Notes Feb 4, 2019

1. Programming Languages Third Edition

* Language Design Criteria

1. Objectives

* Describe the history of programming language design criteria
* Understand efficiency in programming languages
* Understand regularity in programming languages
* Understand security in programming languages
* Understand extensibility in programming languages
* Understand the design goals of C++
* Understand the design goals of Python

1. Background

* What is a good programming language design?
* What criteria should be used to judge a language?
* How should success or failure of a language be defined?
* We will define a language as successful if it satisfies any or all criteria:
  + It achieves the goals of its designers
  + It attains widespread use in an application area
  + It serves as a model for other languages that are successful

1. Background (cont’d.)

* When creating a new language, decide on an overall goal and keep it in mind throughout the design process
* This is especially important for special purpose languages
  + The abstractions for the target application area must be built into the language design
* This chapter introduces some general design criteria and presents a set of detailed principles as potential aids to the designer.

1. Historical Overview

* In the early days, machines were extremely slow, and memory was scarce
  + Program speed and memory usage were prime concerns.
* Efficiency of execution: primary design criterion
  + Early FORTRAN code directly mapped to machine code, minimizing the amount of translation required by the compiler
* Writability: the quality of a language that enables a programmer to use it to express computation clearly, correctly, concisely, and quickly.

1. Historical Overview(cont’d.)

* In the early days, writability was less important than efficiency.
* Algol60 was designed for expressing algorithms in a logically clear and concise way
  + Incorporated block structure, structured control statements, a more structured array type, and recursion
* COBOL attempted to improve.

1. Historical Overview (cont’d.)

* In the 1970s and early 1980s, the emphasis was on simplicity and abstraction, along with readability.
  + Mathematical definitions for language constructs were introduced, along with mechanisms to allow a translator to partially prove the correctness of a program before translation
  + This led to strong data typing
* In the 1980s and 1990s, the emphasis was on logical or mathematical precision
  + This led to a renewed interest in functional languages.

1. Historical Overview (cont’d.)

* The most influential design criteria of the last 25 years is the object-oriented approach to abstraction
  + Led to the use of libraries and other object-oriented techniques to increase reusability of existing code
* In addition to the early goals of efficiency, nearly every design decision still considers readability, abstraction, and complexity control.

1. Efficiency

* Efficiency: usually thought of as efficiency

1. Efficiency (cont’d.)

* Programmer efficiency: how quickly and easily can a person read and write in the programming language?
* Expressiveness: how easy is it to express complex processes and structures?
* Conciseness of the syntax also contributes to programmer efficiency
  + Example: Python does not require braces or semi-colons, only indentation and the colon (:)

1. Efficiency (cont’d.)

* Reliability of a program can be viewed as an efficiency issue
  + Unreliable programs require programmer time to diagnose and correct.
* Programmer efficiency is also impacted by the ease with which errors can be found and corrected
* Since roughly 90% of time is spent on debugging and maintaining programs, maintainability may be the most important index

1. Regularity

* Regularity: refers to how well the features of a language are integrated
* Greater regularity implies:
  + Fewer restrictions on the use of particular constructs
  + Fewer strange interactions between constructs
  + Fewer surprises in general in the way the language features behave
* Languages that satisfy the criterion of regularity are said to adhere to the principle of least astonishment

1. Regularity (cont’d.)

* Regularity can be subdivided into three concepts:
  + Generality
  + Orthogonal design
  + Uniformity
* Generality: achieved by avoiding special cases in the availability or use of constructs and by combining closely related constructs into a single more general one
* Orthogonal design: constructs can be combined in any meaningful way, with no unexpected restrictions or behaviors.

1. Regularity (cont’d.)

* Uniformity: a design in which similar things look similar and have similar meanings

1. Generality

* Generality: A language with this property avoids special cases wherever possible
* Example: procedures and functions
  + Pascal allows nesting of functions and procedures and passing of functions and procedures as parameters to other functions and procedures but does not allow them to be assigned to variables or stored in data structures.
* Example: operators
  + In C, cannot directly compare two structures with ==; thus, this operator lacks generality.

2. Orthogonality

* In a language that is truly orthogonal, constructs do not behave differently in different contexts
  + Restrictions that are context dependent are nonorthogonal, while restrictions that apply regardless of context exhibit a lack of generality
* Example: function return types
  + Pascal allows only scalar or pointer types as return values
  + C and C++ allow values of all data types except array types
  + Ada and Python allow all data types

1. Orthogonality (cont’d.)

* Example: placement of variable declarations
  + C requires that local variables be defined only at the beginning of a block
  + C++ allows variable definitions at any point inside a block prior to use

1. Orthogonality (cont’d.)

* Orthogonality was a major design goal of Algol68
  + It is still the best example of a language in which constructs can be combined in all meaningful ways.

1. Uniformity

* Uniformity: refers to the consistency of appearance and behavior of language constructs
* Example: extra semicolon
  + C++ requires a semicolon after a class definition but forbids its use after a function definition
* Example: using assignment to return a value
  + Pascals

1. Causes of Irregularities

* Many irregularities are case studies in the difficulties of language design
* Example: extra semicolon problem in C++ was a byproduct of the need to be compatible with C
* Example: irregularity of primitive types and reference types in Java is the result of the designer’s concern with efficiency
* It is possible to focus too much on a goal
* Example: Algol68 met its goals of generality and orthogonality, but this

1. Security

* Reliability can be affected if restrictions are not imposed on certain features
  + Pascal: pointers are restricted to reduce security problems
  + C: pointers are much less restricted and thus more prone to misuse and error
  + Java: pointers

2. Security (cont’d.)

* ML and Haskell are functional languages that attempt to be secure yet allow for maximum expressiveness and generality
  + They allow multitype objects, do not require declarations, and yet perform static type-checking
* Semantically safe: languages that prevent a programmer from compiling or executing any statements or expressions that violate the language definition
  + Examples: Python, Lisp, Java

1. Extensibility

* Extensible Language: a language that allows the user to add features to it
* Example:

1. C++: An Object-Oriented Extension of C
2. C++: An Object Oriented Extension of C (cont’d)
3. C++: First Implementations (cont’d.)

(32) Python: A General-Purpose Scripting Language

- Guido van Rossum developed a translator and virtual machine for a scripting language called Python in 1986

- One of his goals was to allow Python to act as a bridge between system languages such as C

(34) Python: Simplicity

(36) Python: Dynamic Typing

(38) Python: Retrospective