

# Ada in Research and Education, an Experience Report

Prof. Dr. Erhard Plödereder  
University of Stuttgart, Institute for Software Technology  
[ploedere@informatik.uni-stuttgart.de](mailto:ploedere@informatik.uni-stuttgart.de)

(with some slides loaned from Franco Gasparoni, AdaCore)

- The University of Stuttgart started using Ada selectively in First Year Computer Science and Software Engineering courses in 1998 ...
- and has been using Ada for the entire First Year education in "practical informatics" (which serves both CS and SE) since about 2000
- these courses add up to 21 ECTS points

# Why use Ada? (in industry and academia)

## Part I: Some facts

## Error rates (per kSLOC)

	Ada	C	C++	norm
Information Systems (kom.)	4,0	7,0	5,1	7,0
Information Systems (mil.)	3,0	6,0	4,0	6,0
Commercial Products	2,8	5,0	3,0	4,0
Telecommunication (kom.)	1,6	2,0	1,7	3,1
Telecommunication (mil.)	1,0	1,5	1,2	2,5
Weapon Systems (ground)	0,5	0,8	0,7	1,0
Weapon Systems (air/space)	0,3	0,8	0,6	1,0

Source: Reifer 1996, 190 projects 1993-96, norm = avg. of 1500 projects 1989-96

## "Mean Time (weeks) to Major Repair Incident"

	Ada	C	C++	Norm
Commercial Products	1,0	0,4	1,0	0,5
Information Systems (mil.)	0,8	0,5	k.A.	0,5
Information Systems (kom.)	1,0	0,5	0,6	0,5
Telecommunication (kom.)	3,0	1,0	2,0	1,8
Telecommunication (mil.)	4,0	2,0	3,0	2,0
Weapon Systems (ground)	6,0	3,0	k.A.	2,5
Weapon Systems (air/space)	8,0	3,0	k.A.	2,5

Source: Reifer 1996, 190 projects 1993-96, norm = avg. of 1500 projects 1989-96

from 1. to 3. project in Ada:

- Reifer 87: +25% SLOC/PM productivity gain
- NASA SEL 89: +50% productivity gain, -40% less errors

other evaluations of productivity:

- NASA SEL 89: ~700 SLOC/PM; 1,0 error/kSLOC
- ESA DB 96: ~400 SLOC/PM (vs. C: ~260 SLOC/PM)
- Reifer 87: 121-415 SLOC/PM

Reifer 87: evaluation of 41 Ada projects

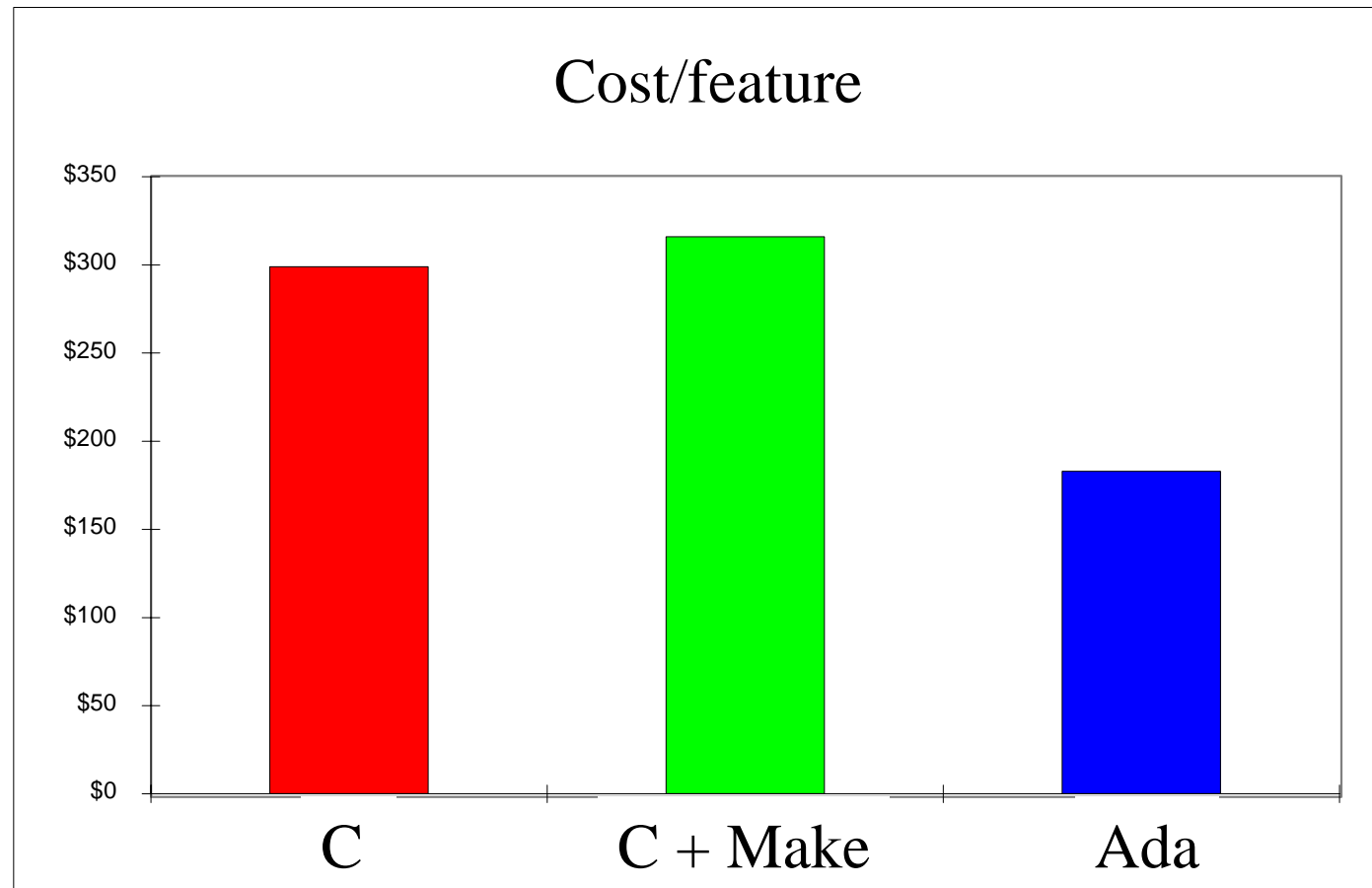
NASA SEL 89: Report to the Information Resources Mgmt. Council

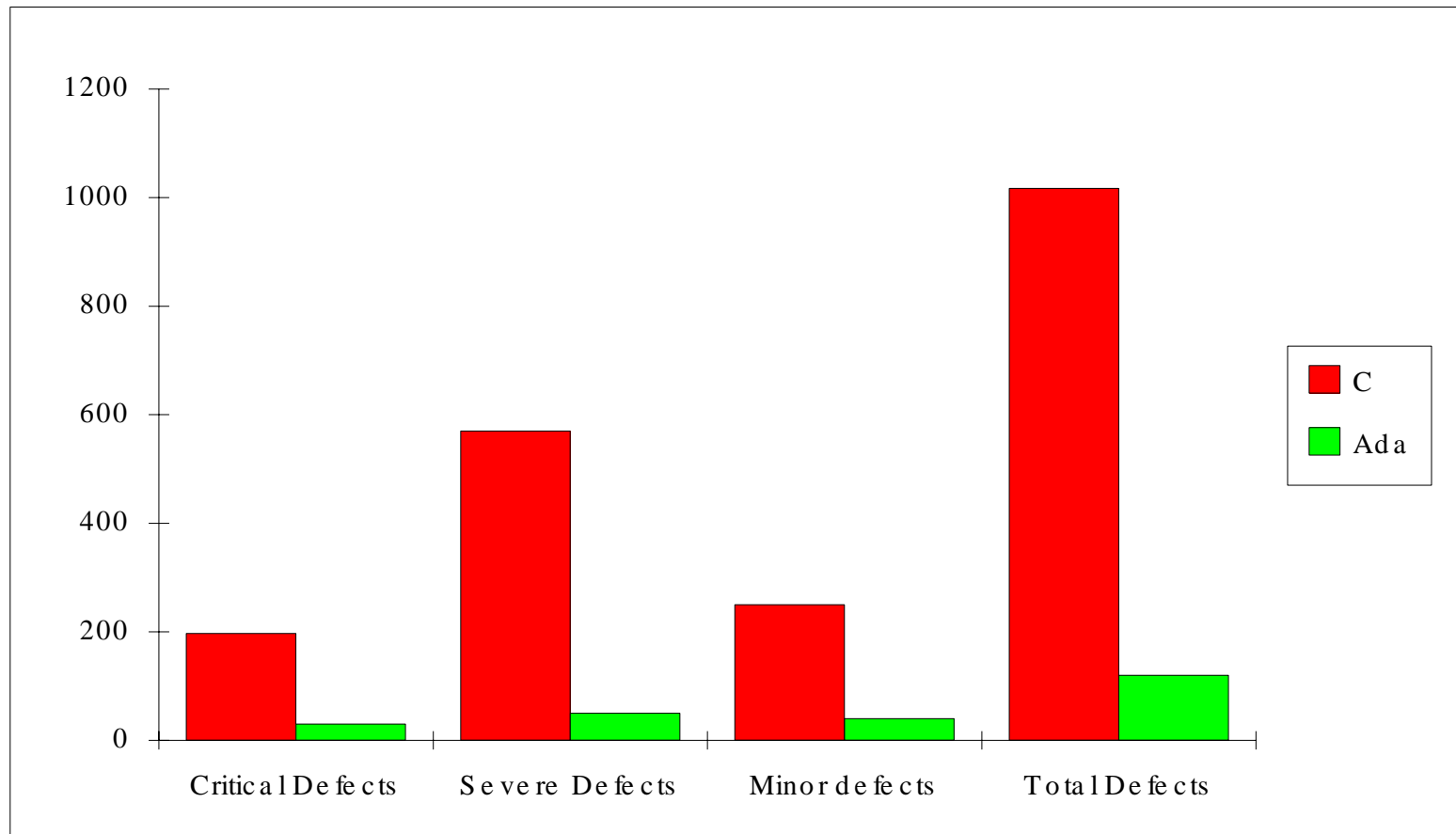
ESA DB 96: Maxwell et al (IEEE TSE, V22/10, 1996): 99 ESA Projects, 34 Ada, 9 C

- Reifer 87: development costs in Ada are **sublinearly** (!!)  
(exponent: 0,95) dependent on project size.
- ESA DB 96: Highest correlation of productivity is with the  
characteristic "Company" and then "Programming Language".

- **1995 study on the VADS compiler**
  - 60 engineers, from 1984 ..1994 with MS degrees in computer science
  - All knew C at hire. All programmed in both C and Ada.
- **VADS**
  - About 4.5 million lines of code,
    - 2 million LOC Ada
    - 2 million LOC C
    - 0,5 million LOC Miscellaneous
  - 22000 files

# Cost per Product Feature





- **Not because of people:**
  - The same people used both languages
- **Not because of process:**
  - The same process was used, for design, for testing, for debugging, for source control, for management, and so forth
  - C required ‘makefiles’, but had tighter coding standards
- **Not because of Ada's highest level constructs:**
  - VADS used few generics or tasks
- **Not because of reuse:**
  - This study considers only unique code, factoring out reuse

- **Ada Enabled Better Error Locality**
  - Most errors caught at compile-time
  - Runtime errors are easier to trace
- **Ada Enabled Better Tool Support**
  - Ada's richer semantic model allows computers to help more
  - For example, builds are automated and guaranteed consistent
- **Ada Reduced Effective Complexity**
  - Function of language complexity and application complexity
  - Standard language complexity is easier to learn and use
- **Ada Encouraged Better Program Organization**
  - Packages, with specifications and private parts

- Developing software in Ada is 60% cheaper than in C
- Code developed in Ada has 9 times less bugs than in C
- Was Ada consistently better? **\*YES\***
  - Over different subsets of VADS
  - For experienced AND inexperienced programmers
  - For both C experts AND Ada experts
  - For the highest AND lowest rated programmers
- Was Ada harder to learn? **\*No\***
- Was Ada code more reliable? **\*YES\***
- [http://www.adaic.com/whyada/ada-vs-c/cada\\_art.html](http://www.adaic.com/whyada/ada-vs-c/cada_art.html)

Why use Ada?  
(in industry and academia)

~~Part I: Some facts~~

*unfortunately mostly ignored*

...

## Part II: The real concerns

- *"We are a YYY shop. We want YYY programmers."*
- *"We use libraries written in YYY. Therefore we code in YYY."*
- *"YYY programmers are easy to find (and cheaper)."*
- *"We hear so much about YYY in the trade press. We want to be part of the community."*
- *"Our community programs in YYY. Therefore we do, too."*

YYY can be instantiated by many language names, notably C++  
Java, C#, but also COBOL, FORTRAN or Ada.



- Boeing 777 (99% of avionics in Ada)
- Airbus A330, A340 Flight Warning System (FWS).
- BE-200 (avionics), Ilyushin 96M
- IMD-HUMS (Helicopter Usage and Maintenance System) on CHE 53E, A109K2 (Swiss REGA), AS350/355 (Spain) , Sikorsky SH60, CH53, S76, S92
- ISS Space Station; robot arm
- GPS on ESA Hermes
- ESA Ariane 4/5



- New York City Metro Carnarsie Line
- Paris Metro, Ligne 14 (METEOR)
- London Metro, Jubilee Line
- Cairo Metro
- Calcutta Metro
- SNCF Position and Transmission System
- Swiss Railroad
- TGV France
- EuroStar (Chunnel)

- Schweizer Postbank, electronic bank transfers
- Weirton Steel Mill plant control
- Pratt-Whitney F119 und PW600 engines
- Manufacturing Resource Planning, Muar Furniture Industries
- Temelin nuclear power plant (backup shutdown system)
- Vision Systems Video surveillance system
- Kingcat Yacht – all control and navigation systems
- Helsinki Radio Telescope control
- BMW Driver Assistance System (Research)

# Teaching Languages in Universities

## Part III: Academic Goals

- **There is a need to be educated in particular languages.**

The answer is that programming courses ought to be offered for each such language.

- **There is a need to be educated in programming language concepts repeated across the landscape of languages.**

The question is whether a programming course in X can explain the concepts of Y and Z.

**I claim that the answer is between "No" and "Not well".**

*"Language can corrupt thought."* **George Orwell**

**but someone else (who?) also observed more or less that**

*what cannot be expressed in language  
can not be thought about.*

*So, which language(s) to use to convey language concepts ?*

- a series of simple languages, each providing for a small number of concepts ("13 languages in 13 weeks") ?
- a "complicated" language that provides a uniform framework to experience many concepts ?

- Ada is designed rigorously to achieve reliability and to support modern real-time systems. It covers most of the concepts worth talking about within one language framework.
- Java is designed rigorously to achieve security in execution, but has substantive problems with reliability and efficiency. It is simple and parsimonious in concepts, but very one-sided towards OOP. Many concepts tie into "magic" from the complicated and often badly documented library.

- C is rigorously designed for efficiency and simple implementation, but shows no concern for security, safety, and reliability. It is reasonably straightforward, but lacks many modern concepts. Using it as a first language in education is therefore close to irresponsible.
- C++ is a rich language but has all the flaws of C in its core language. It does not provide for concurrency or modularization. Its OOP concepts are "unusual" in the field of OOP (value semantics for objects, only virtual methods dispatch, etc.).

```
public final class SynchNode {  
    private int incoming = 0;  
    private int outgoing = 0;  
    private SynchNode[] links;  
  
    public SynchNode() {links = new SynchNode[20]; ... }  
  
    public synchronized void link(SynchNode X) throws  
    aSeriousFit {  
        if (outgoing >= 20) {throw new aSeriousFit();};  
        links[outgoing] = X;  
        outgoing++;  
        X.incoming++;  
    }  
}  
.... SynchNode A,B,C;  
A.link(B); and C.link(B); executed by two concurrent threads  
may corrupt the incoming counter of B !
```

object-based  
synchronisation  
plus  
class-based  
encapsulation  
=  
seriously flawed  
monitor concept  
=  
very basic  
design flaw in  
the language

# the example in Ada

```
type Synchnode;  
type Linkage is access Synchnode;  
type LinkArray is  
    array(Natural range <>) of Linkage;
```

```
protected type Synchnode is  
    procedure Link (To : Linkage);  
private  
    Incoming: Natural := 0;  
    Outgoing: Natural := 0;  
    Links: LinkArray(1..20);  
    procedure Increment_Incoming;  
end Synchnode;
```

```
.... A,B,C: Synchnode;  
A.link(B); and C.link(B);  
works just fine
```

```
protected body Synchnode is  
    procedure Link (To : Linkage) is  
    begin  
        if Outgoing >= 20 then raise Aseriousfit;  
        end if;  
        Links(Outgoing) := To;  
        Outgoing := Outgoing + 1;  
        To.Increment_Incoming;  
    end Link;  
  
    procedure Increment_Incoming is  
    begin  
        Incoming := Incoming + 1;  
    end Increment_Incoming;  
end Synchnode;
```

defined in appropriate classes:

```
.... int echoMeasurement(direction X) {...}
      // returns the distance from the target
.... float calculatedThrust(int distance) {...}
      // calculates thrust to cover the distance
```

```
int curDistance;
curDistance = target.echoMeasurement(Y);
propulsion.feed(position.calculatedThrust(curDistance));
```

Is this good code?

**Mars Climate Orbiter crashed on Mars because of it**

*Disclaimer: not the real software (but the identified cause of the crash)*

defined in appropriate places:

```
typedef int meters;  
typedef int feet;  
typedef float thrust;  
.... feet echoMeasurement(direction X) {...}  
.... thrust calculatedThrust(meters distance) {...}
```

```
feet curDistance;  
curDistance = echoMeasurement(Y);  
propulsionfeed(calculatedThrust(curDistance));
```

Is this good code?

Users might notice the problem in a code review.

The compiler unfortunately finds everything just fine.

defined in appropriate places:

```
type Meters is new Integer;  
type Feet is new Integer;  
type Thrust is new Float;  
... function echo_measurement(X: Direction) return Feet;  
... function calculated_thrust(distance: Meters) return Thrust;
```

```
current_distance: Feet;  
current_distance = echo_measurement(Y);  
propulsion.feed(calculated_thrust(current_distance));
```

Is this good code?

Users might notice the problem in a code review.

**In any case, the compiler refuses to compile the code.**

- Ada allows user-defined numeric types such as *meter, feet, inch, cm, liters, minutes, seconds, newtons, horses, a.s.o.* and checks type compatibility at compile time upon assignments or parameter passing.
- C/C++ provides mnemonic typedefs *for meter, feet, a.s.o.,* but consistency is not enforced by the compiler.
- Java has only „int“. (To get type-checked values, wrapper classes are needed. Issues: reference semantics, a major conceptual inversion in language idiom)
- [Similar statements apply for reals, enumerations and their type safety.]

- What is the result of multiplying the two integers  
 $10^{**6}$  and  $10^{**6}$ ?  
  
**= -727379968 ??**
- Java and the C-family on 32-bit architectures say so.  
(My bank and my math teacher believe differently.)
- Ada raises "Constraint\_Error" for this overflow on 32-bit architectures.

**Bear in mind:** You need to explain these topics to the students in terms of presumably good and simple concepts

- `array[1]` denotes the second(!) array element

Source: "Seven Deadly Sins of Introductory Programming Language Design",  
McIver and Convey

**Bear in mind:** You need to explain these topics to the students in terms of presumably good and simple concepts

- Ada provides modularisation and subsystem structures as integral part of the language. The C-family has no support beyond file boundaries. Java relies on directory structures (and has subtle information protection bugs).
- Ada and Java check interface consistency across compilations.
- Ada prevents linkage of inconsistent linkage units. "Make" is built into the compilation systems.

- most traditional data types in a strong and versatile typing model
- polymorphic object-oriented programming
- all interesting structured control structures
- exception handling
- system structure mapping (subunits, child units, privateness at type, module and subsystem levels)
- generics
- integrated task (=thread) model with high- and low-level communication primitives
- fully defined flexible real-time scheduling
- guaranteed floating-point accuracy

**Ada provides a good unified framework to cover a large set of concepts in programming languages.**

**First Year students appreciate to have to deal only with a single style of syntax and a single set of tools.**

**Subsequent pick-up of other languages is fast.**

# Teaching Languages in Universities

~~Part III: Academic Goals~~

*unfortunately not top priority ...*

# Part IV: The real concerns in academia

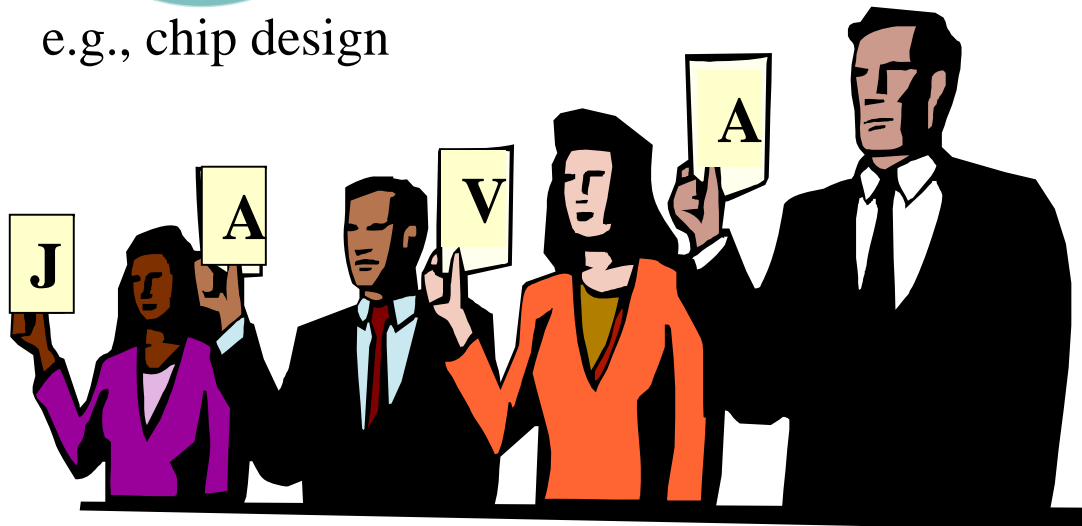
- *"We are a YYY group. We need YYY programmers."*
- *"We use libraries written in YYY. Therefore we code in YYY."*
- *"My students need to program in YYY. Therefore education in programming needs to be in YYY."*
- *"We read so much about YYY in publications. We want to use what is hot."*
- *"Our community programs in YYY. Therefore we do, too."*

YYY can be instantiated by several language names, notably C++  
Java, C#, and occasionally FORTRAN, Ada or some scripting language.

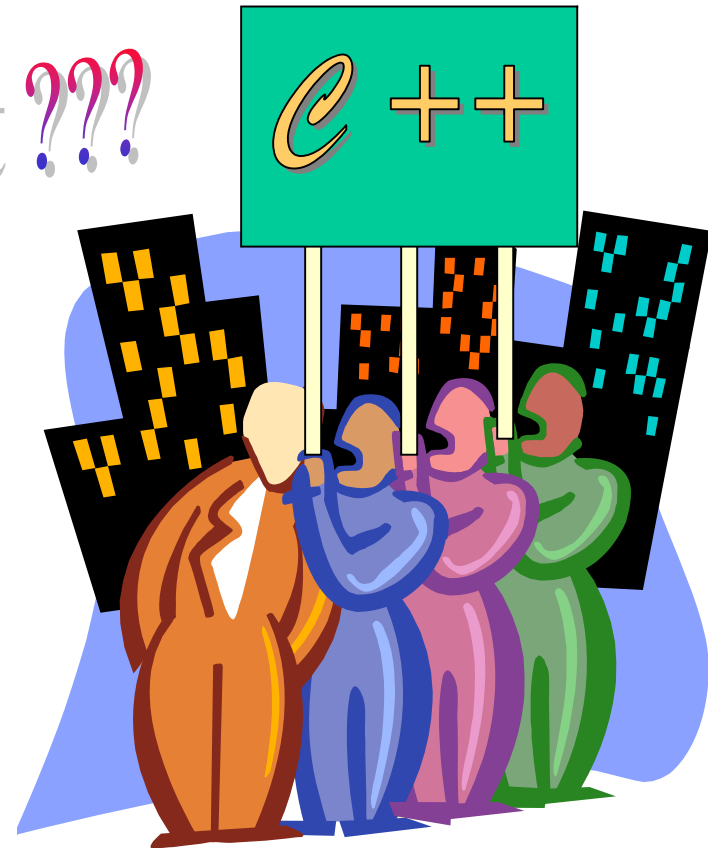


e.g., chip design

??? Now What ???



e.g., web-services; communication

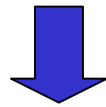


e.g., graphics, scient.comp.

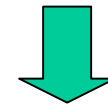
and the answer is ...

# Ada

1. year teaching of the concepts



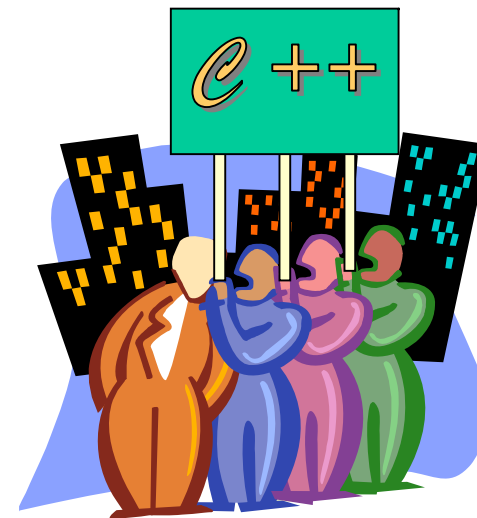
Java programming course



C++ programming course



e.g., web-services; communication



e.g., graphics, scient.comp.

- **from Ada to C++, C, or Java:** fairly easy (except concurrent code)
- **from C++ to Ada:** occasionally with difficulties in working at higher abstraction levels, but manageable
- **from C to Ada or Java:** "only the good survive"
- **from Java to Ada:** mostly hopeless (Java code and semantics force-fed into Ada syntax)

- generally favorable  
(... amazingly more so than by the colleagues in any one the mentioned language camps)
- mitigated by the pragmatic realization that they **HAVE** to know C++ or Java because the market demands the skill



- "Bauhaus" is a tool collection for analysing software (mostly C++, C, Java), written mostly in Ada
- both a research and teaching platform since 1996 and a commercially sold system since 2003
- today, jointly maintained and enhanced by groups at the University of Bremen (Prof. Koschke), at the University of Stuttgart, and at Axivion GmbH
- at last count, about one million lines of Ada code
- in excess of 100 person years of effort
- Bauhaus analyses large programs (into the million LOC range) with acceptable performance



- at times, more than 10 professional developers and 20 students worked concurrently on the system
- five major one-year projects by student teams of 6-9 persons\* added about 180.000 lines of Ada production code to Bauhaus (one control project with equal tasking using Java failed solidly)  
\* reported in Ada-Europe 2005, Springer LNCS 3555, S. 115-128, 2005
- numerous thesis projects were conducted in the Bauhaus context
- "we could not have done it without Ada (and a strong CM)" is the general opinion

- Be clear about objectives: teaching a language in a programming course vs. teaching programming in general
- Be aware of your neighbors' goals
- A lesser number of concepts supported by a language is not an advantage when teaching programming in general
- Ada is a really good and proven candidate for teaching programming



Thanks for listening

Any questions?