Literature Survey of Methods for Comparing Programming Languages

David Colgan

December 6, 2010

1 Introduction

Most computer science majors and software developers have used Java and C. According to the TIOBE Index (`http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html`), these two languages have consistently been among the most popular for general use. They are all established, well understood, and use the object oriented or imperative paradigm.

Most of the other top languages are fairly similar to Java and C. Languages like C#, PHP, Python, Perl, and Objective-C all use some combination of the procedural and object oriented paradigm. But are procedural and object oriented languages the best computer science has to offer for creating reliable, high performing software on a budget? Many lesser-known languages, a good number of which have heavy influence from the functional paradigm, claim increased programmer productivity, fewer errors, shorter programs, and greatly enhanced support for multicore processing.

2 Research Goals

This project seeks to determine if Clojure, Forth, Erlang, Haskell, or J are compelling alternatives to the common procedural and object oriented languages most often used today in commercial environments.

These five languages are ones I deem interesting and have wanted to learn in the past. They are also all somewhat to very mind-bending, and several require a completely new approach to programming when compared to Java or C. The languages are:

- Clojure, a Lisp dialect on the Java Virtual Machine `clojure.org/`
- Forth, a stack-based language `www.forth.org/`
- Erlang, a concurrency-oriented language `www.erlang.org/`
- Haskell, a lazy, purely functional language `www.haskell.org/`

- J, an array language similar to APL `www.jsoftware.com/`

If I somehow get done with these languages and have more time, I could investigate OCaml, Scala, Lua, or Groovy.

3 Literature Survey of Previous Work

At this time there appears to be two kinds of research available in the literature that compare programming languages: feature by feature comparisons and comparisons of small programs.

3.1 Feature Comparisons

Feuer and Gehani take a conceptual approach when they compare C and Pascal.[2] They begin with a history of the languages and discuss design decisions, followed by a step by step walk through each language's major features. They also evaluate C and Pascal for different problem domains. This paper is more of an informative overview than a hard empirical evaluation. The only actual code they show is a single function implementing a binary search.

In a comparison of Ada 95 and Java, Brosgol takes a similar approach,[1] going feature by feature through the two languages. He provides sample code snippets throughout. He arrives at a table of features, highlighting differences in syntax, program organization features, memory management, and OO features like inheritance, polymorphism, and encapsulation.

Nami presents a factual comparison of Eiffel, C++, Java, and Smalltalk[5]. He gives a brief introduction to each language, describing design decisions and history. He then classifies each language based on static vs dynamic typing, compiled vs interpreted build methods, built-in quality assurance facilities, automatic documentation generators, multiple vs single inheritance, and concludes with a brief discussion on each language's efficacy for building infrastructures.

Tang does a similar comparison of Ada and C++ using much of the same methods as the other studies.[7]

Many of these studies are a high level overview of various languages. They compare features, but do not give in-depth examples.

3.2 Small Program Comparisons

Perhaps more useful and interesting are those comparisons done by inspecting the same program written in different languages. These give concrete examples of the differences.

The method I follow for this project is closely related to the method used by Prechelt [6]. He compares C, C++, Java, Perl, Python, Rexx, and Tcl by having various computer science masters students and volunteers from newsgroups online write the same small program in one of the languages. He then compared the resulting programs based on program runtime, memory consumption, lines of code, program reliability (based on whether the program crashes or not), the

amount of time it took each programmer to write the program, and program structure. The program he had the participants write was a simple string processing program that consisted of converting telephone numbers into sentences based on a large dictionary and mapping scheme. Most of the programs he received were fairly small, taking a median of 3.1 hours to write, and averaging around 200-300 line of code.

Some of the more interesting results from Prechelt's study include the observation that in lower level languages, a lot of code is dedicated to writing the data structures, while in the higher level languages, the programmer usually takes advantage of the language's existing built-in capabilities. He also found that scripts tend to require about twice as much memory as C and C++, with Java taking 3-4 times as much, and that C and C++ are about twice as fast as Java and several more times faster than scripting languages.

Prechelt includes a discussion of the validity of his evaluations. He acknowledges the potential problems of asking for self-reported data from the Internet, as well as potential differences in programmer ability and working conditions. He suggests that the large number of people (80) who contributed programs balances out many of these problems, and that the results should not be trusted for small differences. He does assert that large differences are more likely to be accurate.

Henderson and Zorn perform a similar study in [4]. They compare C++, a well known language, with four lesser-known languages: Oberon-2, Modula-3, Sather, and Self. They also write a short program in each of the languages, a simple database for university personnel information. These are all Object Oriented languages, and as such, the comparison is weighted specifically towards OO features. They compare each language's methods of implementing and capabilities for inheritance, dynamic dispatch, code reuse, and information hiding. In addition to OO features, they also compare execution time, lines of code, and compile time. Henderson and Zorn explicitly state that one of the goals of their survey is to increase programmer awareness of lesser-known languages.

In a more informal study, Floyd compares C++, Smalltalk, Eiffel, Sather, Objective-C, Parasol, Beta, Turbo Pascal, C+@, Liana, Ada, and, Drool[3]. He collects an implementation of a linked-list structure from various people and then summarizes the results in a table that compares garbage collection schemes, inheritance (single or multiple), binding time, compilation (compiled vs interpreted), exception handling features, and lines of code. He simply enumerates the implementations and does not do further analysis.

I will combine these two approaches. The deliverables for my project will, like Prechelt, include implementations of the same program in several different languages, but I will also include high level feature comparisons between the languages as well.

4 Research Goals

This project seeks to determine if

5 My Work

I began the semester by implementing the Farkle and GA system in Python, the language I know best, so that I could figure out the general idea of how the program would be implemented. I programmed in a mostly Object-Oriented manner, and came up with a pretty good implementation of the system.

After completing the Python version, I then rewrote the system in Clojure. Clojure is a language I have not had experience with, though I have used other Lisp dialects. In my past experience with Lisps, I did not fully understand or apply the functional style. Having completed the Farkle and GA systme for Clojure, I now have two versions of the same system written in different languages and different paradigms, but which do almost exactly the same thing. This allows me to make some interesting comparisons.

5.1 Lines of Code

The Python code comes in at around 900 lines of code excluding tests, while the Clojure code is around 600 lines, a substantial difference. However, it is very possible for me to reduce the Python code in size if I cleaned it up further. It had the disadvantage of going first.

5.2 Miscellaneous Observations

One very cool side effect of rewriting the system in a second language is that I discovered better ways to write the Python code as I wrote the Clojure code. Even though Python is predominantly a procedural and object oriented language, it has very powerful functional capabilities as well, and some of the ways that Clojure forced me to use a functional style could also be used in the Python code to make it more readable and shorter.

5.3 Learning Clojure

To begin the process of learning Clojure, I began by reading the book Programming Clojure by Stuart Halloway. After going through this book, I was a bit underwhelmed by my level of understanding. I got the basics, but I still didn't feel ready to begin implementing the Farkle system. A few of the other resources I used included a very well-written Clojure tutorial found at `http://java.ociweb.com/mark/clojure/article.html`, the official Clojure website at `http://www.clojure.org`, the very helpful community question and answer site `http://www.stackoverflow.com`, and other random pages found through Google searches.

5.4 Disadvantages of Clojure

Clojure is plagued by a few problems. The most obvious is it's close integration with Java. While this is good in that it can use Java libraries, it also inherits

many of Java's problems. Clojure is not a very good language for scripts or anything that requires fast startup. Starting any Clojure program cold takes on the order of 10 seconds. For this reason, when developing Clojure it is recommended that you have one running Clojure repl and continuously send commands to it.

The second major problem with Clojure is simply its age. The language is only a few years old, and it is rapidly evolving and getting better. However, even core features of the language have been implemented just this year, such as protocols.

A bigger problem is that because the language is so young, the tools are not very mature either. Slime, the flagship editing environment for all kinds of Lisps built on Emacs does not support Clojure nearly as well as the more mature Lisps. A case in point is that the read-line function, the primary way of getting command line input, simply doesn't work in Slime. This created a problem for my command line program. The Slime debugger is also not very helpful when working with Clojure. Stack traces will show 100 levels of Java method calls and a single location in my own code where the error occured.

There is a system developed to write Clojure in Vim called VimClojure, but I could not figure out how to install it completely. After spending multiple hours on it, I eventually went back to Emacs.

There are also Eclipse and Netbeans plugins that I did not investigate.

6 Comparison Table

	Python	Clojure
Purity	Impure	Pure by default
Supported Paradigms	Procedural, Object Oriented, Functional	Functional, Object Oriented
Evaluation Strategy	Eager	Lazy
Execution Method	Interpreted	Byte-code Compiled

References

[1] Benjamin M. Brosgol. A comparison of the object-oriented features of ada 95 and java. In *TRI-Ada '97: Proceedings of the conference on TRI-Ada '97*, pages 213–229, New York, NY, USA, 1997. ACM.

[2] Alan R. Feuer and Narain H. Gehani. Comparison of the programming languages c and pascal. *ACM Comput. Surv.*, 14(1):73–92, 1982.

[3] Michael Floyd. Comparing object-oriented languages. *Dr. Dobb's J.*, 18(11):104–ff., 1993.

[4] Robert Henderson and Benjamin Zorn. A comparison of object-oriented programming in four modern languages. *Softw. Pract. Exper.*, 24(11):1077–1095, 1994.

[5] Mohammad Reza Nami. A comparison of object-oriented languages in software engineering. *SIGSOFT Softw. Eng. Notes*, 33(4):1–5, 2008.

[6] Lutz Prechelt. An empirical comparison of seven programming languages. *Computer*, 33(10):23–29, 2000.

[7] L. S. Tang. A comparison of ada and c++. In *TRI-Ada '92: Proceedings of the conference on TRI-Ada '92*, pages 338–349, New York, NY, USA, 1992. ACM.