CMSC 124

Design and Implementation of Programming Languages

Kristine Bernadette Pelaez
Institute of Computer Science
University of the Philippines Los Baños

Language Evaluation Criteria

1. Readability
2. Writability
3. Reliability
4. Cost
5. *Portability*
6. *Generality*
7. *Well-definedness*

1. Readability

1. Readability

the *ease* with which a program can *be read and understood*

1. Readability

makes *maintenance easier*

Factors affecting Readability

1. Overall Simplicity
2. Orthogonality
3. Data Types
4. Syntax Design

1.1 Overall Simplicity

1.1 Overall Simplicity

are there *a lot of constructs* that I must know?

1.1 Overall Simplicity

are there *a lot of ways*
to do the same thing?

1.1 Overall Simplicity

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

1.1 Overall Simplicity

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

Feature Multiplicity
having more than one way to do a certain operation

1.1 Overall Simplicity

do the operators
have multiple uses?

1.1 Overall Simplicity

```
answer = 5+4*3
fruit = "ba"+("na"*2)
```

1.1 Overall Simplicity

```
answer = 5+4*3
fruit = "ba"+("na"*2)
```

Operator Overloading
operators have multiple meanings

1.1 Overall Simplicity

However,
too much simplicity
makes a language *less readable*.

1.1 Overall Simplicity

```
pikachu pika pikachu pika pika pi pi pika pikachu pika pikachu pi pikachu pi pikachu pi pika pi pikachu pikachu pi pi pika pika pikachu pika pikachu pikachu pi pika pi pika pika pi
pikachu pikachu pi pikachu pi pika pikachu pi pikachu pika pikachu pi pikachu pikachu pi pikachu pika pika pikachu pi pikachu pi pi pikachu pikachu pika pikachu pi pika pi pi pika
pika pikachu pikachu pi pi pikachu pi pikachu
pikachu pikachu pi pikachu
pikachu pika pika pikachu pika pikachu pikachu pika pika pikachu pikachu pi pi pikachu pika pikachu pika pika pi pika pikachu pikachu pi pika pika pikachu pi pika pi pika pi pikachu
pi pikachu pika pika pi pi pika pi pika pika pikachu pikachu pika pikachu pikachu pika pi pikachu pika pi pikachu pi pika pika pi pikachu pika pi pika pikachu pi pi pikachu pika pika
pi pika pi pikachu
pikachu pikachu pi pikachu
pikachu pika pi pika pika pikachu pika pikachu pi pikachu pi pi pika pi pikachu pika pi pi pika pikachu pi pikachu pi pi pikachu pikachu pika pikachu pikachu pika pi pikachu pi pika
pikachu pi pikachu pika pika pikachu pika pi pi pikachu pikachu pika pika pikachu pi pika pikachu pikachu pi pika pikachu pikachu pika pi pi pikachu pikachu pi pikachu pi pikachu pi
pikachu pi pika pikachu pi pikachu pika pikachu pi pika pi pikachu
pi pika
pikachu pikachu pi pikachu
pika pi
pikachu pikachu pi pikachu
pikachu pi pikachu pi pi pikachu pi pikachu pika pikachu pikachu pi pikachu pikachu pika pi pi pika pikachu pika pikachu pi pi pikachu pika pi pi pikachu pika pika pi pika pika
pikachu pika pikachu pi pi pika pikachu pika pi pikachu pikachu pi pikachu pika pikachu pikachu pika pi pi pikachu pikachu pi pika pikachu pi pikachu pika pikachu pikachu pika pi
pikachu pikachu pika pikachu pi pikachu pika pika pi pikachu pi pika pi pikachu pikachu pi pikachu
pi pika
pikachu pikachu pi pikachu
pikachu pikachu pi pika pikachu pi pika pika pi pi pika pi pikachu pi pika pi pika pi pika pikachu pika pi pi pikachu pi pikachu pi pika pi pika pika pikachu pi pikachu
pikachu pikachu pi pikachu
pikachu pi pikachu pika pikachu pi pika pi pikachu pikachu pika pika pi pi pikachu pi pika pi pikachu pi pika pikachu pi pika pi pi pikachu pikachu pika pika pikachu pikachu pi pi
pikachu pi pikachu pi pikachu pi pi pikachu pikachu pi pikachu pi pikachu pi pika pika pikachu pikachu pika pi pika pikachu pi pikachu pi pi pika pikachu pika pi pikachu pi pika pi
pi pikachu pikachu pika pika pikachu pika pika pikachu pi pika pi pika pikachu pi pika pikachu pika pi pika pikachu
pikachu pikachu pika pikachu
pikachu pikachu pika pikachu
pi pi pikachu pi pikachu pika pika pi pikachu pika pika pi pi pika pika pikachu pi pi pikachu pi pika pi pika pikachu pi pikachu pi pikachu pikachu pi pi pika pika pi pika pika pi
pika pikachu pikachu pi pikachu pika pi pi pika pi pi pikachu pikachu pika pi pi pika pika pi pika pikachu pi pikachu pi pi pika pi pika pika pikachu pika pi pika pikachu pi pikachu
pikachu pi pi pika pi pika pika pikachu pikachu pi pikachu
pikachu pikachu pi pikachu
pikachu pi pikachu pikachu pika pikachu pikachu pika pika pikachu pikachu pika pikachu pi pika pikachu pika pika pi pikachu pi pi pika pi pi pikachu pika pika pikachu pikachu pika
pikachu pikachu pi pika pi pi pikachu pikachu pika pi pi pikachu pikachu pika pikachu pika pi pikachu pi pika pi pika pikachu pika pi pikachu pi pikachu pikachu pi pika pikachu pi
pikachu pikachu pi pika pi pikachu pikachu pi pikachu pika pika pi pi pikachu
pikachu pi pi pika pi pi pikachu pika pikachu pikachu pika pika pi pi pika pikachu pi pikachu pi pi pika pi pika pi pi pika pikachu pi pika pi pikachu pika pikachu pika pi pi pika pi
pi pikachu pi pikachu pikachu pika pi pikachu pi pi pika pi pikachu pi pi pika pi pi pikachu pika pikachu pika pikachu pika pi pikachu pikachu pi pi pika pika pikachu
pikachu pikachu pi pikachu
pikachu pikachu pika pikachu
```

1.1 Overall Simplicity

Since there is a lack of complex structures, *more statements will be needed to execute an operation*.

1.2 Orthogonality

1.2 Orthogonality

validity of the combination of primitive constructs to build the control and data structures

1.2 Orthogonality

```
float temperature;

char letter;

unsigned int age;
```

In some languages, the *data type* construct and *identifier* construct is used to create *variables*.

1.2 Orthogonality

lack of orthogonality leads to *exceptions to the rules*

1.2 Orthogonality

✓ `float temperature;`

✓ `char letter;`

✓ `unsigned int age;`

✗ `void variable;`

1.2 Orthogonality

✓ `float temperature;`

✓ `char letter;`

✓ `unsigned int age;`

✗ `void variable;`

✓ `void *variable;`

1.2 Orthogonality

```
struct human{
   char dna[17];
   int age;
   struct human child;
};
```

```
struct human{
   char dna[17];
   int age;
   struct human *child;
};
```

✓

1.2 Orthogonality

less orthogonal: more exceptions

more orthogonal: less exceptions

1.2 Orthogonality

too little orthogonality will result to a *lot of exceptions* the programmer must remember.

1.2 Orthogonality

too much orthogonality will result to a large set of valid control and data structures which *makes looking for errors harder*.

1.3 Data Types

1.3 Data Types

being able to define
data types will make
the *meaning of statements clearer*

1.3 Data Types

```
while(-124){
   printf(“Hello”);
}
```

```
while(true){
   printf(“Hello”);
}
```

1.4 Syntax Design

1.4 Syntax Design

special words or symbols
help in program readability

1.4 Syntax Design

COBOL:

```
IF COND THEN
  IF COND THEN
    <statements>
  END-IF
ELSE
  <statements>
END-IF.
```

C:

```
if(cond)
   if(cond)
      <statements>
else
   <statements>
```

1.4 Syntax Design

the *form of the elements*
of the language may *change the meaning or usage of some constructs*

1.4 Syntax Design

```
answer = 5+4*3

fruit = “ba”+(“na”*2)
```

2. Writability

2. Writability

the *ease* with which a language can be used to *write a program*

Factors affecting Writability

1. Overall Simplicity
2. Orthogonality
3. Abstraction
4. Expressiveness

2.1 Overall Simplicity

2.1 Overall Simplicity

having a lot of constructs may lead to the *misuse/disuse of features*

2.2 Orthogonality

2.2 Orthogonality

too much orthogonality *makes errors in the program harder to detect*

2.3 Abstraction

2.3 Abstraction

ability to *design* structures or operations *with many details ignored*

2.3 Abstraction

process abstraction: uses functions

data abstraction: uses classes

2.4 Expressiveness

2.4 Expressiveness

convenience in specifying commands and statements

2.4 Expressiveness

COBOL:

```
MOVE y TO x.
ADD y TO x GIVING z.
```

C:

```
x = y;
z = x + y;
```

2.4 Expressiveness

a *more expressive* language
is *more writable*
but is *less readable*

3. Reliability

3. Reliability

ability of a language
to *perform to its specifications under all conditions*

Factors affecting Reliability

1. Type Checking
2. Exception Handling
3. Aliasing
4. Readability
5. Writability

3.1 Type Checking

3.1 Type Checking

testing for type errors
during compile-time or run-time

3.1 Type Checking

Python:

```
def sum(x, y):
   return x+y
```

C:

```
int sum(int x, int y){
   return x+y;
}
```

3.2 Exception Handling

3.2 Exception Handling

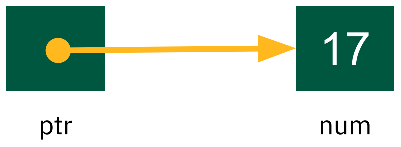
ability of a program
to *intercept and correct run-time errors,*
and continue running

3.3 Aliasing

3.3 Aliasing

allows *two or more distinct names*
to access *the same memory cell*

3.3 Aliasing



3.4 Readability

3.4 Readability

the *easier it is to read* a program,
the *easier it is to maintain/update*.

3.5 Writability

3.5 Writability

the *easier it is to write* a program,
the *more likely it is to be correct*.

4. Cost

4. Cost

how much resource
was used to create a program
in the given language?

Factors affecting Cost

1. Cost of training
2. Cost of writing
3. Cost of compiling
4. Cost of executing
5. Cost of language implementation system
6. Cost of poor reliability
7. Cost of maintenance

4.1 Cost of training

4.1 Cost of training

amount of resource used to learn
the programming language?

4.2 Cost of writing

4.2 Cost of writing

amount of resource used to write
using the programming language?

4.2 Cost of writing

can be reduced by using *IDEs* (integrated development environment) or *autocomplete tools*.

4.3 Cost of compiling

4.3 Cost of compiling

amount of resource used to compile a program in the given language

4.4 Cost of executing

4.4 Cost of executing

amount of resource used to run a program in the given language

4.5 Cost of implementation system

4.5 Cost of implementation system

is the compiler/system/hardware that the PL requires *expensive*?

4.6 Cost of poor reliability

4.6 Cost of poor reliability

if the *program fails*, will the resulting incident be *expensive*?

4.7 Cost of maintenance

4.7 Cost of maintenance

is it *easy to maintain/update* the program?

5. Portability

5. Portability

the ease with which *a program can be moved from one implementation to another*

5. Portability

standardized languages are more portable.

6. Generality

6. Generality

can it be *used in a wide range of applications*?

7. Well-definedness

7. Well-definedness

the language's *official defining document is complete and precise*

CMSC 124

Design and Implementation of Programming Languages

Kristine Bernadette Pelaez
Institute of Computer Science
University of the Philippines Los Baños