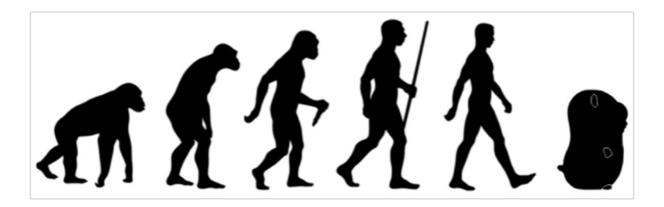
# The Evolution of Go - Robert Griesemer (2015)

GopherCon 2015: Robert Griesemer - The Evolution of Go - YouTube



Designing a **good** programming language is hard. This should not be called *software*.

C has too many complexities with too many differentiations that can all require potential real pages of description when it comes to problem solving a bug.

Language bears similarities to Oberon-2, structurally. It was definitely an influence on the design of Go.

## **Starting Points**

- Clear goal: A better language for Google's uses
- Personal desire: A clean, small, compiled language with modern features
- Clear about what was wrong: Complexity, missing concurrency support, lack of scalability, insane build times.

# **Guiding Principles**

- Simplicity, safety, and readability are paramount
- Striving for orthogonality in design
- Minimal: One way to write a piece of code
- It's about **expressing algorithms**, not the type system
- Collective unconscious history of programming languages

Things of interest should be easy; even if that means not everything is possible.

## Language Design Literature is Sparse

- "Hints on Programming Language Design" (C.A.R. Hoare, 1973)
- "Everything you always wanted to know about programming languages but were afraid to ask" (C.A.R Hoare, 1978)

### Many Day One Ideas Made It Into Go

- Syntax
- Expressions
- Explicitly sized basic types, rune type, no implicit conversions
- Packages and imports (C is just now getting these, 46 years later)
- Methods with explicit receiver parameter
- Interfaces
- Understanding that we would somehow add concurrency support based on Rob's Pike's previous work.

### Most Ideas Come From Previous Ideas

There's nothing new in Go!

They are missing the point:

The task of the programming language designer "is consolidation not innovation"

→ C.A.Hoare, 1973

### Quotes

Make it as simple as possible, but not simpler

→ A. Einstein

### Heritage

Go's heritage is at least as much as Oberon as it is C

- Packages
- Imports
- Strict Memory Safety
- GC
- Dynamic Type Checks
- Etc

# Object Orientation in GO: Interfaces

Inspiration: Smalltalk (Alan Kay, Dan Ingalls, Adele Goldberg, 1972 - 1980)

- Everything is an object
- Any message can be sent to any object

Want: Similar power in (mostly) statically typed language without the type-system fuss

- · Notion of interfaces for static typing
- Usually objects carry type information → restricts object types to "classes".

**Crucial insight:** Can attach methods to any type if interfaces carry type info rather than objects.

Methods and interfaces are the only additional mechanisms needed for OO programming.

### Concurrency

- Good concurrency support was considered essential from day one
- Rob Pike's work on NewSqueak turned out to fit really well into Go.

### Origins:

- "Newsqueak: A Language for Communicating with Mice" → Rob Pike, 1994
- "Communicating Sequential Processes", → CACM, C.A.R Hoare, 1978

#### Generics

- Single biggest language feature asset in Go
- Often missed by newcomers to Go
- Type-system mechanism; unclear if an essential language feature
- Incredibly complex in both semantics and implementations (considerable ones)
- Trade-offs: Larger binary, slower binary, or larger source code
- Not-orthogonal: Affects many other language features as well as how the library is written
- Hold off and keep thinking about it

# Putting It All Together

Had the luxury to spend two years hammering out the basics (thanks to Google).

Crucial: add one feature at a time.

Initially: Team of 3 very different people

Intensive discussions, emotional

Humbling experience

Having multiple people illuminating each new feature from different angles made the language much stronger.

#### Later:

- Russ Cox's razor cutting through the crud, making it work well.
- Ian Lance Taylor providing a 2nd implementation (validation of design)
- Go/types (now in 1.5!) provides a 3rd frontend (validation of compilers and spec)

Having 3 frontends proved tremendously useful

## **Evlolving Go**

Original design went through many (syntactic and semantic) transitions:

- Parallel lib development ensured we didn't design "into the blue".
- gofmt (for language changes) and gofix (for API changes) for existing code.

Features that came in much later:

- Optional semicolons, optional types for composite literals, optional bounds in slice expressions, recover, etc.
- Semantic clarifications (maps, channel ops, etc)
- Small backward-compatible adjustments still happening at a very low rate

### The Future of Go

### What Makes a Programming Language Successful?

- Clear target
- Solid implementation: language, libraries, tools
- Market readiness
- Technological breakthrough
- Language features without competitors
- · Rarely: Marketing (see Java)

#### How About Go?

- Clear target behind design
- Multi-paradigm (imperative, functional, object-oriented)
- Syntactically light-weight
- Language features without competition:
  - goroutines

- interfaces
- defer
- Tools without competition:
  - fast compiler
  - gofmt
  - go build
- Strong standard libs
- Solid implementation
- Excellent documentation, online tools (playground, tour)
- Notice: No corporate marketing to speak of.

#### Will Go Ever Become mainstream?

- Need to cross the chasm from early adopters to early mainstream
- Go community must remain unified behind this goal
- Don't make too many mistakes going forward

It takes about 10 years for a programming language to become "established".

#### **Pitfalls**

The language is frozen, but these are a form of "language design":

- Build tags and other specialized comments
- Special interpretation of import paths and canonical import path comments
- Internal packages
- Vendoring descriptions

These are not part of the language spec and thus may diverge over time or have different semantics on different platforms.

Need to be watchful of this development.

# **Closing Thoughts**

- In 1960, language experts from America and Europe teamed up to create Algol 60.
- In 1970, the Algol tree split into the C and Pascal branch.
- 40 years later, the two branches join again in Go.
- Let's see if Go can enjoy an equally long run as its predecessors!