GoLang

A Programming Language Case Study

Kyle Jon Aure

Version 1.0, 2018-10-31

Table of Contents

1. Introduction

In the computer science field, programming languages vary almost to the same degree of spoken languages. Like spoken languages there exist programming languages that are well established and often used. Then, there are languages that are rarely spoken, and those that are dying due to unpopularity. In this case study the language to be examined is `Go`.

The art of language design and the factors that constitute an objectively 'good' language vary. Those who develop programming languages do so for specific purposes, company interests, or to improve an existing language. (Scott, 2016, pp. 7-9) Therefore, this case study will examine the history, support, ease of use, software engineering capabilities, and performance of the `Go` programming language.

2. History

The `Go` programming language was developed by Robert Griesemer, Rob Pike, and Ken Thompson. These programmers started working on `Go` in 2007, and it became a public open source project on November 10, 2009. ("FAQ/Origins," n.a.) Today, it is estimated that there are between 504,000 to 966,000 `Go` programmers, often referred to as `Gophers`. (Cox, 2018)

`Go` was developed in response to the boom of cloud computing in the last 2000's. At the time, developing server software using the traditional programming languages like `C++` and `Java` was becoming increasingly complex. Griesemer, Pike, and Thompson wanted to develop a language that would tackle this problem and make concurrency, garbage collection, and memory management more optimized. Since, the release of `Go` in 2009 the platform has yet to see a leveling off of new users adopting the language. ("Stack Overflow Trends," 2018)

3. Support and Uses

The `Go` programming language has made a big impact on the cloud computing space with the most notable products developed on `Go` being `Docker` and `Kubernetes`. In addition, as a Google product `Go` is being used to develop some of the largest scale software today. (Pike, 2012) More and more companies are turning to `Go` for their product development including, BBC, Uber, and SoundCloud.

`Go` can be developed on the Windows, MacOS, Linux, and more open source operating systems. (Cōngruì, 2018) Additionally, `Go` can be used to develop software for use on personal computers, servers, and mobile devices. [[Add More]]

Being a newer language `Go` was developed to be supported by existing Integrated Development Environments (IDEs). `Vim`, `Visual Studio Code`, `IntelliJ`, and `Atom` are some of the most popular IDEs with `Go` plugins that support syntax highlighting, debugging, and testing. [[Add More]]

4. Ease of Use

`Go` is an imperative programming language that includes themes from the Von Neumann, Object-Oriented, and Scripting language classifications. (Scott, 2016, pp. 11-12)

4.1. Lexical Elements

4.2. Types

```
Public Class Foo (int bar){
  int a; ①
}
```

4.3. Functions

4.4. Scope

4.5. Garbage Collection

- SYNTAX: What forms does the language allow?
  - How does the programmer know how to construct statements that will be accepted by the complier?
  - Standard notation system?
- SEMANTICS: What does this form mean?
  - Is there any ambiguity? (for example dangling else)
- EXPRESSIVENESS: Is there consistancy with commonly used notation?
  - Is arithetic infix, postfix, or prefix?
  - How are assignments (=) and assertions (==) differentiated?
  - How is input-output handled?
- UNIFORM: Do similar constructs have similar meanings?
  - If I pass a parameter to a function will it be taken by value or reference?
- ORTHOGONAL: Are all concepts independent from one another?
- GENERAL: Can this language handle any type of problem?
  - What cannot be done?
- PEDAGOGY: Is it easy to learn?
  - In general how easy is the language to pickup?

5. Software Engineering

- RELIABILITY: Is it difficult for errors to go unnoticed by the compiler/programmer?
  - What sort of type checking is done?
  - How readable is the program?

- MODULARITY: Can a large program be broken up into moduals?
    - Are functions, subroutines, libraries, etc supported?
- EXTENSIBLE: Can a programmer easily create and use new data types to fit the problem at hand?
    - What data types are supported?
    - What data structures are included in the Lanaguage?
- PROVABILITY: Can formal methods be used to prove the correctness of a program?

6. Performance

- FAST COMPILATION: How long does compilation take?
    - How is the program compiled?
- EFFICIENT OBJECT CODE: How much object code is producted after compilation?
    - Does the language make neccessary verbose object code?
- PORTABILITY: Can a program be run on different opperating systems?
    - Does it need to be recompiled?

7. Conclusion

8. Bibliography

Stack Overflow Trends. (2018). Stack Overflow. Retrieved from https://insights.stackoverflow.com/trends?tags=go%2Chaskell%2Cgroovy

Cox, R. (2018). How Many Go Developers Are There? Research!rsc. Retrieved from https://research.swtch.com/gophercount

Cōngruì, W. (2018). GoLang Minimum Requirements. GoLang. Retrieved from https://github.com/golang/go/wiki/MinimumRequirements

Pike, R. (2012). GoLang Minimum Requirements. Google. Retrieved from https://talks.golang.org/2012/splash.article

Scott, M. L. (2016). *Programming Language Pragmatics*. Morgan Kaufmann Publishing.

FAQ/Origins. (n.a.). GoLang. Retrieved from https://golang.org/doc/faq#Origins