Lecture 2
The Expected Characteristics of P.L.

prof. Marco Bellia, Dip. Informatica, Università di Pisa

February 18, 2013

The Expected Characteristics of P.L.

- The Language Semplice: Syntactic Domains (*for starting*)
- Languages: Expected Characteristics
- The Table of Characteristics of Semplice

The Language Semplice

Table 1			
Syntactic Domains			
T	::=	<u>Int</u> | <u>Bool</u> | ...	(*Types*)
D	::=	<u>Var</u> I T | D D | ...	(*Dichiarazioni*)
C	::=	I := E | <u>While</u> E <u>do</u> C} | C C | ...	(*Comandi*)
E	::=	I | VL | ...	(*Espressioni*)
I	::=	...	(*Literals*)
VL	::=	...	(*Literals*)

- Domain = Set of the Legal Terms (+ Relations on them, e.g. ordering)
- Syntactic Domain = Legal Syntactic terms (+ Flatten ordering)
- Abstract Syntax = Syntactic Domain speaks about the Abstract Syntax of terms

Example

Hence no matter in the name of constructors, e.g.:
`While _ _` instead of `While _ do_`

Expected Characteristics

- 1. Clarity, Simplicity, Unity
- 2. Orthogonality
- 3. Application Development
- 4. Support for Abstractions
- 5. Modification and Reuse
- 6. Properties Verification
- 7. Programming Environment
- 8. Portability
- 9. Efficiency
- 10. Technology Supported

Wasserman ADA: Requirements for High Order Programming Languages

1. Clarity, Simplicity, Unity

- The Language must provide user with a:

  clear to understand [hence unambiguous, clear in the expressed meaning, behavior)]

  simple to use [hence with simple composition rules or program construction laws: Syntactic as well as Semantics]

  unified in the involved conceptual framework [Declarations for entity introductions, Commands for state transformations, Expression for value computations, Types for structure categorizations, etc.]

  set of concepts that can be used as primitives in developing the algorithms;

Example

Semplice has Clarity, Simplicity and Unity.

2. Orthogonality

- Ability to combine all together, and in all ways, the language features, having each combination:
  syntactically legal, and
  semantically meaningful
- An excess in orthogonality may lead to dangerous situations

Example

In C: "while E do ..." can be combined with any expression E, always obtaining a legal program that does not always compute what for which it was written.

- Good compromise

Example

Semplice meets the requirement in an acceptable way: Construct while, for instance, can be combined with any expression of type Bool.

3. Application Development

- **Suitable for Development of Applications.** The language structures must cope with those that are needed to the algorithms that are used to deal with the applications to be developed.
  - Data Structures + Operations
  - Control Structures
- Semplice meets only for *Application on Integers*

Example

Write the following two programs:
(P1) A program for ordering two given integers.
(P2) A program for ordering k integer.
Noting that, program (P2) requires $o(K)^2$ statements of Semplice.

3. Application Development/ 2

- Fortran/Algol,/Pascal/C/...

  Algorithms on Collections of data (atomic or scalar, including, integers, String, and non-atomic or structured data) with operations for general purpose applications.

- Cobol

  Algorithms on Collection of non-homogeneous data with operation for business-commercial applications

- ML/Miranda/Ocaml/Haskell/...

  Algorithms on Collections of possibly structured, non-mutable values, and abstract values. These algorithms can use Functions-as-Value, Higher-Order applications, Combinators.

- Prolog/DataLog/...

  Deductive Algorithms.

- Java/Scala/C#/...
  Algorithms on data with with environment, inheritance, sub-typing. These algorithms can be applied to code reuse.

4. Abstraction Support

- **Data and Control Abstraction.** The best, prominent way to do language extension in:
  - data representations and in
  - sequence control transfers.

Example

The scheduling of the lessons of students in the rooms requires data for representing *students*, in addition to operations that are needed for describing the *behaviors* that the algorithm computes for the lesson participation of the students of the different classes

- Semplice has not abstraction mechanisms, exception for *naming*

5. Modification and Reuse

- **Modification.** Easy modification of the software to the changes in the context of use of the program (including, changes in I/O interactions, functionalities, performance requirements)
- **Reuse.** Ability to use software
    - in an integral way (i.e. computing all its original functionalities), and
    - without changes (i.e. no part of its original code has been modified)
    - developed for a system (i.e. possibly, already running within a complete system)

  within a different system.
- Both the two require that: Each portion (localized syntactic part) of code *must be able to exhibit its functionalities*

Example

Semplice has not such abilities: Its programs have no portions that exhibit functionalities, since they are a unique sequence of commands

6. Properties Verification

- **Easy Verification** The language guarantees that its programs are developed in a way and, use constructs that allow the use of non-expensive techniques of properties verification.
- **Correctness** The properties to be verified must be easily identified and formulated

Example

Reliability Specific language mechanisms may be used against untrapped errors that result in unpredicted, behaviors and values (e.g. the example in the slide on orthogonality). Such mechanisms include type systems

Example

Semplice could be equipped with a sound and complete type system that guarantees an high level of program reliability.

7. Programming Environment

All languages ??of some value deserve to be equipped with a development environment that includes:

- **syntax editor**
- interpreter
- compiler
- **debugger**
- **tests generator**
- **multithreading management system** of multiple tools
- **graphical interface** for viewing windows of interaction with each running instrument

Example

The language structure allows to equip Semplice with a quite good one

8. Portability

- **Language Portability** The behavior that each language program showed, during its development, in a given platform, must be the same on each other platform on which the program can be executed.
- **Requires Language Formal Definition** Formal definition of the language allows to check for the correctness of execution tools that are designed for running in different platforms, but implement an executor of the language.
- **Virtual Machines** are a good way to reduce (at the level of the development of language tools) the difference among the platforms.

Example

"Semplice" has a formal definition.

Example

For a long time, the portability of language C has been problematic

9. Efficiency

It consider the language from the following 3 viewpoints

- **Development Cost**
- **Execution Performance**
- **Maintenance Cost**

Example

"Semplice" could be equipped with a very good executor (compiler, or interpreter). However development and maintenance will be, anyway, drammatically poor since the limits in its expressivity (see slide on application development)

Example

"Semplice" has not arrays of integers but we can encode n-tuples of integers with a single integer: Then decoding and re-coding to select items once changed. Using this kind of programming, the resulting programs will be *even more poor* in efficiency and (in points 1,5, and 6).
What can you say about the data structure implementation in abstract data type definitions?

10. Technology Supported

In the last years, new computation models are defined (for instance, quantum computing, molecular computing) and they introduce new programming languages. These languages may include operations that currently, have not a reliable technology supporting their implementation.

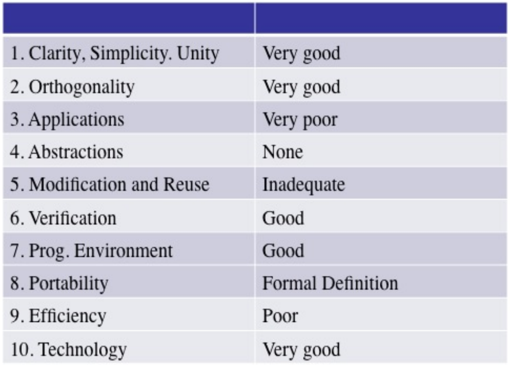
Example

"Semplice" is technology well supported.

Example

In the unrestricted DNA model of molecular computing, the operation *amplify* is not technology supported since current technology is unable to guarantee the correct computation in presence of small change in the temperature.

The language Semplice: The Table



1. Clarity, Simplicity. Unity	Very good
2. Orthogonality	Very good
3. Applications	Very poor
4. Abstractions	None
5. Modification and Reuse	Inadequate
6. Verification	Good
7. Prog. Environment	Good
8. Portability	Formal Definition
9. Efficiency	Poor
10. Technology	Very good