Exploring major design features in selected HLLs (Java, C++, Python)

Julian Garcia
New Mexico Institute of Mining and Technology
Department of Computer Science
801 Leroy Place
Socorro, New Mexico, 87801

julian.garcia@student.nmt.edu

ABSTRACT
This article draws comparisons between the programming languages: Java, C++, and Python. Each of these programming languages are actually very similar in terms of the purpose of their design, as each is generally made to cater to as many types of users as possible, especially beginner programmers. However, to make distinctions and to draw comparisons between each language, Power, Abstraction, Security, Robustness, Efficiency, Platform Independence, and Readability will all be evaluated for each programming language. In evaluating these categories of performance, this paper will dissect the advantages and disadvantages to each programming language. Ultimately this will help determine which domains of use each programming language each language is more suited for, despite the fact that each language is made for more of a general domain of use than most other programming languages..

Categories and Subject Descriptors
D.3.3 [**Programming Languages**]: *ALL*

General Terms
Design, Reliability, Experimentation, Languages, and Theory.

Keywords
Usage Domain, HLL, Object Oriented Programming, Memory Utilization, Programming Languages

1. INTRODUCTION
Programming languages are a method of communication between a user and the machine, as the user uses the programming language to help specify certain directions within that language in order to reach a desired output. In drawing comparisons and contrasts in different programming languages, it's important to keep in mind that no one language is "better" than the other. Each language needs to be evaluated on specific merits and categories to classify which domain of use it's meant for. In this paper, Power, Abstraction, Security, Robustness, Efficiency, Platform Independence, and Readability will all be evaluated for each language to draw comparisons between each language. It's also important to note that no one language can encapsulate every desired use, as in gaining merit in one category it may lose in another. [1]

2. POWER
It's difficult to achieve a complete definition of power, as there are multiple ways to achieve "power" in a programming language. For the purpose of this paper the most important methods of obtaining power in a programming language will be looked at. To increase power a language needs to do any or all of the following:

1) Allow powerful semantics in a short amount of syntax, in essence allowing a large amount of "work" to be done with little code written.
2) Provide methods/tools that will let the programmer do as much with the programming language as they can imagine.
3) Allow for recursive statements. Allowing recursion provides the power to solve a vast amount of problems with much less code.
4) Allow for polymorphism, by allowing polymorphism you're providing the ability to overload and granting genericity to your programming language.

2.1 Abstraction
Power tends to be interdependent on abstraction. In increasing abstraction, which is in essence "hiding details from the user", you provide improvements such as modularity, and you begin to facilitate the separation of the domains between the user and implementer. In gaining abstraction such as modularity, you can increase the power of the system, however there are instances when the separation between the domains between the user and implementer can reduce the power provided by the language. As each programming language being evaluated in this paper are Object Oriented HLL's , they all inherently have a high level of abstraction.

2.1.1 Java
The main two methods of achieving abstraction in Java are through the use of Java's Interfaces and Abstract Classes.

2.1.2 C++
The main two methods of achieving abstraction in C++ are through Classes and Headers.

2.1.3 Python
Python utilizes abstraction mainly through Abstract Classes and Methods.

2.1.4 Critique
C++ grants the least abstraction but the most power of the three programming languages, and Python grants the most abstraction.

3. SECURITY
Security in programming languages relates to capturing any mistake made in the code and preventing incorrect code from being executed as well as incorrect outputs. In increasing security

power is typically sacrificed, this is due to the correlation between power and insecurity. Powerful aspects of a programming language such as overloading allow for vulnerabilities in the security of a language and as such reduce the security of a language. As each of the HLL's being evaluated in this paper are HLL's designed to be used by inexperienced programmers, each of these HLL's have a high level of security as is. In evaluating security, it may be better to evaluate what a programming language *cannot* do, or more specifically, what it restricts the user from doing.

3.1.1 Java

In evaluating security in java, it's important to go into detail about Java's lack of pointers. Instead of pointers, Java has references, and there are four types of references, each type dependent on the type of relationship it has to the garbage collector which is specific to Java. The garbage collector essentially eliminates unreachable objects for the sake of freeing up memory on the heap. This garbage collection ensures that any reference made is always deleted and as such Java has implicit garbage collection. This saves the programmer from making the errors that are common in explicit garbage collection and ensures Java remains a robust HLL.

3.1.2 C++

The biggest concern to security in C++ comes from its use of pointers. C++ uses explicit garbage collection, each use of memory must be explicitly claimed back by the programmer. This reduces the robustness of the programming language as it trusts that the user will correctly reclaim whatever they use.

3.1.3 Python

Python doesn't contain the ability to manipulate pointers or anything similar to that functionality. Python as a language seems to be designed with robustness in mind above all else. Python as a language was designed for simplicity and for development ease of use. As such it was designed to keep programmers from making critical errors.

3.1.4 Critique

Of the three, C++ offers the least in terms of security, as C++'s focus on power and efficiency allow the user to implement insecure code of multiple forms.

4. ROBUSTNESS

In essence, robustness describes the software reliability of a programming language.. In increasing robustness power is typically sacrificed, this is due to the correlation between power and insecurity. Run time robustness is critical in achieving a secure programming language, and as such it is highly correlated with security.

4.1.1 Java

In evaluating the robustness in java, it's important to go into detail about Java's lack of pointers. Instead of pointers, Java has references, and there are four types of references, each type dependent on the type of relationship it has to the garbage collector which is specific to Java. The garbage collector essentially eliminates unreachable objects for the sake of freeing up memory on the heap. This garbage collection ensures that any reference made is always deleted and as such Java has implicit garbage collection. This saves the programmer from making the errors that are common in explicit garbage collection and ensures Java remains a robust HLL.

4.1.2 C++

A big issue with C++'s robustness comes from its inclusion of template metaprogramming [5]. Template metaprogramming essentially the "process of writing template based C++ programs that execute during compilation" [5, p.233]. Since these programs execute during compilation, it makes the programs much harder to debug and much more complicated to understand.

4.1.3 Python

Python doesn't contain the ability to manipulate pointers or anything similar to that functionality. Python as a language seems to be designed with robustness in mind above all else. Python as a language was designed for simplicity and for development ease of use. As such it was designed to keep programmers from making critical errors.

4.1.4 Critique

In this criteria, it's easy to implicate C++ as the least robust of the three languages.

5. EFFICIENCY

Efficiency tends to come at a cost of security/robustness. In essence, efficiency relates to the speed of execution of a program, independent of how the programmer writes the program. Efficient programming languages tend to make multiple sacrifices in fields other than security; abstraction and readability are typically hindered as efficiency increases. Conversely, power is typically increased with efficiency. Each of the HLL's being evaluated in this paper prioritized and tailored to efficiency in vastly different ways.

5.1.1 Java

Java achieves its efficiency from its Just-In-Time (JIT) compiler and from its support for concurrency. The JIT compiler is a part of the Java Runtime Environment. The just-in-time compilation has the ability to skip over code sections at run time, which speeds up the program's execution (i.e interpretation). The JIT greatly improves the efficiency of Java programs by compiling bytecodes into native machine code 'just in time' for them to run. The Java Virtual Machine (JVM) calls the compiled code directly. Since the code is not interpreted, compiling does not require processor time and memory usage. In theory, a Java program may be able to run as fast as a native application due to this.

Java also applies dynamic object orientation. This means that what calls what and how is decided on the fly at runtime. This adds a lot of overhead to the program.

5.1.2 C++

C++ is intrinsically stingy with memory, a C++ struct has no memory overhead if there are no virtual functions. Smaller things run faster due to caching, and are also more scalable.

C++ is compiled to binaries, so it runs immediately.

C++'s original conception was based on Simula, which didn't have the same facilities as dynamic OOP and instead did all the work of defining what calls what and how at compile time, although C++ can be used in a dynamic OOP manner. Instead of putting methods into a lookup table, it does a trick called name mangling, which is essentially a hash function that creates a name that associates it with the class that defines it so the program can call it directly rather than via method dispatch. The flipside of this is that whatever polymorphisms you choose to implement are baked into the class itself at compile time; if you want to replace a method in a class, you have to subclass it and implement the method in the subclass, or directly edit the code library itself. C++ is strict about ensuring the programmer doesn't pay for what they don't use, which translates to a substantial speed-up at the cost of some flexibility.

5.1.3 Python

Python is an interpreted language and as such its efficiency suffers greatly. It also implements dynamic OOP and as such contains that same overhead that Java does, however it lacks Java's JIT and as such is even slower than Java.

5.1.4 Critique

In this category, it's relatively easy to see that C++ is the most efficient programming language of the three. Its greatest advantage comes from the fact that it is a compiled programming language, ditching the overhead that both Python and Java have. While Java's JIT makes up for a lot of the efficiency lost, it's still less efficient overall than C++.

6. PLATFORM INDEPENDENCE

Platform independence typically comes with HLL's and as such is related to abstraction. To be platform independent, a programming language must be portable between different operating systems, its code must be able to be used on multiple operating systems at ease.

6.1.1 Java

Java is a platform independent language. Each Java program is compiled to bytecode which is then interpreted by the Java Virtual Machine to actually execute. Since Java is compiled to byte code like this, its portable between multiple OS's, as long as each OS has the JVM on it. The JVM itself is platform dependent, as each OS has a different JVM for it, but JVM is what grants Java itself platform independence.

6.1.2 C++

C++ is not platform independent, when a program is written and compiled in C++, the code is directly converted into executable code. This code is generated as a .exe file. The generated .exe file can run only on a specific operating system. For example, when a program is compiled in windows the .exe file can run only in windows and not in Unix.

6.1.3 Python

Python is also a platform independent programming language. Python programs are also compiled to byte code and they're to be interpreted by the Python Virtual Machine [2]. The only issue in it's independence comes in its modules, as some modules that can be used in Python are actually platform specific. An example of such modules are the Windows-only msvcrt module, and the Unix-only tty and termios modules [3]. To give an example of the implementation for one of these modules a program meant to detect a single keypress under Windows systems is included below:

```
"""keypress - A module for detecting a single
keypress."""

try:
    import msvcrt

    def getkey():
        """Wait for a keypress and return a single
character
    string."""
        return msvcrt.getch()
```

To implement this code in Unix the tty and termios modules are required.

6.1.4 Critique

The only outlier here comes in C++, as it is compiled to a platform specific code which makes it platform dependent unlike Java and Python. Java and Python both achieve their platform independence by compiling to bytecodes to be processed through their respective virtual machines.

7. READABILITY

Readability in a programming language is highly associated with its abstraction. Readability is essentially how easy it is for a programmer to interpret a programming languages syntax and semantics. Readability tends to come at a cost to efficiency and power as it tends to be associated with more lines of code and more syntax to parse.

7.1.1 Java

Java is a moderately readable code, the most confusion being granted by the need to import and define classes along with the dot notation in using the classes that are defined and imported. Below is a coding example for adding two numbers by taking user input in Java:

```
import java.util.Scanner;  // Import the Scanner class

public class Addition {
  public static void main(String[] args) {
    int firstNum, secondNum, sum;
    Scanner myObj = new Scanner(System.in);  // Create a Scanner
    System.out.println("Type a number:");
    firstNum = myObj.nextInt(); // Read user input

    System.out.println("Type another number:");
    secondNum = myObj.nextInt(); // Read user input

    sum = firstNum + secondNum;
    System.out.println("Sum is: " + sum); // Output user input
  }
}
```

7.1.2 C++

The biggest obstruction in C++ comes from its inclusion in pointers. Pointers in C++ are bound to confuse programmers due to pointers confusing nature as well as the near arbitrary syntax given to pointers. Assigning values to memory locations instead of actual values in itself makes the code much less readable. For example, this code prints the addition of two numbers using normal direct methods in c++:

```
#include <iostream>
using namespace std;

int main()
{
    int firstNum, secondNum, sum;

    cout << "Enter two integers: ";
    cin >> firstNum >> secondNum;
    sum = firstNum + secondNum;

    cout << firstNum << " + " <<  secondNumb << " = " << sum;

    return 0;
}
```

Now, here's the same code except using pointers to add the two:

```
#include <iostream>
```

```
using namespace std;

int  main()

{

  int  firstnum, secondnum, sum;

  int  *ptr1,* ptr2;

  cout<<"\n Enter two integers: ";

  cin>>firstnum >> secondNum;

  ptr1 = &num1; //assigning an address to pointer

  ptr2 = &num2;

  sum = *ptr1 + * ptr2; //values at address stored by pointer

  cout<<"\n Sum is: "<< sum;

  return  0;
```

7.1.3 Python

Python was specifically designed with readability and ease of use in mind. Python doesn't require the user to define a main function, and it doesn't even require the user to include packages for most commonly used statements such as a print statement. Due to these considerations, Python is a very readable and simple programming language.
Below is a statement to print the addition of two numbers:

```
# Store input numbers
num1 = input('Enter first number: ')
num2 = input('Enter second number: ')

# Add two numbers
sum = float(num1) + float(num2)

# Display the sum
print('The sum of {0} and {1} is {2}'.format(num1, num2, sum))
```

7.1.4 Critique

In simply comparing the code for addition in the three examples, it's easy to see that Python is the most readable code of the three and C++ is the least readable. C++ as a language is determined to ensure that nothing unnecessary is brought into memory, which translates to more code being necessary even for the most simple statements.

8. SUMMARY

From the culmination of all these comparisons it can be concluded that the major way each of these programming languages differ is in how much power they choose to provide their users. C++ prefers to offer their users the most power, including pointers and other simple powerful syntax despite security concerns. In providing their users more power C++ also proves to be the most efficient programming language, essentially offering shortcuts and asking the most of the programmer to ensure that efficiency is high. Java takes more power away from the user, limiting how they use pointers and doing implicit garbage collection to ensure the user can't make many of the mistakes which come with pointers, improving security. Java's JIT helps it maintain relative efficiency, but ultimately Java can't match C++'s compiled efficiency. Python is the least powerful of the three but the most readable and user friendly. Python takes most of the powerful methods included in the other programming languages away from the user, but ultimately protects the user from encountering anything confusing, which allows them to develop efficiently, though the actual execution of python code is the most inefficient.

9. REFERENCES

[1] Bruce J. Maclennan, Programming language comparisons, general knowledge, system and language characteristics. *Principles of Programming Languages, Design, Evaluation, and Implementation 3rd edition.* Oxford University Press. 1999.

[2] Lutz, Mark. *Programming python*. " O'Reilly Media, Inc.", 2001.

[3] "Msvcrt - Useful Routines from the MS VC++ Runtime." *Python 3.9.2 Documentation,* docs.python.org/3/library/msvcrt.html.

[4] "tty — Terminal control functions" Python 3.9.2 Documentation, docs.python.org/3/library/tty.html.

[5] Meyers, Scott. *Effective C++: 55 Specific Ways to Improve Your Programs and Designs. 3rd ed*., Addison-Wesley Professional, 2005.