A Comparison between Go and C++

JOAKIM ANNEBÄCK
and JOHAN STJERNBERG

KTH Computer Science
and Communication

Bachelor of Science Thesis
Stockholm, Sweden 2011

A Comparison between Go and C++

J O A K I M A N N E B Ä C K
a n d J O H A N S T J E R N B E R G

Bachelor's Thesis in Computer Science (15 ECTS credits)
at the School of Computer Science and Engineering
Royal Institute of Technology year 2011
Supervisor at CSC was Alexander Baltatzis
Examiner was Mads Dam

URL: www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2011/
anneback_joakim_OCH_stjernberg_johan_K11073.pdf

Kungliga tekniska högskolan
Skolan för datavetenskap och kommunikation

KTH CSC
100 44 Stockholm

URL: www.kth.se/csc

Abstract

This is a study on the new, Google-based, programming language Go. In the study, selected common areas in programming languages are described and when available how they work in Go. Furthermore, it features a comparison between Go and C++ on these areas as well as in summary.

Go introduces a new approach on concurrent programming that is much easier than in the recognised languages of today and works very smoothly and quickly both when it comes to compiling and running. However, some very popular features such as inheritance and generic data types have been left out and is not currently supported.

We believe that the ideas introduced with Go are very interesting indeed but that it is too early to tell if Go will become a recognised language or stay a small-scale experiment for Google.

Referat

En jämförelse mellan Go och C++

Det här är en studie av det nya, Googlebaserade, programmeringsspråket Go. Utvalda delar av programmeringsspråk beskrivs i studien och hur de fungerar i Go, då de stöds. Dessutom jämförs Go med C++ både inom de olika områdena samt sammanfattningsvis.

Go introducerar ett nytt tänk vid programmering i flera trådar som är mycket lättare än i de vedertagna språken i dag och fungerar väldigt smidigt samt snabbt både när det gäller att kompilera och köra programmen. Dock har några mycket populära funktioner såsom arv och generiska datatyper utelämnats och stöds för närvarande inte.

Vi anser att de idéer som införs med Go är mycket intressanta men att det är för tidigt att säga om Go kommer att bli ett vedertaget språk eller förbli ett småskaligt experiment för Google.

Contents

Chapter 1

Introduction

1.1 Preface

This document is was written for the course *Degree Project in Computer Science*, DD143X/dkand11 at the Royal Institute of Technology. The authors of this document are Joakim Annebäck and Johan Stjernberg, both 3rd year students at the Computer Science programme at the Royal Institute of Technology. Their supervisor for the work was Alexander Baltatzis and the examiner Mads Dam.

The purpose of the document was to investigate the programming language *Go*. Both authors have been engaged in every part of the project. The writing of the document has been divided evenly between us two authors.

Joakim Annebäck has written: History of Go, Background to C++, Inheritance, Data Types, Garbage Collection, Speed, Operators.

Johan Stjernberg has written: Purpose, The Purpose of Go, Functions and Methods, Syntax, Compilation, Results.

Both have worked together on the Problem Statement, Method, Discussion and Conclusion.

1.2 Purpose

Our project aims to investigate the new programming language Go, which is currently being developed by some Google employees. Very few new programming languages have been successful over the last couple of years, so when a well-established company like Google announces a new one it is of course tempting to try it out. Here is a quote from the Go projects' web page:

> Go was born out of frustration with existing languages and environments for systems programming.[1]

The goal of our report is to analyse the programming language; what distinguishes Go from other languages? Is it better than the rest? Will it be a huge success or suffer the same destiny as Google Wave, that did not become the success Google hoped for? These and similar questions have made us interested in trying out Go and is the main reason for our choice of project. Another is the challenging twist in that Go is so new and unexplored.

1.3 Problem Statement

- What are the advantages and disadvantages with Go?
- What are the differences between Go and the more well-established programming language C++?
- Proof of concept: the development of a simple Go application, for example the implementation of a guest book or search algorithm.

In the first question we want to investigate, in general, how good Go is compared to other languages. We also want to know for what kind of tasks Go is more suitable to use than other languages.

Since the first question is more about the general view of Go, the point of the second question is to analyse more closely and in detail how different the language is to another more established programming language. We have chosen the recognised and well-established language C++ for this comparison.

Lastly, we include a simple demonstration of how Go can be used and in particular how the code looks. We choose to implement the Bubble sort algorithm, showing how Go works and how the syntax looks through an easy example like this.

Chapter 2

Background

2.1 What is Go?

Go is an open source programming language developed largely by Google employees. It is intended to be fast compiled, strongly typed, garbage collected and with explicit support for concurrent programming.

2.2 History

The original developers Rob Pike, Robert Griesemer and Ken Thompson started out in September 2007 with the design of the new programming language, Go. In the beginning of year 2008 Thompson had started working on a compiler which generated C code as output. This was to investigate ideas on design of the compiler for Go. Since the development had now become a full-time project, just a few months after the first experiments with a new compiler, it had been stabilised enough for a production compiler. Ian Taylor started working on a GCC front end for Go using the draft specification in May 2008. In the end of 2008 Russ Cox helped move the language and libraries from prototype to reality. Since Go is a open source project many others have contributed to it with code, documentation, ideas and discussions. It was officially announced with implementations for Linux and Mac OS X in November 2009. Implementation for Windows has also been announced although it is not optimised.

2.3 The Purpose of Go

While the computer hardware has improved a great deal over the last decade, the improvement of the programming languages have more or less stopped. There have not been any new successful programming language in many years now, and the Go developers hope to change that.

Go is an attempt to combine the ease of programming of an inter-

> preted, dynamically typed language with the efficiency and safety of a statically typed, compiled language. It also aims to be modern, with support for networked and multicore computing. Finally, it is intended to be fast: it should take at most a few seconds to build a large executable on a single computer.

2.4 Background to C++

C++ was invented by the Danish programmer Bjarne Stroustrup. He began developing a language in 1979 called "C with Classes" which later came to be known as C++. Stroustrup used some design rules when developing C++ :

- C++ should be type safe, statically typed.
- C++ should have support for separate compilation with multiple programming styles, e.g. object-oriented programming, generic programming.
- C++ should be as compatible with C as possible for easy transition.
- C++ uses the "zero-overhead rule".[4, p. 121]
- C++ should be designed to give the programmer a choice, even if it is possible to make an incorrect one.[4, p. 121]

C++ deals with type safety by checking all function calls at compile-time.[4, p. 92] With the use of header files C++ gives support for separate compilation with multiple programming styles.[4, p. 120] Instead of designing C++ with support for only C++ techniques, Stroustrup designed C++ to be backwards compatible with C. Thus making it easy for programmers used to C to start programming in C++, using "warnings" for suggesting the C++ technique instead of using the C technique.[4, p. 112] Virtual functions, multiple inheritance, runtime identification, exception handling and templates all owe a part of their design to the "zero-overhead rule", which states that: "What you don't use, you don't pay for". In C++ the programmer can know the cost of using a feature, in the language, since it is not distributed, over e.g. various classes. The programmer then get to choose for himself making the decision between cost and benefit of using this particular feature.[4, p. 121]

2.5 Inheritance

Although Go provides methods and types that makes it possible for an object-oriented style of programming, Go does not have type inheritance.[1] Where other programming languages, i.e. C++, features templates, classes and subclasses Go provides interfaces. The interface in Go can be compared to a pure abstract class[1]

[1] A class that contains no data and where all methods are pure virtual.

in C++. An interface can also be very lightweight since it does not even have to contain any methods, if there is no need for it. Types in Go can satisfy more than one interface at once, without any of the complexities that follows with traditional multiple inheritence.[1] To declare an interface in Go simply use following notation:

```
type Test interface {
    // List of Methods
}
```

In comparison to Go, C++ is object-oriented and offers classes. This provides inheritance, which allows one type of data to inherit from other data types. Multiple inheritance is also supported by C++. This gives a data type the ability to inherit from one or more data types. With inheritance the base class does not only provide the declaration of the member function, but also implementation and member data.[5]

2.6 Functions and Methods

Although similar in general, there is at least one notable difference between C++ and Go on this matter. Go lacks the feature of function overloading which is quite commonly used in C++. That is, in Go you may not have several declarations of a method with the same name but different parameters. According to the Go developers, the reason for this is that it simplifies method dispatch and makes code less confusing.[1]

2.7 Data Types

When it comes to data types Go has many of the common data types that most of the object-oriented programming languages popular today have.

- Pre-declared numeric types: *uint, int, float, complex, uintptr*
- Integer numeric types: *uint8, uint16, uint32, uint64, int8, int16, int32, int64*
- Floating-point numeric types: *float32, float64*
- Complex types: *complex64, complex128* [3]

In comparison to C++ int and int32 are two distinct types, even if both have the same size. Since the int type is generic, specification of the size is required if the number if bits it holds is important. Here is an example showing the declaration of two different types of integers:

```
var i int32:        // i and n do not have the same types
var n int = 17;
```

The default size of an int or an uint is 32 bits no matter if the 32 or 64 bits compiler is used. The data types complex and floats are always sized, "because programmers should be aware of precision when using floating-point numbers". [1]

When it comes to strings in Go, they are immutable values[3], which mean that they are not just arrays of byte values. Like a const string in C++, the string in Go can not be change once it has been built. Even if the string itself cannot be changed, the string variable still can by simply reassigning it.

2.8 Generic Types

Go does not currently support generic types. However, it does support objects without a type using the empty interface.

> Generics are convenient but they come at a cost in complexity in the type system and run-time. We haven't yet found a design that gives value proportionate to the complexity, although we continue to think about it. Meanwhile, Go's built-in maps and slices, plus the ability to use the empty interface to construct containers (with explicit unboxing) mean in many cases it is possible to write code that does what generics would enable, if less smoothly. [1]

C++ on the other hand has support for generic types. Generic programming is supported by the use of function templates, class templates and, in some cases, specialised templates in C++.

A function template is a special function that can operate with generic types. When creating a function template it's functionality can be used for multiple data types without the need of repeating the code for each data type. With a special kind of parameter - a so called template parameter - this can be done in C++. The template parameter can be used to pass a type as an argument similar to a regular function parameter that can be used to pass a value to a function. The function template can use these parameters as if they were regular types. All the following code examples are in C++. [6]

```
template <class identifier>
    //function declaration
;
```

Class templates gives the possibility for a class to have members that use template parameters as types.[6]

```
template <class identifier>
class classname {
    //class declaration
};
```

In some cases one may want to define a different implementation for a template where a specific type is used as a template parameter, this can be done using a specialisation of that template.[6]

```
template <> class classname <specified_type> {
    //specialised code
}
```

2.9 Concurrency

Goroutines are Go's model of threads/coroutines. To start a goroutine, one can simply put *go* as a prefix to any function call. This causes the function to run in a new, different goroutine, which is run in parallel with the current computation, in the same address space. [8]

```
func server(i int) {
          for {
                  print(i)
                  sys.sleep(10)
          }
}
go server(1) //start new Goroutine
go server(2) //start new to run in parallel with the first one
```

According to the Go team, goroutines are cheap and developers need not to worry about the stack size.[9] In contrast, there is currently no built-in way to manage threads in C++, although the feature is expected in the upcoming new standard and is already today possible using third-party libraries. When starting the Go project, one of the big goals was to create a design that makes software development easier on multi-core hardware.[1]

The main method of synchronising between goroutines is the *channel communication*. Each send operation on a particular channel is matched to a corresponding receive operation from that channel, usually in a different goroutine. The send operation on a channel happens before the receive operation from the channel completes.

```
var c = make(chan int, 10)
var a string

func f() {
  a = "hello, world"     //write to a, before it is sent to c
  c <- 0    //send to c
}

func main() {
  go f()    //start new goroutine
```

```
  <-c    //receive on c completes
  print(a) //print string "hello, world"
}
```

2.10 Syntax

The Go syntax is somewhat unique. Semicolons are optional but brackets in if and else cases are mandatory. This will result in a compile-time error:

```
if(5 > 3) fmt.Println("great");
```

whereas this is okay:

```
if(5 > 3) { fmt.Println("great") }
```

One may also note that the quite popular ternary operator, available in C, C++ and Java among others, is due to this fact not available in Go. The following will generate a compilation error:

```
x = (x > 10 ? x : 10);
```

Functions or methods work similar to many other programming languages but have an unconventional syntax. Here is a method head example that is valid:

```
func returnTwo(a, b int) (c int, d int) {
```

Note that the data type (here: int) must follow after the variable name, in contrast to having it before the variable name which is more conventional. It is also possible to skip the data type if it is the same for several variables in a row, as demonstrated by the parameters a, b here. Another pleasant feature is the possibility to return more than one variable (here: c, d), which is not a unique feature for Go but still quite uncommon today. In this case we chose to explicitly say that both c and d are of type integer, but we could just as well have written it the same way as the parameters a, b. Lastly, note that the return statement comes last in this case, as opposed to languages like C and Java which start with the return parameter.

Here is an example method summing up two integers a, b and returning the result s.

```
func sum(a, b int) int { // returns an int
    s := a + b
    return s
}
```

When it comes to iteration and loops, the syntax is somewhat similar to what we are used to.

```
for i := 0; i < 10; i++ {
    fmt.Println("Num:", i)
}
```

Note here the use of ':=' in the first statement. It means that we declare i and that Go is to automatically guess the data type. Alternatively, we could have specified i to be of type integer as is the case here:

```
var i int
for i = 0; i < 10; i++ {
    fmt.Println("Num:", i)
}
```

However, in this case we use '=' instead of ':='. Using ':=' in this case will result in a compile-time error due to the fact that i has already been declared.

While loops do exist but do not use the conventional name 'while' - they are also called for-loops.

```
for i < 7 {
    i++
}
```

Many programmers would like to use paranthesis in if- and for-statements such as this, ie. $for(i < 7)\{...\}$, but it is not only discouraged but even forbidden in Go and results in a compile-time error.

Another syntax related decision worth mentioning is Go's approach on the everlasting fight between programmers as to whether the opening bracket after a construct, ie. if-case, should be on a line of its own or at the same line as the construct. Go forbids having the opening bracket anywhere but at the same line, so the following will not compile:

```
if a > 5
{
[...]
}
```

However, the following works fine:

```
if a > 5 {
    [...]
}
```

2.11 Garbage Collection

Go uses garbage collection for memory management, which means that it is not possible nor necessary to free memory explicitly like in C++. Currently, Go uses a mark-and-sweep collector for this task. The automatic garbage collection also eases the writing of concurrent programming code as the programmer does not need to think about memory management as different objects are passed between Goroutines. The Go team's approach has been that it's easier for them to tackle

this quite complex issue once and for all rather than having every programmer think about it while developing. [1]

C++ does not have any garbage collection and relies heavily on that the programmer knows what he or she is doing. However, a default destructor is automatically created if the programmer does not write one. The default destructor will free up memory in some obvious cases but is unable to safely handle objects accessed in more complex ways, such as through pointers. The pointer will be deleted, but not the object itself.

2.12 Compilation

There is a set of compilers for Go, supporting both 32 and 64 bits systems, named 6g, 8g and 5g. Additionally there is a compiler that uses gcc named gccgo, which has proven to be faster in some cases.

6g - the original Go compiler - was first considered to be written in Go itself, but the developers decided not to because of issues with bootstrapping and the open source distribution. It is the most "mature" compiler with an efficient optimiser that generates good code. The compilers can be used with Linux, FreeBSD, OSX and Windows operating systems.

One of the main objectives with the Go language was to not only run programs quickly but also compile them very fast. The compilation times can be reduced with help from goroutines, which splits up the compilation into more than one thread, making it compile simultaneously.

2.13 Speed

As mentioned before one of the main purposes of Go was to have a fast programming language when compiling code. Go is a concurrent language which uses the fact that the computers have gotten faster and faster, hardware-wise. With the use of goroutines and the fact that it is a fully garbage collected programming language, it utilises the multi-core CPUs which for the most part results in a fast compilation.[1] Sometimes the use of more than one CPU core may result in the program slowing down. This depends on the nature of the program. If its Goroutines takes a lot of time communicating on channels it may result in a decrease of preformance. Since Go is a relatively new programming language the Goroutine scheduler is not yet fully developed. The aim is to have it recognise these kinds of cases, where use of multi-core slows the program down, and optimise the usage of OS-threads. [1]

In comparison to C++ it is much faster at compiling when it comes to larger programs. One reason for this is that Go does not use header files, instead each source file is a part of a defined package[1]. This avoids much of the included files and libraries you get from using header files. Also there is no hierarchy in Go which increases the speed since there is no time spent defining the relationships between types.[1]

2.14 Operators

Table 2.1. C++ Operators [7]

Operator	Description
::	scope
() [] . -> ++ −− dynamic_cast static_cast reinterpret_cast const_cast typeid	postfix
++ −− ˜ ! sizeof new delete	unary (prefix)
* &	indirection and reference (pointers)
+ -	unary sign operator
(type)	type casting
.* ->*	pointer-to-member
* / %	multiplicative
+ -	additive
≪ ≫	shift
> < >= <=	relational
== !=	equality
&	bitwise AND
^	bitwise XOR
|	bitwise OR
&&	logical AND
||	logical OR
?:	conditional
= *= /= %= += -= ≫= ≪= &= ^= |=	assignment
,	comma

As shown in the table Go has almost the same kind of operators as C++ has. Although Go has support for pointers it does not provide pointer arithmetics. Therefore this kind of operators does exist in C++ but not in Go[8]. Another notable difference is that C++ supports overloading of operators. Go does not support operator overloading since the developers think that “things are simpler without it” and “it seems more a convenience than an absolute requirement”. [1]

Table 2.2. GO Operators

Operators	Description
* / % ≪≫ & &^	arithmetic, multiplication
+ - | ^	arithmetic, addition
== != < > <= >=	comparison
&&	logical AND
‖	logical OR
!	logical NOT
<-	receive
& *	address
(type)	conversion
= *= /= %= += -= ≫= ≪= &= ^= |=	assignment
[]	index

Chapter 3

Method

Go was announced in November 2009 and since it is so recent very few reports on the subject exist. However, a great deal of material – including a thorough language specification – is available on the web and we believe it made a sufficient basis of material for our study.

Our research is mostly based on the language specification and other documentation about the Go language in comparison to some other languages in general, and in particular compared to C++. We have also investigated other relevant material on the subject and compared how a specific algorithm work in the two primarily compared languages.

We have in our research tried out both the existing 8g compiler and *The Go Playground*[2]. We used them for testing how the syntax, functions, methods, compilation and concurrency work.

Apart from this, we also conducted a small-scale survey on important features and qualities in programming languages, in order to get a feeling of how important the Go features and skipped features are.

Chapter 4

Code comparison

In the investigation between Go and C++, we have compared the well-known "Bubble sort" algorithm written in both programming languages.

At first we can see that the Go implementation takes just an array of ints, using the built-in function *len* to get the length of it. The C++ implementation takes a vector of ints as input, using the built-in function *size* to get the number of elements. Since there are of course many ways that we can implement the algorithm in both of the languages, we can only confirm the syntax of functions and methods corresponding to eachother are very much alike. Compared with eachother Go has the absence of brackets and semicolons, and in C++ we have to specify that the *int*s *i* and *j* in the for-loops have to be *unsigned int*s.

Firstly, we take a look at a C++ implementation.

```
#include <vector>
using std::vector;
void bubbleSort(vector<int> &v){ //takes an vector of int as input
    if(v.size()>0){//check if the vector is empty, if it is we don't do anything
        int temp;
        for (unsigned int i = 0; i < v.size() - 1; i++){
            for(unsigned int j = i+1; j < v.size();j++){
                if (v[i] > v[j]){ //if two elements are in the wrong order, swap!
                    temp = v[i];
                    v[i] = v[j];
                    v[j] = temp;
                }
            }
        }
    }
}

int main(){
```

```
    vector<int> vec(10); //test
    vec[0] = 1;
    vec[1] = 2;
    vec[2] = 0;
    vec[3] = -5;
    vec[4] = 8;
    vec[5] = 9;
    vec[6] = 27;
    vec[7] = 3;
    vec[8] = -7;
    vec[9] = 9;
    bubbleSort(vec);
    for(unsigned int index = 0; index<vec.size();++index){
        std::cout<< vec[index] <<endl;
    }
}
```

Secondly, we look at the same algorithm but in Go.

```
package main

import "fmt"

// Takes an array v of type int as input and returns the sorted array.
func bubbleSort(v []int) []int {
  var temp int
  // Loop through all of v
  for i := range v {
    // Loop from current i to end of v
    for j := i + 1; j < len(v); j++ {
      if v[i] > v[j] {
        // Two elements are in the wrong order. Swap!
        temp = v[i]
        v[i] = v[j]
        v[j] = temp
      }
    }
  }
  return v;
}

func main() {
  var a = []int {  2, 4, 9, 3, -2, -3, 0, 18, -5}
  var b = []int {}
  var c = []int {3}
```

```
  var d = []int {-5, 5}
  var e = []int {5, -5}

  fmt.Println(bubbleSort(a))
  fmt.Println(bubbleSort(b))
  fmt.Println(bubbleSort(c))
  fmt.Println(bubbleSort(d))
  fmt.Println(bubbleSort(e))
}
```

Chapter 5

Survey

In order to get a better feeling of how important different built-in features in a programming language are, we wrote a survey that other students in Computer Science at the Royal Institute of Technology (KTH) answered. The question was how important six specific features are to have built-in in programming languages. The features were garbage collection, concurrency, inheritance, generic types, speed (compiling) and speed (runtime). For each feature or quality, the respondee was asked to choose how important, on a scale from one (not important) to five (very important) it is to have that feature or quality in a programming language. In the question formulation, it was made clear that it regards having the feature built-in in the language.

15 students took the survey independently and filled it out on printed paper. The result is provided here as the mean value of the responses on the same scale from one to five, where one means not important and five very important.

As seen in the results, although with a small margin, the students prioritise garbage collection, inheritance and run-time speed over concurrency, generic types and compilation speed.

The survey that we conducted with students at the Computer Science programme at the Royal Institute of Technology in Stockholm gave us some information about what kind of functions are important, for programmers, in a programming language.

Garbage collection	4.00
Concurrency	3.87
Inheritance	4.13
Generic types	3.87
Speed (compilation)	3.27
Speed (runtime)	4.27

Table 5.1. Survey result with the mean value of importance level for different features and qualities in programming languages, according to some KTH students.

Assuming these indications are true, it was a very good design choice of the Go team to include automatic garbage collection, and a big plus that it also can handle garbage when working with multiple threads. However, the lack of support for inheritance, which we believe is important in order to conduct object-oriented work, is a big issue that the Go team should really consider solving.

It should be noted that this is a small-scale survey and that the result is not statistically reliable. However, we see it as an indication of that inheritance and garbage collection are important and appreciated features. Furthermore, it seems more important that programs run fast than that they compile quickly. In fact, the respondees seem to think that quick compilation is the least important factor of those included in the survey. We think this factor could depend on how big the programming projects are and since the students at the university usually do not conduct large scale projects, they may think it is less important than programmers at companies like Google, who may save a lot of money and time if the compile times are quick.

Chapter 6

Discussion

In the aims for Go, the developers were really focused on designing a concurrent and fast compiling programming language that utilises the power of computers today, with emphasis on that there has been a lot of improvement of the computer hardware in the last years, but not in the software field.

There has not been any successful programming languages released in over a decade that has taken the rapid improvement of computers' hardware and used it to its advantage[1]. The Go team is on its way to achieveing this goal, but because of this some of the functions, that are found in the more common programming lanuages (such as Java and C++), have been set aside. For example support for generic programming has not yet been implemented and remains an open issue. Going back to the first question in the problem statement – *What are the advantages and disadvantages with Go?* – this may be a disadvantage for Go reaching its goal to emerge as a major programming language.

One of the main contributing factors to the speed of Go is that it is a concurrent programming language with the unique approach in the introduction of the *goroutines*. The purpose of goroutines is to make concurrency easier to use. In comparison, it is possible to run multiple threads in C++, but it is not a built-in feature in the language. Therefore, one currently needs to use some kind of third-party library for concurrency to work. This is clearly not as easy as just writing *go* as a prefix to a function for starting a new goroutine, which we see as a big advantage of Go.

The Go developers claim that the use of interfaces instead of inheritance has real advantages, and state that one of the contributing factors to the fast compilation of Go code is the absence of inheritance. In Go, types can satisfy more than one interface at once, which eliminates the complexities of the multiple inheritance that exists in e.g. C++. Therefore, no time is spent on defining relationships between types[1].

Chapter 7

Conclusions

C++ and Go are two quite different languages regarding focus during development and which features are built-in. For instance, C++ has extended support when it comes to inheritance, generic types and pointer arithmetics while Go focuses on garbage collection, concurrency and compiling quickly.

Go's main advantage is its ability to produce fast compiling programs utilising concurrent programming. But because of the absence of inheritence it is not an object-oriented programming language like the more popular languages today, e.g. Java and C++.

C++ was originally designed and developed in the 1980's and despite being very popular even today, a lot has happened since then and the focus has in some cases changed since then. For example, concurrency was not a big deal back then but is a very hot topic today as multi-core processors become more and more common. The Go developers say that the language was born out of frustration with existing languages and aims to be a very modern language with the latest features. In our opinion, the extensive support for concurrency in terms of goroutines supports that claim.

Despite this, the long awaited new standard for C++ does contain native concurrency support and it is still a much more popular language than Go. Although Go is still a very young language, we feel that the development team needs some sort of a major breakthrough in order to make Go a commonly used language in the programming society. If they could find a good way to support generic types and inheritance without slowing down Go or making coding too complex, that would probably be a very good beginning.

Furthermore, the Go team enforces both unconventional syntax and a new thinking when it comes to software development that most people today are not used to or trained in. "Do not communicate by sharing memory. Instead, share memory by communicating." [9] This is an example of how the Go team asks programmers to reconsider how they work and try their new ideas. In this case, they mean that concurrent programming in Go is different from that in C, C++ and Java. Although the syntax and built-in support for concurrent programming is not too hard to get

used to, we believe that it takes time to have people change their way of coding and thinking.

Even if it is all but certain that Go will ever have a major breakthrough and become an accepted, popular language, one can only hope that the ideas of simplicity and making concurrent programming much smoother that Go introduces will inspire other language designers in their future work. We find the ideals of Go highly interesting, and believe that they really do have good potential in making software development less painful.

Bibliography

[1] The Go Programming Language: FAQ, 2011. Available on Internet: `http://golang.org/doc/go_faq.html`; [Accessed 2011-04-13]

[2] The Go Playground, 2011. Available on Internet: `http://golang.org/doc/play/`; [Accessed 2011-04-10]

[3] The Go Programming Language Specification, 2011. `http://golang.org/doc/go_spec.html`; [Accessed 2011-04-13]

[4] Stroustrup, B. (1994) *The Design and Evolution of C++*. AT&T Bell Laboratories, Murray Hill, New Jersey.

[5] Friendship and inheritance, 2011. `http://www.cplusplus.com/doc/tutorial/inheritance/`; [Accessed 2011-04-10]

[6] Templates, 2011. `http://www.cplusplus.com/doc/tutorial/templates/`; [Accessed 2011-04-10]

[7] Operators, 2011. `http://www.cplusplus.com/doc/tutorial/operators/`; [Accessed 2011-04-10]

[8] Go For C++ Programmers, 2011. `http://golang.org/doc/go_for_cpp_programmers.html`; [Accessed 2011-04-08]

[9] Pike, R. (2009). *The Go Programming Language*. `http://golang.org/doc/GoCourseDay3.pdf`;[Accessed 2011-03-20]

[10] Schmager, F. (2010).*Evaluating the GO Programming Language with Design Patterns*. Victoria University of Wellington, New Zealand `http://ecs.victoria.ac.nz/twiki/pub/Main/TechnicalReportSeries/ECSTR11-01.pdf`; [Accessed 2011-04-08]