Programming Language Concepts

Mehdi Jazayeri
Distributed Systems Group
Technische Universität Wien
mjazayeri@alum.mit.edu
http://www.infosys.tuwien.ac.at

Outline

- Computer science environment before each edition
- Languages covered in each edition and why
- Principles in each edition
- Structure of each edition
- Today’s questions: C++, Java, C#, …?
- Comments and conclusions

Period 1975 - 1980

- IBM mainframes
- PL/I
- Algol 60, Pascal
- PDP-10 and LISP
- MJ at the University of North Carolina, Chapel Hill
- CG came to visit in 1978-79
Environment before first edition
(1980)

- Excitement about functional programming
- Excitement and apprehension about Ada
- C versus Pascal debates
- Unix and available computing (moving away from mainframes)
- Surprise: PROLOG

Environment before second edition
(1985)

- Japanese Fifth Generation Project
  - Logic programming
  - PROLOG
- Acceptance of multiple paradigms
- Lowered expectation of formal methods
  - formal semantics
- Widespread unix
- Surprise: C++

Environment before third edition
(1997)

- Dominance of object orientation
  - C++, Ada 95
- More functional programming
  - ML
- Excitement about Internet
  - Java, Java, Java, Java, Java, …
  - PCs, WWW, language implementations
- Surprise: ?
Today’s environment

- Domination of marketing
- Short term focus (time-to-market)
- ...

Choice of languages

First Edition

- Pascal
  - beauty and simplicity
  - teaching value
- Algo 68
  - blind adherence to language design principles
- Simula 67
  - fundamental support for abstraction through the class concept
- But not Ada -- why not?

Glossary of selected languages

First Edition

- Ada
- ALGOL 60
- ALGOL 68
- APL
- Bliss
- C
- CLU
- Concurrent Pascal
- COBOL
- Euclid
- FORTRAN
- Gypsy
- LISP
- Mesa
- Modula
- Pascal
- PL/I
- PLZ
- SIMULA 67
- SNOBOL4
# Glossary of selected languages

## Second Edition

<table>
<thead>
<tr>
<th>Language</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ada</td>
<td>FORTRAN</td>
</tr>
<tr>
<td>ALGOL 60</td>
<td>Gypsy</td>
</tr>
<tr>
<td>ALGOL 68</td>
<td>LISP</td>
</tr>
<tr>
<td>APL</td>
<td>Mesa</td>
</tr>
<tr>
<td>BASIC (added)</td>
<td>Modula-2 (repl. Modula)</td>
</tr>
<tr>
<td>Bliss</td>
<td>Pascal</td>
</tr>
<tr>
<td>C</td>
<td>PL/I</td>
</tr>
<tr>
<td>CLU</td>
<td>PROLOG (repl. PLZ)</td>
</tr>
<tr>
<td>COBOL</td>
<td>SIMULA 67</td>
</tr>
<tr>
<td>Concurrent Pascal</td>
<td>Smalltalk</td>
</tr>
<tr>
<td>Euclid</td>
<td>SNOBOL4</td>
</tr>
</tbody>
</table>

## Third Edition

<table>
<thead>
<tr>
<th>Language</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ada</td>
<td>COBOL</td>
</tr>
<tr>
<td>ALGOL 60</td>
<td>Modula-2</td>
</tr>
<tr>
<td>ALGOL 68</td>
<td>SIMULA 67</td>
</tr>
<tr>
<td>APL</td>
<td>Concurrent Pascal</td>
</tr>
<tr>
<td>BASIC</td>
<td>Modula-3</td>
</tr>
<tr>
<td>C++</td>
<td>Oberon-2</td>
</tr>
<tr>
<td>C++</td>
<td>Eiffel</td>
</tr>
<tr>
<td>CLOS</td>
<td>OPS5</td>
</tr>
<tr>
<td>CLIPS</td>
<td>Icon</td>
</tr>
<tr>
<td>CLOS</td>
<td>Pascal</td>
</tr>
<tr>
<td>ML</td>
<td>Perl</td>
</tr>
<tr>
<td>CLOS</td>
<td>Python</td>
</tr>
<tr>
<td>ML</td>
<td>Scheme</td>
</tr>
<tr>
<td>ML</td>
<td>SETL</td>
</tr>
</tbody>
</table>

## Additions in Third Edition

<table>
<thead>
<tr>
<th>Language</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++</td>
<td>Eiffel</td>
</tr>
<tr>
<td>CLIPS</td>
<td>Icon</td>
</tr>
<tr>
<td>CLOS</td>
<td>Java</td>
</tr>
<tr>
<td>ML</td>
<td>Modula-3</td>
</tr>
<tr>
<td>ML</td>
<td>Tc/Tk</td>
</tr>
<tr>
<td>CLOS</td>
<td>Oberon-2</td>
</tr>
<tr>
<td>ML</td>
<td>OPS5</td>
</tr>
<tr>
<td>ML</td>
<td>Perl</td>
</tr>
<tr>
<td>CLIPS</td>
<td>Python</td>
</tr>
<tr>
<td>ML</td>
<td>Scheme</td>
</tr>
<tr>
<td>ML</td>
<td>SETL</td>
</tr>
</tbody>
</table>

## Removed Languages

- Bliss, Gypsy

*indicates addition

*removes Bliss, Gypsy*
Third Edition

What added languages reflect

- C++
- CLIPS
- CLOS
- Icon
- Java
- ML
- Modula-3
- Oberon-2
- Perl
- Python
- Scheme
- SETL
- Tcl/Tk
- Eiffel
- Java
- ML
- Modula-3
- Oberon-2
- Perl
- Python
- Scheme
- SETL

Third Edition

Classification of added languages

- Object Orientation
  - C++, Eiffel, Modula-3, Oberon-2
- Internet
  - Java, Perl, Python, Tcl/Tk
- GUI
  - Tcl/Tk, Java
- Paradigms
  - CLOS, ML, Scheme
- CLIPS, OPS5
- Scripting
  - Perl, Python, Tcl/Tk
- Others
  - SETL
  - Icon

Structure of first edition

- Based on software engineering goals imposed on languages
  - Data abstraction
  - Control abstraction
  - Program correctness
  - Programming in the large
Structure of second edition

- Based on software engineering goals imposed on languages but more emphasis on language mechanisms
  - Data types
  - Control structures
  - Programming in the large

Structure of third edition

- Based on structure and structuring
  - Structuring the data
  - Structuring the computation
  - Structuring the program

Java: a good fit

- An object-oriented network-centric programming language
- Provides
  - type-safety
  - concurrency
  - modularity
- Excellent example of applying language design principles
- Finally...an American language :)!
Java: not a perfect fit

- Big language
- Pure object orientation
- Inner classes
- Lack of genericity
- ...too much hype...

My wish for a Java feature: templates

- Generic (polymorphic) components allow the raising of level of abstraction
  - queue (T)
  - sort (queue (T))
- They lead to a cleaner language
- They lead to cleaner programs
  - avoid the casting to Object
    e.g. hashtable(key, value)
  - uniformity of primitive and nonprimitive types

Lack of templates forces reliance on type casting

- A generally useful collection class such as Vector or Hashtable should be written to accept any kind of object: integer, employee, etc.
- In Java, they are therefore written to hold Object
Vector class

- Some methods of Vector:
  - public void addElement(Object)
  - public Object firstElement()
  - public int capacity()
  - public int size()
- What if we want to insert a `Point p` into vector `v`?

Need for casts with Vector

- To insert Point `p` in Vector `v`:
  - `v.addElement(p);`
- What about getting an element out?
  - `p = v.firstElement();` XXX No! XXX
- We must use a cast:
  - `p = (Point) v.firstElement();`

More problems:

- What if the vector may contain objects of different types: Points, Pixels, Boxes…?
- Need runtime checks:
  - `Object o = v.firstElement();`
  - `if (o instanceof Point) {
      // code to process Point object
    }
  if (o instanceof Pixel) { ...`
Object wrappers

- What if we want to insert int objects into Vector? Problem: primitive types are not derived from Object!
- Wrappers make objects out of primitive types:
  v.addElement (new Integer(i));
  Integer i = (Integer) v.firstElement();
  int in = ((Integer) v.firstElement()).intValue();
- C# does the conversions implicitly ...

Templates versus inheritance

- In C++, templates are used to write generic collection classes such as Vector and Hashtable
- Such generic collections can accept any type of object, including primitive types
- There is no need for casting or wrappers
- Required runtime checks in Java make the code ugly and inefficient

Principles
1st edition

- concepts -- to support sw engineering
- languages: Pascal, Simula 67, Algol 68
- Unix
- functional programming -- Backus FP
- Use of simplesem operational semantics
- list of languages
**Principles**

2nd edition

- paradigms -- more on functional but logic and rule-based also
- Fifth Generation
- formal semantics
- list of languages

**Principles**

3rd edition

- concepts -- structure
  - structuring the data
  - structuring the computation
  - structuring the program
- languages: C++, ML, Java, Ada 95
- paradigms: OO
- list of languages

**New possibilities**

Third edition

- Web site
  - http://www.infosys.tuwien.ac.at/pl-book
- Simplesem interpreter in Java
  - first edition: concepts
  - second edition: more rigorous
  - third edition: supported by interpreter
Anticipating language developments

- Inside view
  - linguistic details such as control and data structure
  - drives some (research) languages
- Outside view
  - how is the language used?

External view

- External view determines development of languages
- Context of use
  - execution
  - development
- Successful languages take the external view into account

External view

- Execution context
  - user-interfaces
    - multimedia devices
  - computational model
    - Internet
    - middleware
  - database integration
  - dealing with time
External view

- Development context
  - Visual interfaces
  - Visual languages
  - Programming by assembly (software components)

Concluding questions

- Do we know what language will emerge and dominate in the future?
- What are the factors that determine the “success” of a language, i.e. adoption by a large user community?
- Will there be one dominant language?
- Should a programming language concepts course be required in computer science?

Final word

- The study of programming languages is fun and exciting. The fun and excitement will continue…
  - Will Java kill C++, Smalltalk, and Eiffel?
  - Will C# kill Java?
  - Will there be a C* that will replace all else?