CIS 343 - Structure of Programming Languages

Ira Woodring

Language Evaluation Criteria

(Follows the Sebesta Text Chapter 1)

Language Evaluation Criteria

We know now that different languages address different domains.

- Sometimes that may be enough for us to make a choice.
- For instance, we wouldn't want to use Java or Python to make an operating system, as they require an interpreter.
- C may be a poor fit for many simple desktop apps.

Sometimes though, languages may be more general purpose. So then, how can we compare them?

Language Evaluation Criteria

There are a lot of different criteria. The main ones we will discuss in this class are

- Readability
- Writability
- Reliability
- Expressivity
- Cost

Language Evaluation Criteria

We will mostly focus on the first three in this class

- **Readability**
- **Writability**
- **Reliability**
- Expressivity
- Cost

Language Evaluation Criteria - Readability

Refers to how easily the language can be read and understood.

- For instance, here is the source code for the original *Prince of Persia* game: https://raw.githubusercontent.com/jmechner/Prince-of-Persia-Apple-II/master/01 POP Source/Source/HIRES.S
- In case you're interested, here is the original game play on the Apple II: https://www.youtube.com/watch?v=T5-06QnCHKY

Language Evaluation Criteria - Readability

This Assembly Language code was very low-level.

- It lacks concepts like for-loops and other higher-level constructs.
- Because of these deficiencies it is much more cumbersome to express ideas concisely.

However, having higher-level constructs alone does not necessarily make a language more readable. Consider the following code:

https://www.ioccc.org/2020/kurdyukov1/prog.c

Language Evaluation Criteria - Readability

This is perfectly valid C code.

- It includes higher-level concepts than Assembly Language typically provides.
- Still, anything but readable.

So then, what makes code written in one programming language more readable than another?

Language Evaluation Criteria - Readability

Simplicity.

- Having a small number of basic constructs
- Low feature multiplicity
- Careful use of operator overloading

Language Evaluation Criteria - Readability

Having a small number of basic constructs

- How many keywords and operators does the language have?
- The larger the number the less readable.
    - Consider COBOL:

  https://www.ibm.com/support/knowledgecenter/en/SSZJPZ_8.7.0/com.ibm.swg.im.iis.ds.mfjob.dev.doc/topics/r_dmnjbref_COBOL_Reserved_Words.html

Language Evaluation Criteria - Readability

Compare to Python:

https://www.programiz.com/python-programming/keywords-identifier

Language Evaluation Criteria - Readability

Feature multiplicity

How many different ways can you accomplish the same operation? For instance, in many languages you may do any of the following to accomplish the same task:

```
count = count + 1
count += 1
count++
++count
```

The more ways to accomplish a task, the lower the readability.

Language Evaluation Criteria - Readability

Operator Overloading

How many meanings do the operators have?

Consider the C programming language. The * operator can have many meanings.

- It can be used for
  - multiplication
  - to declare a pointer
  - to dereference a pointer
  - to help signify a comment

For instance:

Language Evaluation Criteria - Readability

```c
#include <stdio.h>
#include <stdlib.h>

/* Let's create some awesome code! */

int main(void){
        int * a = malloc(10 * sizeof(int));
        *(a + 0) = 42;
        *(a + 1) = *a + *a;
        int b = *a + 1;
        printf("%i\n\n", b);

}
```

What does this even output???

Language Evaluation Criteria - Readability

For readability, we want our language to be **orthogonal**.

- relatively small set of primitive constructs
- can be combined in a relatively small number of ways to write code
- and that every possible combination of primitives is legal and meaningful.

(Notice use of the word "relatively").

Language Evaluation Criteria - Readability

For instance, in the Java programming language it is perfectly legal to return any data type from a function; in other languages though (such as C), attempting to return an array data type from a function is invalid code.

Eg:

Java

```java
public Car[] getCars(){
  Car[] cars = this.my_cars;
  return cars;                // Valid!
}
```

C

```c
int[] getValues(){
  int values[10];
  for(int i=0; i<10; i++)}
    values[i] = i;
  }
  return values;             // Not valid!
}
```

Language Evaluation Criteria - Readability

What that means is that in this instance Java is more orthogonal.

- Java and C both have a small number of constructs (such as the idea of arrays)
- C has more limits to how they can be used
- Java has no such restrictions.
- Java is therefore more readable; in C we have to use pointers to return an array, but we also use pointers to return pointers from functions.
- **Understanding the programmer's original intent becomes harder.**

Language Evaluation Criteria - Readability

But, with everything there is a catch.

- Too much orthogonality can be bad too
- For instance, are conditionals allowed on the left side of an assignment statement?

```
5 * a = b           // May be valid (in some languages), but what is the intent?
```

Language Evaluation Criteria - Readability

Are the constructs adequate to describe what needs to be described?

- In Python we have a boolean type, but C does not.
- In C we must use an integer type to simulate a boolean:

```
goAhead = true
```

```
int goAhead = 1;
```

(Yes, I know there are boolean libraries for C. They still don't provide full boolean functionality though. Try using them in an `if` statement.)

Language Evaluation Criteria - Readability

Are there easy to use facilities for denoting code groups?

Many languages use braces and semi-colons

```
for(int i=0; i<10; i++){ ... }
```

Some languages use tabs or spaces

```
for i in range(0,10):
    ....
```

Others use matching words

```
if [ $a $eq 1 ]
then
  ...
fi
```

If a language lacks such facilities, it will be much harder to read.

Language Evaluation Criteria - Readability

Can special words be used as names of variables (hopefully not!)? Consider how confusing this might be:

```
int while = 10;

while( while < 100 ){
  ...
}
```

Language Evaluation Criteria - Readability

Form and (fairly) obvious meaning.

- Meaning should be apparent from the syntax, without the need for context.
- Consider this C code:

```c
static double pi = 3.14159;           // outside of a function definition;
                                      // means only available to code in
                                      // this file.

int main(int argc, char** argv){
  static int my_val = 1;              // inside a function; static here means
                                      // this is a compile time variable
}
```

Writability

Measures how easily the language can be used to create programs.

- The same issues that affect Readability also affect Writability.
- But, there are a few more!

Writability - Simplicity and Orthogonality

The larger and more complex the language, the harder it is to use it to solve problems.

- Consider how hard it is to debug a language in which every combination of primitives is legal.
- For instance, in C it can be very confusing to discover pointer errors.

```c
#include <stdio.h>

int main(int argc, char** argv){
        int a = 42;
        int b = 2;
        int* x = &a;
        int* y = &b;
        int z = *x++*++*y;
        printf("%d, %d, %d\n", a, b, z);
}
```

Writability - Support for Abstraction

Abstraction refers to the ability to define and use complicated structures without worrying about the underlying details.

- In our Ruby matrix example earlier, we didn't need to know or care to know how the matrix was stored behind the scenes.
- All we cared about was that we were given a data structure that could efficiently store and operate on a matrix for us.

Writability - Support for Abstraction

- There are two types of abstraction in computing, process abstraction and data abstraction.
    - **Process Abstraction** is the use of subprograms (a function or module) to repeatedly solve a problem. For instance, if we can pass a custom comparison operator to a sorting function then we can use the same code to sort our data regardless of the data type.
    - **Data Abstraction** allows us to define a data type and operations that work on that data type (such as a class in Java) that we can then reuse.

Writability - Expressivity

Are there convenient ways to accomplish computations, or do we need to rely on cumbersome methods?

- Python, for instance, doesn't allow us to use the (nearly ubiquitous) prefix and postfix operators for variables.

```
count = 0
count++                    // NOT VALID!

count = count + 1          // Valid.
```

Reliability

Does a language always perform to its specifications, under all conditions?

- Don't think reliability as in the mechanical sense.
- It isn't as if the language isn't going to "start-up" one day.
- More, "How well can the language protect us from the unexpected?"

Reliability - Type Checking

Does the language test for type errors either during compilation or execution?

- For instance, are these errors?

```
int pi = 3.14159
bool ok = -16384;
```

- Some languages will see this as a type error. Others may not.

Reliability - Exception Handling

If the language provides facilities to catch run-time errors, correct an issue and continue running, then it has exception handling.

```
try {
  ... Open a file on a disk for instance ...
}
catch (FileNotFoundException fnfex){
  ... recover; don't let the program crash ...
}
```

Reliability - Aliasing

Does the language allow for two or more names for the same memory location?

- Many languages do, but it can be dangerous and harder to program with!
- This makes these languages less reliable.

```
int value = 42;
int & alias = value;     // alias now refers to the same location
                         // For all intents and purposes they are
                         // the same variable.
```

Writability - Cost

We won't be discussing this factor much in this class (it is more a Software Engineering topic), but Cost is a factor in language comparison.

- Training programmers to use the language. Often more powerful languages are more complicated to teach.
- How well the language fits the domain of the application can affect cost. If the domain is not well supported by the language it may take much longer to program the application.
- Compilation costs such as the price of the compiler (proprietary compilers can be expensive!), downtime waiting for compilation to complete, etc. affect cost.
- Cost can also be influenced by runtimes. Consider for instance renting time on a supercomputer. This is VERY expensive. A less efficient language may require longer run times to complete similar jobs.

Writability - Cost

- Reliability can affect cost as well.
- If the language doesn't have built-in exception handling, you will need to write code to simulate it.
- Products that fail can cost your company a lot!