Justification of Programming Languages – an Introduction

LOVISA BRÖDERMAN SKEPPE

DD144X, Bachelor’s Thesis in Computer Science (30 ECTS credits)
Degree Progr. in Computer Science and Engineering 300 credits
Royal Institute of Technology year 2012
Supervisor at CSC was Linda Kann
Examiner was Olle Bälter

TRITA-CSC-E 2012:095
ISRN-KTH/CSC/E--12/095--SE
ISSN-1653-5715
Abstract

New programming languages emerge all the time. To distinguish them from each other, the language designers provide reasons to justify their languages. What do these reasons look like and are they enough to justify a languages existence?

The thesis concludes that language theoreticians need to find appropriate socio-technical methods to evaluate programming languages. Language designers should also make use of scientific research to justify the problems they attempt to solve with their language.

I have found that there could be economic aspects in owning a language and thus it should be financially justified to find good research methods concerning the creation of new programming languages.
Referat

Motivering bakom skapandet av programmeringsspråk

Nya programmeringsspråk dyker upp hela tiden. För att skilja dem från varandra ges skäl från skaparna som motiverar deras språk och dess design. Hur ser dessa skäl ut och är de tillräckliga för att motivera ett språks existens?

Avhandlingen drar slutsatsen att språk teoretiker måste hitta lämpliga socio-tekniska metoder för att utvärdera programmeringsspråk. Språkkapare bör också använda sig av vetenskaplig forskning för att motivera de problem som de ser och som de försöker lösa med deras språk.

Jag har funnit att det kan finnas ekonomiska aspekter på att äga ett språk och därför bör det vara ekonomiskt motiverat att hitta bra forskningsmetoder för utvärdering av nya programmeringsspråk.
Contents

1 Introduction ......................................................... 1
  1.1 Problem definition ............................................. 1
2 Methodology ......................................................... 1
  2.1 Limitations ..................................................... 2
3 Definitions .......................................................... 3
  3.1 Programming language .......................................... 3
  3.2 Domain-specific languages (DSL) ............................. 3
  3.3 General-purpose programming language ...................... 3
  3.4 Programming paradigm .......................................... 3
  3.5 Core language and standard libraries ......................... 3
  3.6 Open-source .................................................... 3
4 Background .......................................................... 4
  4.1 The history of programming languages ....................... 4
  4.2 Software Sciences neglect of human factors ................ 4
  4.3 Criticism against language design and how it is conducted . 5
  4.4 Hardware development - Multi-core processors .............. 5
5 Go ................................................................. 6
  5.1 Go the programming language .................................. 6
  5.2 Purpose of Go .................................................. 6
6 Java ............................................................... 7
  6.1 Java the programming language ............................... 7
  6.2 Purpose of Java ................................................ 8
  6.3 The story behind Java and why it became so popular ........ 9
7 Discussion .......................................................... 10
8 Conclusion .......................................................... 12
9 Reflections on my own work ....................................... 12
  9.1 Source criticism ............................................... 13
  9.2 Ideas for further research ..................................... 13

Bibliography 14
1 Introduction

With numerous programming languages available today, I am probably not the first person to ask the question: why do we need new programming languages? Instinctively one might think it is because of the technical improvements a new language can offer. But how can we be sure that a new language provides this? Can we trust what the authors of a language claim?

Language designers normally publish a design paper, white paper or other documentation explaining what the language looks like and how it behaves. These papers can serve to distinguish between the new language and other languages. The documentation often focuses on problems in software development and how the new language can solve them where others have not.

This paper serves as an introduction to the problems surrounding design claims for a new programming language and the difficulties developers and language designers may encounter when lacking an understanding in the aim and purpose of a language’s existence. This lack of understanding can have economic consequences for both companies and developers that seek to invest money and time in a programming language.

1.1 Problem definition

Java is said to be one of the most popular languages in the world [1]. Go is Google’s new programming language with its first release in March 2012.

If Java is so popular, why do we need a new language? How is a new language justified and is it enough? The problem I see is that programming language researchers is unaware of their importance in the area of language design that is conducted outside of the walls of academia.

The questions I intend to answer:

- What reasons justify the creation of a new programming language? Do the makers of Go provide enough reasons for its existence, and did the makers of Java provide enough reasons for theirs? If not, why do they exist?
- Is it possible to evaluate if the expectations on a language have been met?
- Are there economical gains involved in owning a language?

2 Methodology

I have chosen two languages and examine their design claims. The reason for this is to find out the purpose of the language designers and how they have chosen to communicate their ideas and design decisions. This is mainly to get the idea of the designers and their investors point of view.

I have looked at Go’s official website golang.com and read its ‘frequently asked questions’ page. I have not found another document that provides information
about the authors design decisions and purpose of Go but I believe it serves as a valid source for the designers point of view.

I have also looked at the Java environment’s white paper written by James Gosling and Henry McGilton from 1996, which provides the technical description and explanation of the design of the Java programming language. The white paper was written a year after Java’s launch and thus Java had already become fairly accepted as a programming language and adopted by a number of programmers. I do not see this as a problem for my work because it is how they back up their claims and formulate their problem definition that is important for this paper.

Since Java was created and funded by the hardware company Sun microsystems, I will attempt to examine the possibility for economical gain in developing a programming language. By reading the history of Java, I will try to find a correlation between Java and Go with the knowledge that Google Inc is standing behind the Go project. Therefore, seeking the possible economic profit in having a programming language to its brand.

I am not building my thesis on any scientific method except source criticism. My background ‘The criticism against language design’ (section 4.3) and ‘Software Science neglect of human factors’ (section 4.2) is building mainly on two papers, one by Shane Markstrum, Professor at Bucknell University and the other by Stefan Hanenberg, Professor at University of Duisburg-Essen. It is from their discoveries I am building my argumentation on the lack of scientific research methods where the human factor is central or partly.

2.1 Limitations

I have chosen to limit this analysis to two programming languages. One new, Go, and one established, Java. The reason I chose Go is because it is a new language which has already earned some attention in the development community but also because of the financial support it has received from Google which I find interesting. The choice of Java is mainly because of its popularity and its good documentation (the white paper). The history of Java can possibly help the discussion of the economical prospect in owning a programming language. It can also explain why a new language was needed in a time where the computer landscape was different then now.

The background is based on how the history of programming languages and programming language theory was in America, thus it is possible that there is other aspects of this from other parts of the world.

The definition of programming language that I have used includes general-purpose languages and not domain-specific languages, since they have no bearing on this paper.
3 Definitions

3.1 Programming language

A programming language is defined by Anthony A. Aaby as: "A programming language is a notation for writing programs" [2].

3.2 Domain-specific languages (DSL)

A domain-specific language is the opposite of a general-purpose language. A DSL is designed to work within a specific application area. Two examples are HTML, which is a DSL and a mark-up language, and SQL, which is a programming language used for data management in a relational database [4].

3.3 General-purpose programming language

General-purpose languages are the opposite of domain-specific languages. A general-purpose language has no specific application area and thus it is possible to write software in many different application domains. Examples are C, Java, Python and Go [5].

3.4 Programming paradigm

A programming paradigm is a way of describing abstractions from different types of elements of a program, like objects and functions, and the steps in a computation, like assignments and evaluation [6].

3.5 Core language and standard libraries

Most programming languages have a standard library. Standard libraries can contain all sorts of facilities for the programmer. Usually it is preferable to keep the standard library small and basic and then additional libraries can be used as optional tools. Most common features of modern languages are mechanisms for input and output (I/O) instructions, data structures and algorithms, like sorting algorithms. The core language is the programming language definition plus its standard libraries [7] [8].

3.6 Open-source

Open-source is often used as a term when software source code is available for free for everybody to use, usually via the Internet [9].
4 Background

4.1 The history of programming languages

Serious investments were made in computer development in the 50’s, due to the Cold War [11]. The field of Computer Science began to establish itself as a distinct field of academia. During this early stage in the modern history of Computer Science the field of programming language theory became important in order to decrease time spent on programming the new machines and to make the most of their capacity. Languages such as FORTRAN, LISP and COBOL were created mainly from the funding of DARPA and IBM [11]. In the 70’s the computer industry started to shift. The invention of microchip led the way to the creation of personal computers and new types of computers could be funded by commercial interests from private consumers [12]. An important language for modern programming language and software development is the C language. It was part of the development for the Unix operative system funded by Bell Labs [13].

During the 90’s the development of a programming language was no longer necessarily a matter of heavy invested research. It was not either necessary that a programming language was coupled to the usage of hardware or for scientific research or purpose. Languages like PHP, Ruby, and JavaScript were constructed and designed by single or a smaller group of non-professionals. It had also become more important for language development and a language’s survival to be open-source, meaning that people outside the inventor core were able to contribute to development and improvement of the language [15].

4.2 Software Sciences neglect of human factors

Stefan Hanenberg published a paper, *Faith, Hope, and Love - A criticism of software science’s carelessness with regard to the human factor* [14]. The paper argues for empirical methods in programming language research that relies on studies of developers. Hanenberg classifies two approaches of research in the field of programming language theory and software development; the first one is the technical approach, where software is either examined in an analytical way or an experimental way. Software is then the subject or the input and mathematics and statistics are the research methods. The second approach is the socio-technical, where the subject now is the developer and its construction of software with new artefacts and the method is empirical [14]. It is the second approach that is interesting when evaluating a new programming language since it seems that there is certain goals in language design that is desirable where the human factor plays a central role (see section 4.3).

However, Hanenberg finds difficulties in applying the socio-technical approach on the evaluation of software artefacts and that the difficulties is mainly because how the existing methods classifies the subject. He investigates the possibility of borrowing research methods from other fields where humans are the subject. He
tries with medicine and psychology but also mentions social science, pedagogy and human-computer interaction.

Hanenberg does not think that the socio-technical approach is the only method needed for evaluating a programming language or a new software artefact. However, if an evaluation is going to answer some questions where the human factor is central or partly central then there must be a underlying study that take the impact of humans in consideration [14].

4.3 Criticism against language design and how it is conducted

Crista Videira Lopes, professor at University of California[15], states in a blog post, the goals of language design; Performance, Human Productivity and Verifiability. In the performance goal she puts the need for languages to catch up with hardware innovations, such as multi-core processors. Videira Lopes says that "The Internet are full of articles claiming improved software development productivity for just about every other language. No evidence is ever provided..."[15]. In a paper written by Shane Markstrum, Professor at Bucknell university, he addresses the problem Videira Lopes says as 'lack of claim-evidence correspondence' [16]. Markstrum’s report focuses on the history of unsupported design claims for programming languages and he discovered a lack of fact-based claims from an early state in programming language history. He concludes that this behaviour is unacceptable for the health of the research community [16]. Videira think that language designers should let go of claims altogether and stress the importance of being honest when doing scientific work [15].

Since the construction of low-level languages like C, the area where language design is conducted has expanded [15]. Language design is no longer an occupation reserved only for well funded research groups. Many new languages have been made by single or small groups of hobbyists (PHP and Ruby) and engineers at computer companies (Java, .NET and Go) [15]. The type of software and the organization of software development has also changed and extended since the 60's covering web applications and phone applications and new development methods like SCRUM and XP [17].

4.4 Hardware development - Multi-core processors

Up until the beginning of this millennium processor frequency increased at the same pace as the number of transistors put on chips. The latter is referred to as Moore’s law and the speed is ’transistor doubling’ every two years [18]. The higher frequency leads to faster computers but has then reached a plateau. The measure taken to keep the development of faster computers was to put multiple processors on the same chip providing single processors working together leading to better performance but run on lower frequencies [18]. This multiple processor chip solution is called multi-core processors. In 2009 IBM revealed its research chip for cloud computing, SCC, with 48-cores on one chip [19]. In an article by EE Times from the Multicore Expo
conference in 2006 the panellists generally agreed that the future seems bright for multi-core processors although they stated that software development is different, for example, debugging software becomes different because it is not possible to simple stop the processor, since others will continue [20]. Tomas Evensen, CTO at Wind River Systems says at the conference "To use multicore, you really have to use multiple threads," [20]. Evensen says that if you are not well experienced with programming with multiple threads then "...there are lots of ways to shoot yourself in the foot. The bugs you introduce with multi-threading are so much harder to find."[20]

5 Go

5.1 Go the programming language

Go is a programming language officially announced in 2009 with the first stable release, Go 1, announced in March 2012. Go is available as open-source on the golang.com website [21]. Go was initially created in 2007 as a 20% -project1 of Robert Griesemer, Rob Pike, and Ken Thompson at Google Inc [22]. It is a system programming language that is said to "combine the ease of programming of an interpreted, dynamically typed language with the efficiency and safety of a statically typed, compiled language."[25]. The Go language provides concurrency with primitives called 'goroutines' and 'channels' [23]. Concurrency is when multiple computer agents interact with each other in different ways. For example accessing shared resources and communication between each other [24]. The Go designers ideas with concurrency are derived from Hoare’s Communicating Sequential Processes, CSP. A goroutine is "...an independently executing function, launched by a go statement."[23]. Unlike threads, the goroutines are stacked onto a set of threads enabling communication between goroutines with channels.

Go’s choice of syntax is developed from the C family, and the declarations as well as the use of packages come from the Pascal/Modula/Oberon family [26]. Go is a compiled language. It does not run on a virtual machine and it is not interpreted. It allocates memory at runtime and keeps track of it so it can be freed when appropriate [27].

5.2 Purpose of Go

"There are many things in the Go language and libraries that differ from modern practices, simply because we feel it's sometimes worth trying a different approach."[28]. On Go’s frequently asked questions page, the authors of Go state that the project was grown out of frustration towards programming languages and environments of today. They state that the computing landscape has changed over time but yet no

---

1Google employees are encouraged to take 20% of their work time to spend on projects of their own.
major system programming language has emerged in over a decade. On their web page they state five trends which are listed below [29].

- "Computers are enormously quicker but software development is not faster."
- "Dependency management is a big part of software development today but the “header files” of languages in the C tradition are antithetical to clean dependency analysis—and fast compilation."
- "There is a growing rebellion against cumbersome type systems like those of Java and C++, pushing people towards dynamically typed languages such as Python and JavaScript."
- "Some fundamental concepts such as garbage collection and parallel computation are not well supported by popular systems languages."
- "The emergence of multicore computers has generated worry and confusion."

The designers of Go claim that programming nowadays has become more difficult and that programming languages are partly to blame. They say that efficient compilation, efficient execution and ease of programming are not available in the same multi-purpose language. To counter this problems they call for a new language which address linguistic issues such as "expressive but lightweight type system; concurrency and garbage collection; rigid dependency specification; and so on" [25]. They claim that Go is an attempt to solve this issue and that "In every respect the language was designed by thinking about what programmers do and how to make programming, at least the kind of programming we do, more effective, which means more fun."[30].

6 Java

6.1 Java the programming language

Java, the programming language, is an object-oriented language targeting end-users for development of application software. The first release of Java was presented in 1995 [31]. Java was initially owned by Sun Microsystems, which was in turn bought by Oracle Corporation in 2009 for 7.4 billion dollars [32]. In a press release from Oracle it states that 'Java is one of the computer industry’s best-known brands and most widely deployed technologies, and it is the most important software Oracle has ever acquired’ [32]. The reason for Oracle’s investment is also because their own fast growing business, Oracle Fusion Middleware, is built on Java technology meaning they now have full control over Java’s further development [32].

Java goes under the lead words "write once, run anywhere" which means that the source code you write is platform independent and can be executed on all types of computers if they support the Java virtual machine, JVM [33].
The language is object oriented from the ground up, meaning that even primitive types for instance int, boolean and char have their corresponding wrapper classes Integer, Boolean and Character. Java’s standard library is the java.lang package which contains all classes that are fundamental to the Java design. Java is strictly hierarchical which means that all classes build on the roots class hierarchy. For example the Object class is the root class in the java.lang package and all the other classes in the lang package inherit the methods from Object, even Integer, Boolean and Character.[34]

6.2 Purpose of Java

The purpose of Java is explained in the so called white paper written by James Gosling and Henry McGilton in 1996[35] where they claim that "The design requirements of the Java™ programming language are driven by the nature of the computing environments in which software must be deployed."[35].

The design goals of Java are to be simple, familiar, object oriented, multi-threaded and interpreted [35]. It aims to be secure and reliable since the purpose of Java is to operate in distributed environments [35].

Gosling and McGilton state that "the fundamental concepts of Java technology are grasped quickly; programmers can be productive from the very beginning" [36]. Java is said to be familiar because it has inherited a C++ look (C++ was the language of tongue at this time[35]) so the idea of adopting a C++ syntax was to attract C++ programmers to try out Java [35].

Java is said to be more secure than other languages because its compiler catches a lot of errors at compile time, in addition the Java runtime environment catches errors at runtime. This means that it will prevent many program crashes caused by bad or forgotten error handling. The lack of pointer arithmetic in the language reduces the risk of pointing to wrong or non-existing addresses that can also result in program crashes.

Gosling claims that "language features guide programmers towards reliable programming habits" [36].

The removal of manual memory management "eliminates entire classes of programming errors that bedevil C and C++ programmers" [36].

According to the design principles, Java is said to be simple since it does not need 'extensive programmer training' [36]. The designers also say that 'the fundamental concepts are grasped quickly' so that programmers can be productive quickly. In the white paper written by Gosling, there is a chapter about the reasons for the decisions to leave out some of the things in C and C++. They claim that in C and C++ there is a lot of context you need to understand to be able to grasp another persons code. They also claim that the usage of these C features results in a 'new programming language that is incomprehensible to anybody other than its creator, thus defeating the goals of good programming practices'. An example of this is the related header files and the #define. Instead of using the C++ header files, in Java, you declare other classes and methods directly in the Java source file with
the import declaration. The usage of classes removes the need for structures and unions like those in C and C++.

Threads are used as a way of scheduling the computer’s CPU into different processes. When using threads, one has to be careful in handling concurrency in the computer’s memory and registers meaning that threads have to deal with data in an ordered way. This can lead to ‘hard-to-find’ bugs [37]. Threading can be done with C and C++ by accessing the native threading API. What was new with Java was that the threading usage was abstracted into the core language with the class Thread available in the standard library.

Java is both a compiled and interpreted language. This means that Java source code is compiled into Java byte code which then is interpreted by the Java Virtual Machine. It has the same features as other compiled languages but the source code becomes portable since its compiled byte code can be interpreted by the JVM.

6.3 The story behind Java and why it became so popular

In 1991 James Gosling, Mike Sheridan, Patrick Naughton and ten other programmers started working on a secret research project separated from the rest of the Sun Corporation [31].

The group worked under the name: Green project and the idea was to foresee the next big thing in technology and hence be the first to deploy it [31]. The focus fell on designing a system that would combine and control all new types of home electronics, an interactive remote control [31]. This was a new market that was different from the traditional computer business, where big money was spent on the most powerful and accurate engineered components [31].

The Green project group needed to establish some important key concepts that were different from what the traditional work station programmer were accustomed with. For example, in the consumer electronics market “consumers demanded low-cost, bug-free and relatively simple, easy-to-use products.”[38]. The project group also stated that the product needed to be backwards compatible since components and hardware would change with time [38].

To meet these requirements the team established that they needed a platform-independent process for development. Gosling, who had previously created a pseudo-code virtual machine for Pascal programs during his school years in the 80’s [39], was not late in seeing the need of a new language that could be supported on all types of software and executed on all types of hardware.

The group started by extending the C++ compiler. After a while, this extension did not meet the requirements and the idea of a new programming language started to take form [31]. "All along, the language was a tool, not the end," Gosling says. "This was nice in a number of ways. The goal was never 'Let's take on C++,' [but] to build a system that would let us do a large, distributed, heterogeneous network of consumer electronic devices all talking to each other."[31]. "In 18 months, we did the equivalent of what 75-people organizations at Sun took three years to do" says Naughton [38].
The launch of the interactive remote control did not go as planned but the management at Sun had seen the potential of the programming language, at the time called Oak, and looked around to see if it could be useful elsewhere. Their eyes fell on interactive TV and the Green project was dissolved and restructured into another group called FirstPerson, but the market for TV business was premature and no one saw the potential in TV on demand [31].

At the same time, the Internet grew popular and new types of interactive web browsers were constructed, such as Mosaic later reconstructed into Netscape [40]. The team thought that the growing web could be a potential bearer of the platform independent language. "We released the source code over the Internet in the belief that the developers would decide for themselves,..." [41] says Lisa Friendly, FirstPerson employee and original member of the Java Products Group.

One of the reasons why Java became one of the most popular programming languages is because of Java applets. A Java applet is a small program that can be embedded in web pages. The applets enable programmers to make a more interactive web page with graphical and animated little programs. The applets enhanced the Java language claim on being safe. The applet is downloaded from a server and its code executed in a 'sand-box' environment on the browser giving limited access to the client’s file systems and local storage [42].

7 Discussion

The area where language design are conducted has changed since the 60’s. Programming language projects are more often than before founded as side projects, Go as a 20%-project and Java as a tool for another artefact. Many of these languages are further developed in an open-source community, enabling input from other developers. At some point a few of these projects are picked up by companies for example Google and Sun. I have not found a reason for a company to spend time and money on an open source project. The exception being when Oracle bought Sun with the explanation that Java was important for gaining control over the language that their software ran on.

A hypothesis that I will address as a suggestion for further research is that when a language gains adoption and software is built in that language, a language owning company gains a solid ground for their business in terms of power. Thus owning a language could potentially be worthwhile in marketing terms and the company’s position in industry.

The knowledge of where language design is conducted shows us that it has moved from being largely invested projects held mainly by scientists to hacks by engineers and developers. This leads to another type of responsibility for language theoreticians: to evaluate the impact of new artefacts on the area of Computer Science. Right now it is up to the developers and users to investigate if the new language is something that they need or not. This leads to an adoption of languages mostly based on people’s beliefs and thoughts.
If a developer seeks to find a language that is ‘easy to learn’ there is no scientific research done on humans thoughts of ‘easy to learn’ programming. For example, as Hanenberg emphasized, who are the subject here? a professional developer or a beginner. What type of programs will be written in the language and thus have the subject already experience in that field?

When Java states that their language is ‘simple’ and ‘easy to learn’ they seem to explain why, for example ‘Java is said to be simple since it does not need “extensive programmer training”’. However, there are no facts explaining the meaning of ‘extensive programmer training’, again who are the subject? The lack of knowing who the subject is means that the reader has nothing to relate to and thus needs to conduct a lot of own research to take a stand on whether to trust the designers claim or not.

The reasons for justifying Java are not sufficient even though the authors try to explain their decisions. I believe that this means that the designers of Java would have welcomed studies and surveys that they could have used in their argumentation for Java. In retrospective we can see that Java has and is an important language. However, this can not be seen as correlated to its design claims but most certainly to their aims on helping developers writing code with an object-oriented language.

The lack of observed human factors in scientific research methods for evaluating programming languages, if believing Hanenberg, Videira and Markstrum, means that we are left uncertain of the consequences a new programming language generates.

Go does not claim to solve problems, it aims to solve problems. This approach corresponds to the attitude that Videira wanted from language designers, which she thinks leads to a healthier way of designing and encouraging innovation in that area. However, Go’s designers do make claims in their problem definition for example they express in terms of ‘growing rebellion’ and ‘worry and confusion’. They also claim that software development has not become faster in the same pace as hardware development. There are no facts backing up these claims provided by the authors thus the reader is left with taking a stand on whether to trust them or not.

I too have discovered that there is a problem in addressing multi-threading on multi-core processors enabling developers to write software that maximize the usage of the hardware. If Go aims to provide a solution with their language, then yes I think it justifies Go’s existence. However, how can we know that the language designers have succeeded in their aims with Go if there is no method of testing if a ‘programming language aids a developer to write software that is sufficiently parallelized’.

It is possible that it do exist scientific research methods to be used for this problems (evaluation and problem definition). However, Hanenberg’s paper stress that the ones that do exist is hard to apply which can explain why they are not used for backing up languages design claims. My interpretation is that the lack of scientific research in industry work is most likely an economical question (take too much time) but I believe that companies are taking a risk if they do not understand what problem their language seek to solve, in terms of time and money invested in
language projects. Therefore I think that language owners would welcome research and research methods in evaluation of programming languages.

However, it is possible that companies have other purposes with owning a language than contribution to language development and human productivity. For example I see a potential power position for Google if Go is to become the language for parallelized programming and that is for the next generation of operating systems. If developers find Go as a good choice for writing multi-threaded software then the operative systems for multi-core computers will most likely be written in Go code too, thus giving Google a quite solid ground for its brand.

But again, it is all about what developers in the end choose to use and if we would conduct research on the human productivity when programming in a certain language. Investors in programming languages could more accurately know what people want and thus maximize the chance for them to become the language of choice by developers.

8 Conclusion

We need new programming languages if we know what problems they seek to solve and how they aim to do this. If we can conduct research using scientific methods we can more accurately argue about problems and thus ease the designers on how they aim to solve them. Well chosen scientific methods would also evaluate designers decisions and thus it will serve as guidance for developers in their search for a programming language.

The reasons for Go justifies Go’s existence because the designers aim to solve problems. They do not know if they will succeed and thus they let the reader know that they are aiming for a solution. However, their problem definition is built on beliefs which cannot be seen as sufficient in scientific eyes to justify the purpose of their work. Java’s design claims do not justify its existence but the attempt of the designers to try to prove them serves as an important argument for the need of proper scientific methods for evaluating programming languages.

I have not found useful material whether there is an economic gain in owning a language, although my conclusions are that it is likely for advertising purposes of the owning companies brand and the control of a language can be important to ensure the development of the company’s software in that language. I have addressed this as a possibility for further research.

9 Reflections on my own work

I highly underestimated the complexity in this discipline and I did not know much about programming language theory and design.

I am not very familiar with scientific research and scientific writing thus I see this work as an introduction to this area of science and as a learning process for myself.
I think the area is interesting and important for the community between language designers, developers and scientists and thus I hope my work can at least spark interest in further discussion and/or research.

### 9.1 Source criticism

I found during this work process that valid sources were difficult to find. This can possibly be explained by my lack of experience in writing scientific papers.

It seems that describing recent history is more difficult than older history. I can imagine that it is because of the involving people are still alive and thus remembering everything from their own point of view. With Internet as a medium everybody can write whatever they want without a publisher or other reviewers. I have therefore chosen Wikipedia as a source even though it is not consider as a respected source for scientific research and I agree. However, I believe it can be more accurate in terms of less objectivity when it comes to this type of 'recent history'. Due the number of active reviewers at Wikipedia it can create more objective information.

TIOBE Programming Community Index is based on the search engines: Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube and Baidu. The index is an indicator on the popularity of programming languages and thus it is not a scientific proved index I believe it gives enough support for claims about a language popularity.

### 9.2 Ideas for further research

One approach is to investigate more deeply what the process of designing a language looks like. At what events does a designer call for a new language? What research does the designer do to find reason for a new language? What expectation does a designer have on the language and when does she/he know that the expectations have been met?

Another idea is to see how a language has developed in time. Many changes have been made in the Java language from its first release to the seventh. Comparing that with the initial design claims could determine if the claims are valid or not, or if they have been used as a guidance for development. This could be a help for designers who want to market their language to the language developer community.

I did not find sufficient information about the reason for a company to invest in a language project. If these facts are to be found it could potentially serve as a help when finding scientific methods for evaluating a programming language. If a company can evaluate a language in terms of profit, then scientists could potentially use that information for own evaluation methods.

I would like for researchers in other fields to interest themselves in the problems that the usage and expectations from programming language have for example, economical and marketing research on the question of how a programming language should be marketed and how it can bring profit for a company.
Bibliography


[3] [Swedish] programspråk, [webpage], Nationalencyklopedin, seen at [2012-08-27], available at http://www.ne.se/lang/programspråk


[9] [swedish] öppen källkod, [webpage], Nationalencyklopedin, seen at [2012-08-27], available at http://www.ne.se/öppen-källkod

[10] [swedish] imperativa programspråk, [webpage], Nationalencyklopedin, seen at [2012-08-27], available at http://www.ne.se/imperativa-programspråk


[12] [swedish] mikroelektronik, [webpage], Nationalencyklopedin, seen at [2012-08-27], available at http://www.ne.se/lang/mikroelektronik


[29] Go’s Frequently asked questions, [webpage] Build version go1.0.1., seen at [2012-06-01], available at http://golang.org/doc/go_faq.html#what_is_the_purpose_of_the_project


[40] Computer history museum exhibits timeline over the computer history, [webpage], seen at [2012-06-01], available at http://www.computerhistory.org/timeline/?year=1994

