Ada and C++ in Education

Ray Toal
Loyola Marymount University
Los Angeles

November 10, 1994

Outline

- Introductory Remarks
- The Software Engineer
- The University
- Programming Languages
- Ada before C++
- OO?
- Language Usage at LMU
- Ada Experience at LMU
- Summary

What This Talk is About

- The role of the university in educating computer scientists and software engineers
- What software engineers need to know
- The use of Ada and C++ in the curriculum
- Ada and C++ language features that help (hinder) the development of software skills
- Ada as a first language
- Provoking discussion

What This Talk is NOT About

- The history of Ada and C++
- An overview of Ada and C++
- Which universities use which languages (ask Richard J. Reid)
- The marketing of Ada and C++
- Slamming C++
- Language warfare

The Competent Software Engineer

Constructs systems whose

- **logical** design (nearly) exactly mirrors real-world objects and events
- **physical** design exhibits clear separation of concerns and is resilient to change

and can visualize the four dimensions of system design

	LOGICAL	PHYSICAL
STATIC	Class	Module
DYNAMIC	Object	Task

Roles of the University

- To "educate" computer scientists and software engineers
- To deal effectively with the difficult task of teaching students to develop **skills** and **intuition** required in megaprogramming
- To enhance students' creative thinking skills
- To expose students to a wide variety of "viewpoints" (or paradigms)

Languages and Teaching

- Industry dominated by imperative languages such as Fortran, C, C++ and Ada
- Some niches for LISP and SQL
- Some researchers allowed pleasure of working with ML, Prolog, APL, ...
- Visual languages on the rise

Thus, students need a solid understanding of imperative languages

They *do* benefit from exposure to other paradigms

(Also, they need to be able to distinguish between "languages" and "paradigms")

Domains of Common Languages

Assembly	simplify machine language
FORTRAN	numerical computation
COBOL	business
LISP	symbolic computation; AI
Algol	algorithmic description
Simula	simulation
Pascal	teaching structured programming
C	systems programming
Prolog	exprt systems; NLP
Smalltalk	workstations
LOGO	kids
C++	simulation
Ada	embedded systems; megaprogramming
ML	theorem proving

Most languages are ill-suited for applications outside their intended domains!!

Choosing the FIRST Language

Three theories:

1. **No Language**: students are first exposed to design methodology. (Okay if sufficiently formal and specifications can be executed.) Variation: use a modern visual language.
2. **Simplicity and Elegance**: e.g. ML, Haskell, Scheme, ...
3. **Something they might really use**: e.g. Ada, C, C++, ...

We must ensure students do not form first-language "biases" nor become "limited" in their way of thinking

Ada as a First Language

If an imperative langauge is used first, Ada is the best choice:

- More refined than, say, Pascal
  - superior syntax (end, return)
  - can return anything (almost) from a function
  - safe **for**-loop, variant records, case statements
- Errors are caught early
- Exceptions, Aggregates, Packages
- Ada 83 is an ISO standard
- As advanced topics need to be introduced (e.g. concurrency) there is no need to move to a new language

Why NOT Use C++ First?

- Syntax (open, type definitions...)
- 30000 + 30000 = -5536 on 16-bit machines
- Errors caught late (linker errors, even)
- IMPLICIT COERCIONS!!
- Exceptions pasted on language (not integrated)
- Overreliance on pointers
- Switches and for-loops not so nicely structured
- Module structure unsophisticated, external to language; compilation seems more independent than separate.

What About OO?

- OOT is good — OOA, OOD, OODB — provides a natural way of modeling the world
- You don't need an OOPL to implement OOA and OOD but OOF's think so
- OOFs distinguish object-based from object-oriented
- Inheritance good for extensibility and AFs, but it compromises abstraction
- IRONY OF C++: excellent object features **mixed with** (do not hide) insecure system programming foundation.

OOP and Ada

Ada 83 offers:

- ADTs through packages and private types
- Inheritance through derived types
- Static polymorphism only
- Tasks to model both active objects and resources

Ada 94 adds:

- Hierarchical libraries for superior physical organization
- Inheritance and dynamic polymorphism through tagged types
- Protected objects for resources (check these out!)

Language Usage at LMU

Philosophy: give them two years of Ada before letting them loose on C++

Some undergraduate CS courses, and featured languages of instruction:

Intro to CS (Ada)	**Programming Lab** (Ada)
Data Strs/Algs I (Ada)	**Data Strs/Algs II** (Ada) **Systems Programming** (Ada,Assembler)
Computation Theory (-) **Programming Languages** (ML,Ada,C++,Smalltalk)	**Object Orientation** (C++) **Operating Systems** (C)
Computer Graphics (C++) **AI** (LISP/CLOS,Prolog) **Software Engineering** (C++)	**Compiler Construction** (C++,Ada) **Database Systems** (SQL,Prolog) **Senior Thesis** (?)

Ada in the Freshman Courses at LMU
(1 of 2)

When introducing Ada first the CS1 teacher must focus more on program structure than "traditional" top-down algorithmic design.

with-clauses in the first example program(s) can be a good thing!

Don't teach too much of the language: but packages and exceptions are essential!

Progression of Ada-related topics:

1. A simple Ada program (subprogram!)
2. Putting your program in the Ada library
3. Subprograms
4. Writing one's own packages

Ada in the Freshman Courses at LMU
(2 of 2)

Second Semester: Programming Laboratory Course. Exercises (courtesy of P. Dorin)

1. Clock Simulation
2. Water Tank Simulation
3. Package for interfacing to ANSI.SYS
4. Tank Simulator Animation with ANSI package
5. Playing Cards package
6. Eight Queens
7. "Unbounded" Integer package
8. Fibonacci number with unbounded integers
9. "Make Change" using Dynamic Programming
10. Quicksort

Experience with Ada

Benefits of using Ada early have been realized among LMU students

- Packages and Exceptions are learned as basic, not "advanced", language constructs and are used properly
- Student programs look much prettier (are always perfectly indented) than past programs in Pascal or C
- Initially learned "good habits" in programming style carry over into C++ (comments in header files, use of readable identifiers)

Summary

- Software Engineers require certain skills
- Universities must enhance the development of these skills
- There are different theories regarding the use of programming languages in the cirricula
- Ada should be taught before C++