Fundamental Concepts of Programming Languages

PL quality factors
Lecture 01

conf. dr. ing. Ciprian-Bogdan Chirila

University Politehnica Timisoara
Department of Computing and Information Technology

October 4, 2022

1 Overview

2 The place of the PL in the software development process

3 Criteria for evaluating a PL

Lecture and lab

- Ciprian-Bogdan Chirila PhD
  - Associate professor
  - PhD UPT + Univ. Nice Sophia Antipolis, Univ. Jyvaskyla, Finland + Univ. Bonn, Germany
  - Reverse inheritance in Eiffel
  - chirila@cs.upt.ro
  - http://www.cs.upt.ro/∼chirila
- Raul Brumar and Alexandru Grosu
  - IT&C engineers
  - programming, game production, graphics
  - internships in Oulu, Finland; Klagenfurt, Austria
  - raulbrumar@yahoo.de; alex.gm98@yahoo.com
  - C# programming concepts

Lecture outline (1)

- Programming languages
- Definition and implementation of programming languages
- Program entity attributes
- Parameter transmission
- Generic subprograms

Lecture outline (2)

- Data types
- Abstract data types
- Object-oriented programming languages
- Control structures
- Functional programming

Programming Language

- Programming language (PL)
  - formal notation specifying several operations to be executed (by a computer)
- Many programming languages exist today
- Few are used on a large scale in writing nowadays programs

1 Overview

2 The place of the PL in the software development process

3 Criteria for evaluating a PL

The place of the PL in the software development process

- A complex software product is developed usually in 5 steps or phases:
    - Requirements analysis and specification
    - Software design and specifications
    - Implementation
    - Validation
    - Maintenance

Phase 1: Requirements analysis and specification

- During the analysis the user needs are concentrated in a series of requests
- The result of this phase is a document describing WHAT the system must do
- There nothing said about HOW it will be done
- The final evaluation of the software product will refer the requirements set in this phase

Phase 2: Software design and specifications

- After reading the requests the software system will be designed accordingly
- In this phase we do
    - The project specification
    - Module definitions
    - Interface definitions

Phase 3: Implementation

- Is done according to the specification
- The PL is chosen to be the most suitable for the system context
- Several criteria are taken into account
    - How much the programmer knows the PL
    - How much the PL features are suitable to the requirements
    - What features offer the IDE (Integrated Development Environment) for coding and testing
    - What execution speed performances are reached by the compiled system in the selected PL

Phase 4: Validation

- Is done in each phase of the development process
- It means checking whether the system respects the requirements
- Intense testing process
  - Using multiple data sets
  - Reaching all program branches
  - Creating extreme conditions

Phase 5: Maintenance

- After deployment errors may occur
    - Fixing is needed
- Possible causes
    - Undiscovered errors in the validation phase
    - Extending the program with new features
    - Optimizing parts of the programs leading to a better performance
    - Hardware or software platform changes

The place of the PL in the software development process

- Where is its impact?
- Directly in phase 3 in the implementation phase
- Interacts with all other development tools
- Is involved in all the other phases

The place of the PL

- Some PL properties may affect
  - validation
  - maintenance
  - design
- e.g.
  - Information hiding as design method and language facility in describing abstract data
  - Information hiding involves:
    - Segregation of the design decisions that are the most likely to change
    - Decomposing the system in modules
    - Modules must have interfaces (sets of functions)
    - The access to the modules is made only through the interfaces
    - Modules internal structures is not visible from the outside
  - Programming languages supporting these facilities are object oriented-programming languages (OOPLs)

1 Overview

2 The place of the PL in the software development process

3 Criteria for evaluating a PL

Criteria for evaluating a PL

- the PL is not an end in itself
- the PL must allow creating in an efficient way quality software
- In order to define a good PL we must define a good software system
- The three basic quality factors we consider are:
  - reliability
  - maintainability
  - efficiency

The three quality factors

- Reliability
  - Correct functioning of the system even in the presence of software and hardware incidents
- Maintainability
  - The capability of including new features or upgrading the existing ones
- Efficiency
  - It means offering optimal services in the context of existing resources

Other factors

- Design methods
- IDE (Integrated Development Environment) tools
- Algorithms
- Human factors
- and last but not least ... the PL!!!

PL qualities

- Consistency with the usual notation
- Readability
- Exception handling
- Error detection
- Automatic formal checking
- Orthogonality
- Uniformity
- Scalability
- Portability
- Efficiency

Consistency with the usual notation

- The notation used in programming must be close to the usual notation
    - Scientific
    - Technical
    - Economical
    - etc.
- The programmer can focus on program semantics for solving the problem and not on notation issues
- Less errors
- Greater productivity

Readability

- The program must be easily read
- Its logic must be deducible from the context
- Is important when programmers modify the code of other programmers
- For increased readability the PL must have
  - Identifiers
  - Expressive keywords
  - Software decomposition facilities

Exception handling

- Important for creating reliable programs
- Program sequences can be specified which will be activated when exceptional phenomena occur
  - arithmetic overflow, underflow
  - external events
  - etc.
- Thus, the program behavior becomes predictable

Error detection

- PL definition must allow detecting errors at compile time as much as possible
- Useful redundancy - imposed by the majority of modern PLs
- The same information (implicit or explicit)
    - is specified in multiple places of the program
    - is verified at compile time

Compile time checking

- An entity must be first declared and then referred or used
- Type correspondences between
  - Operands
  - Operands and operators
  - Left hand side and right hand side of an assignment, etc.
- Type correspondence between actual and formal parameters
- Respecting visibility rules
  - Domain rules
  - Import and export rules of entities between modules
  - Abstract types
  - Objects

Compile type checking

- can not detect program logic or semantic mistakes
- can not guarantee that a fully compiled program function according to imposed specifications

Formal verification

- Is the act of proving or disproving the correctness of algorithms with respect to a formal specification using formal methods of mathematics
- Involves the formal description of specifications
- PL semantic definition according to a formalism compatible with the formal checking method
- Building the semantic of the checked program based on the PL semantic
- Tools implementation for checking the matching between the specification and the semantics of the program

Formal verification

- Useful in:
  - cryptographic protocols
  - combinational cicuits
  - digital circuits having internal memory
  - software expressed as source code
- Used mathematical objects:
  - finite-state machines, labelled transition systems, Petri nets
  - formal PL semnatics like operational semantics, denotational semantics, axiomatic semantics

Orthogonality

- The language must be defined on basic facilities
- Facilities must be able to be freely combined
- With predictable effects
- With no restrictions
- e.g. lack of orthogonality in Pascal
    - functions can not be members in a structured type

Uniformity

- Similar constructions have similar semantics
- e.g. lack of uniformity in C for the static keyword
  - Used in a function static refers to memory allocation (opposed to automatic)
  - Used outside a function influences visibility

Uniformity

```
#include<stdio.h>
int fun()
{
  static int count = 0;
  count++;
  return count;
}
int main()
{
  printf("\%d ", fun());
  printf("\%d ", fun());
  return 0;
/* outputs 1 2 */
}
```

```
#include<stdio.h>
int fun()
{
  int count = 0;
  count++;
  return count;
}
int main()
{
  printf("\%d ", fun());
  printf("\%d ", fun());
  return 0;
/* outputs 1 1 */
}
```

Uniformity

Parameter	Internal Static Variables	External Static Variables
Keyword	"static"	"static"
Linkage	Internal static variable has no linkage.	External static variables has internal linkage.
Declaration	Internal static variables are declared within the main function.	External static variables are declared above the main function.
Comparison	Internal static variables are similar to auto(local) variables.	External static variables are similar to global(external) variables.
Visibility	Internal static variables are active(visibility) in the particular function.	External Static variables are active(visibility)throughout the entire program.
Lifetime	Internal static variables are alive(lifetime) until the end of the function.	External static variables are alive(lifetime) in the entire program.
Scope	Internal static variables has persistent storage with block scope (works only within a particular block or function).	External static variables has permanent storage with file scope (works throughout the program).

Scalability

- Program modularization
- Component hierarchy
- Main facilities
  - abstract types
  - modules
  - separate compiling
  - object files *.obj, *.o

Portability

- Moving a program from a computer to another
  - without modifications
  - with small modifications
- The goal of "machine independence" is impossible to achieve
- Some PLs allow a close approach
  - Java runs on JVM running on Windows, Linux, MacOS
  - C# runs on .NET Framework running on Windows, Linux, MacOS
- Problems
  - Different lengths for the computer word
  - Different floating point representation conventions
  - Different input-output operations

Efficiency

- From the point of view of
    - compilation
        - The PL must be defined as such in order to facilitate the creation of fast compilers
    - object program
        - Declaring variables and their types
        - Expression type inference at compile time
        - Strong typing like in Pascal

Bibliography

1. Brian Kernighan, Dennis Ritchie, C Programming Language, second edition, Prentice Hall, 1978.
2. Carlo Ghezzi, Mehdi Jarayeri – Programming Languages, John Wiley, 1987.
3. Horia Ciocarlie – Universul limbajelor de programare, editia 2-a, editura Orizonturi Universitare, Timisoara, 2013.