Evolutionary Trends of Programming Languages
By
Thomas M. Schorsch and David A. Cook

Mikael Malm
mmal@cs.aau.dk

27. september 2007
The Article

- On trends in programming languages
  - Introduction to the February issue with the theme: Programming Languages
- Published by Software Technology Support Center (STSC)
  - Established in 1987 by the U.S. Air Force
  - Assists the U.S. Air Force and other U.S. government agencies in software procurement and development.
The Authors

- **Thomas M. Schorsch**
  - Ph.D. in computer science from Air Force Institute of Technology
  - Thesis on “Formal representation and application of software design information”
  - Published “The Capability Im-Maturity Model” in CrossTalk, 1996
  - Last known occupation as deputy department head of CS department at U.S. Air Force Academy
The Authors

• **David A. Cook**
  ◦ Ph.D. in computer science from Texas A&M University
  ◦ Was associate professor and department research director at U.S. Air Force Academy
  ◦ Wrote several papers on various topics such as simulation and data abstractation in the context of Ada during the ‘90s (ACM)
  ◦ Has also written a number of articles on software process improvement and OO development (CrossTalk and Other).
  ◦ Currently senior Research Scientist at AEgis Technologies Group, Inc. working with verification and validation in Modeling and Simulation area.
Article Content

- Context
- Language Evolution
  - Machine Independent Programming
  - Virtual Machines
- Interoperability
- Modularity
- Scripting Languages
- Conclusion
Context

- What is a programming language
  - Tool that allow communication between developer and computer
  - Not a static thing… they evolve

- Advances by building abstractions
  - Distance between hardware and language increases
  - Distance between language and real-world decreases

- New languages allows us to express concepts in a simpler, more readable manner.
Language Evolution

- **1\textsuperscript{st} generation**
  - Machine code

- **2\textsuperscript{nd} generation**
  - Mnemonics to describe machine code

- **3\textsuperscript{rd} generation**
  - Fortran, COBOL, C++, java, C#

- **4\textsuperscript{th}, 5\textsuperscript{th} and future generations**
  - No general agreement
  - Complementing existing 3GL or DSLs
Language Evolution

- Control Structures
  - Started with simple jump statements
  - Fortran had GOTO, DO and IF statements
  - Structured Programming

- Data Structures
  - Started with data items that could be directly represented in HW
  - Then followed characters, strings, booleans, etc.
  - Then followed by general records, user defined types, dynamic data structures.
Language Evolution

- Once the individual elements were HW independent, entire languages could be made more compatible across HW platforms.
- This was one of the goals for Ada.
Language Evolution

- Virtual Machines
  - Abstract computer running on top of an existing computer
- Also achieves HW independence
- Lisp and Prolog was some of the first to run on a VM. More recently Java and C#
- In the future almost all languages will run on a VM
- New languages can produce code for existing VM
Language Interoperability

- Developer should be able to choose language depending on the problem domain
- Difficult because of interoperability
  - So developers stick with general purpose languages
  - Language designers add more features to make language more appealing
  - Even true for Ada which was designed to replace most other PL’s at DOD
Language Interoperability

- Related with development of new languages
  - C++ builds on C
  - Fortran and existing libraries
- First step is ability to make external calls to function in another language
  - Few languages support this directly
  - Ada has extensive support
- Data Representation
  - Standards such as XML makes it easier
Language Interoperability

- Another step is component interoperability
  - DLL, COM, CORBA

- Latest step is the .NET platform (CLI)
  - Common Type System
  - Brings together machine independence and interoperability
Modularity

- Designers reduce complexity by decomposition
- Initially supported by functions and user defined data types
- Then modules, packages and namespaces
- Later with object oriented programming
- The final unit of functional modularity is the framework, i.e. a GUI framework.
Modularity

• Cohesion and Coupling
  ◦ Highly cohesive and low-coupled modules preferred

• Languages provide very little syntactic support to facilitate this

• Some facets are inherently cross module boundaries
  ◦ Error handling is an example

• Aspect oriented programming addresses this issue
  ◦ Already starting to influence language design
Scripting Languages

- Glue or integration languages
  - Construct applications of pre-written components
- Originated from command languages for computer operator tasks
  - Job Control Language, Rexx
- Had major progress in late ’80s with Perl and Tcl
- Many has followed since
  - Javascript, Ruby, Python, etc
Scripting Languages

- Designed to be flexible and powerful
  - Interpreted
  - Dynamically typed
  - Forgiving syntax
  - Powerful text and input/output capabilities
- Predict that the heavy lifting will be done in system programming language by writing reusable modules
- Scripting languages will be used to create applications
Conclusion

- In near future the general trends will continue
  - Increased system-independence, modularity and interoperability
- Certain areas will continue to use domain specific languages
  - ProModel, MATLAB
- General purpose languages will evolve by incorporating new features and discarding obsolete ones
What happened after

- Modularity
  - AOP and modularity is incorporated into new languages (Scala, Fortress).

- Interoperability
  - .NET continues to evolve supporting more and more languages.
  - New languages targets existing VMs

- Platform Independence
  - Again new languages use existing VMs
My Opinion

- The good point
  - Article gives a good broad overview of language evolution
My Opinion

• The bad points
  ◦ References
    • 8 of 19 was for the small quotes between the sections
    • Java JVM reference to webopedia.org
  ◦ Questionable source and wording “Security: Because the JVM has no contact with the OS…”
  ◦ Ada fetish
    • “Ada is so portable”
    • “Even Ada had to evolve…”
  ◦ A little unfair to other languages
    • “…pointers and GOTOS are historical relics, yet they continue to be included in languages (C++)”
    • One word: “Backwardcompatibility”,
  ◦ Modularity Section
    • First little support for HCLC then next paragraph in just gets better and better
  ◦ Don’t really understand why these authors?
    • To quote Cook from his own reference:
      • “My opinion is that current language is sufficient for our needs… snip … we need tools”
To be fair

- Audience
  - Probably written both for developers, and for executives.
¿Questions?