil { … } if err != nil { … }
if err != nil { … } if err != nil { … } if err !=
nil { … } if err != nil { … } if err != nil { … }

EMBRACE Go's error handling

- First: annoying, Now: I am writing better code
- Go's error handling is **explicit**
  - Often: try { <1million LoC> } catch { <rethrow> }
- **Add new context to the error**
- Do not misuse panic&recover
- They are working on better ways ("Go 2")
  - Error handling
  - Error values

Generics?

- Obviously some things do not work that well
  - E.g. common data structures, operators, …
  - Performance drops with interfaces
- But… **DO NOT** use empty interfaces (`{}interface`)
  - You are loosing type safety
  - Use interfaces instead! … or ...
  - If you really need generics: use a generator for now
- Also, they are working on generics (“Go 2”)

Explicit interfaces?

- The “implicit” interface design of Go is: **amazing**
  - No “import hell”
  - Do interfaces “on the fly”
    - Just implement a new interface without changing code
    - No subclassing/`implements`-hell
- But… explicit interfaces?!
  - If you really need it or make sure:

```
var _ Animal = (*Dog)(nil)
```

How to do OOP (object-oriented programming)

- Embrace two things
  - Interfaces
  - Composition
- What about …
  - Method overloading?
  - Operator overloading?
  - …
  - “Do it the Go way”: Think about the API e.g. function names

Doing Tests?

- Test framework is officially included in the STD
  - Subtests, parallel tests, coverage, … all in STD!
  - No real discussion about test tooling but…
  - Assert-Framework? I like stretchr/testify, you may like something else
- Go-projects tend to have lots of automated tests
  - Finally writing tests is not a discussion
  - Write own tests == design better APIs/code
- Write your own test “frameworks”/functions
  - E.g. https://golang.org/pkg/testing/#B.Helper

Table-driven Tests (usually)

```
var tests = []struct {
    name string
    word string
    want []string
}{
    {"empty input string", "", []string{}},
    {"two anagrams", "Protectionism", []string{"Cite no imports", "Nice to
imports"}},
    {"input with space", "Real fun", []string{"funeral"}},
}

func TestFindAnagrams(t *testing.T) {
    for _, tt := range tests {
        t.Run(tt.name, func(t *testing.T) {
            got := FindAnagrams(tt.word)
            if got != tt.want {
                t.Errorf("FindAnagrams(%s) got %v, want %v", tt.word, got, tt.want)
            }
        })
    }
}
```

Code from https://medium.com/@pliutau/table-driven-tests-in-go-5d7e230681da

Table-driven Tests (usually)

```
var tests = []struct {
    name string
    word string
    want []string
}{
    {"empty input string", "", []string{}},
    {"two anagrams", "Protectionism", []string{"Cite no imports", "Nice to
imports"}},
    {"input with space", "Real fun", []string{"funeral"}},
}

func TestFindAnagrams(t *testing.T) {
    for _, tt := range tests {
        t.Run(tt.name, func(t *testing.T) {
            got := FindAnagrams(tt.word)
            if got != tt.want {
                t.Errorf("FindAnagrams(%s) got %v, want %v", tt.word, got, tt.want)
            }
        })
    }
}
```

This does not scale with more fields

Stacktrace would not tell us which test case failed. You have to look it up in the report and then find the name in the table

Table-driven Tests (Symflower)

```
func TestFindAnagrams(t *testing.T) {
    type testCase struct {
        name string // This is the name of the test case.

        word string // This is the group of fields for the input.

        want []string // This is the group of fields for checking the output.
    }

    validate := func(t *testing.T, tc *testCase)
        t.Run(name, func(t *testing.T) {
            got := FindAnagrams(tc.word)
            if got != tc.want {
                t.Errorf("FindAnagrams(%s) got %v, want %v", tc.word, got, tc.want)
            }
        })
    }

    validate(t, &testCase{
        name: "two anagrams",

        word: "Protectionism",

        want: []string{"Cite no imports", "Nice to imports"},
    })
```

Always the same groups

Explicit names are human-readable

ALWAYS the correct stacktrace

Write your own tools

- IMHO Go wants you to write your own tooling
- E.g.
  - Static analysis is easy (use go/ast, go/parser, go/types…)
  - Generate certain code (e.g. use `go generate` or go/ast)
  - Refactoring (`gofmt -r`, `eq`, your own? -> go/ast)
  - Dynamic analysis: not that easy
    - E.g. look at `go tool cover`
    - Uses code generation

Maintain Big Projects

- Tools help (especially own static analysis rules)
  - Define your rules, check them automatically
- Go's many conventions and idioms help
  - One … for almost everything
- "No cyclic package dependencies" help
  - You are forced to restructure packages
- Monorepo helps (has lots of not-Go-specific advantages)
  - Vendoring support (now Go modules) helps
- "Promise of compatibility" HELPS

Some Tips for Starting with Go

- Do not be biased
  - Do not compare things to other languages
  - Do things the “Go” way and embrace them
- Look for open source projects and help out
  - The only way to learn something is to dive into it
  - Google projects are usually of high quality
  - Code reviews will help you learn a lot

Go, take a look

symflower
AUTOMATING QUALITY ASSURANCE

https://symflower.com/en/jobs/learning-resources/

Go is simple, Go is fast, Go is safe

Evelyn Haslinger eh@symflower.com
Markus Zimmermann mz@symflower.com

symflower

AUTOMATING QUALITY ASSURANCE

Evelyn Haslinger eh@symflower.com
Markus Zimmermann mz@symflower.com