Teaching 'Concepts of Programming Languages' with Ada

Theodor Tempelmeier

University of Applied Sciences Rosenheim

17th International Conference on Reliable Software Technologies – Ada-Europe 2012

Stockholm, Sweden, June 11-15, 2012

Hamburg
Berlin
Frankfurt
Munich
Rosenheim

Rosenheim:

A small city (pop. ~ 60000)
in the southern part of Germany
close to Munich, close to the Alps

Motivation

… for Doing this Course at the University

- promote usage of Ada
- being convinced that my students can learn a lot from studying Ada

… for this Contribution

- promote the inclusion of teaching topics in this conference series
- encourage teachers to *use Ada even under difficult circumstances*

A Few Initial Remarks

- **Just a personal experience report**
  No claim that the presented method of teaching is better
  (in what sense?) than yours

- **If you like it: Good!   You may copy some of the presented ideas**

A Few Initial Remarks

- **Just a personal experience report**
  No claim that the presented method of teaching is better
  (in what sense?) than yours

- **If you like it: Good! You may copy some of the presented ideas**

- **If you don’t like it: Also good!**
  Present your contrasting ideas at next year’s conference!

1 Introduction

University of Applied Sciences
(Fachhochschulen, Polytechnics, …)

- **Degrees:**
  7 semester bachelor program
  3 semester master program
  no doctorates awarded
- **Typical course:**
  4 semester credit hours, 60 contact hours, 5 ECTS credit points
  2 hours lecture or seminar-like tuition, 2 hours practical per week
- **Very strong focus on practical applicability**
  è teaching mostly centred on mainstream programming languages
  C, C++, C#, Java

Practical applicability!

1 Introduction

Practical applicability!

actical applicabil

1 Introduction

cal app

1 Introduction

2 Design of the Course “Concepts of Programming Languages”

2 Design of the Course “Concepts of Programming Languages”

- **Pre =>**

  Students have a lot of programming experience

  (6 semesters of C, Java, …, various project assignments, etc.)

- **Post =>**

  Students have a deep understanding of *some* of the concepts of programming languages

2 Design of the Course “Concepts of Programming Languages”

- **Introductory chapters:**
  history, COBOL, FORTRAN 77

- **Decision:**
  Only one language in the practical
  (consequently also to be used in the lectures as the central theme)

- **Decision:**
  Ada is used as the central theme

Σ Single language approach to teaching
“Concepts of Programming Languages”

Reasons for Choosing Ada

- **Richness of concepts in Ada**
- **Bring a new, different world to the students**
- **Free excellent compiler, free ISO standard**

Why not a functional language?

Separate elective module about functional languages

Biased towards applications in technical systems, esp. embedded and safety-critical systems

Emphasis on …

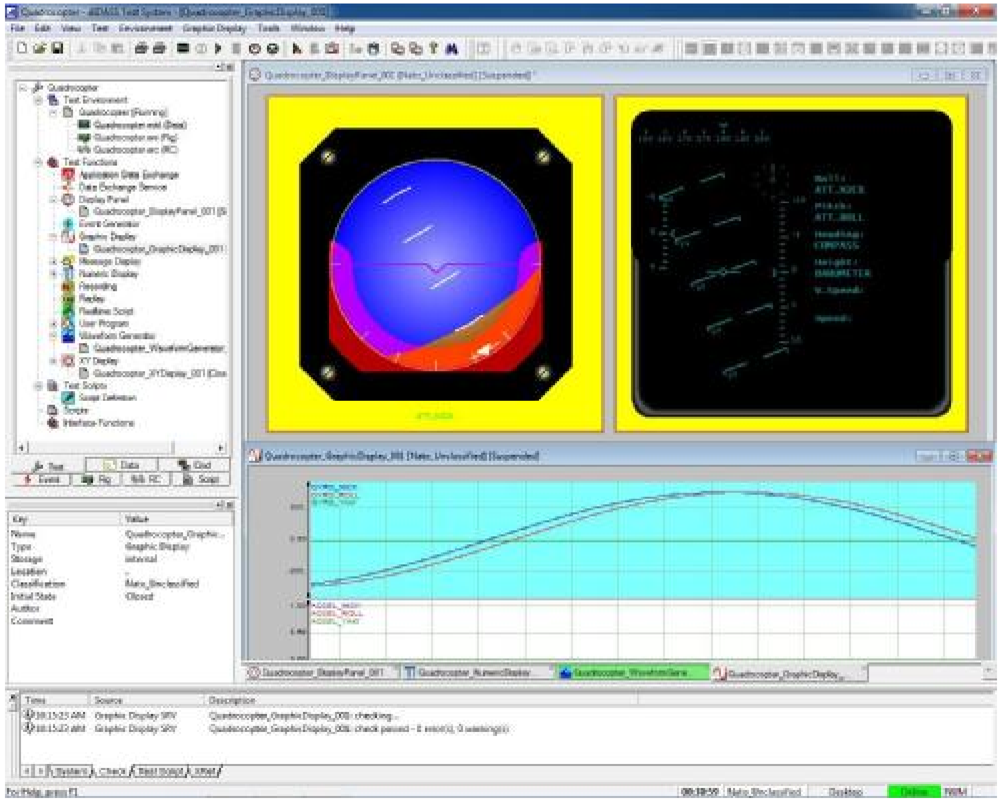
- **Type systems**
- **Packages**
- **Generics**

Selection of these topics based on

- **experience from industrial projects in the embedded domain**
- **25 years of teaching experience**

Example: Importance of type systems

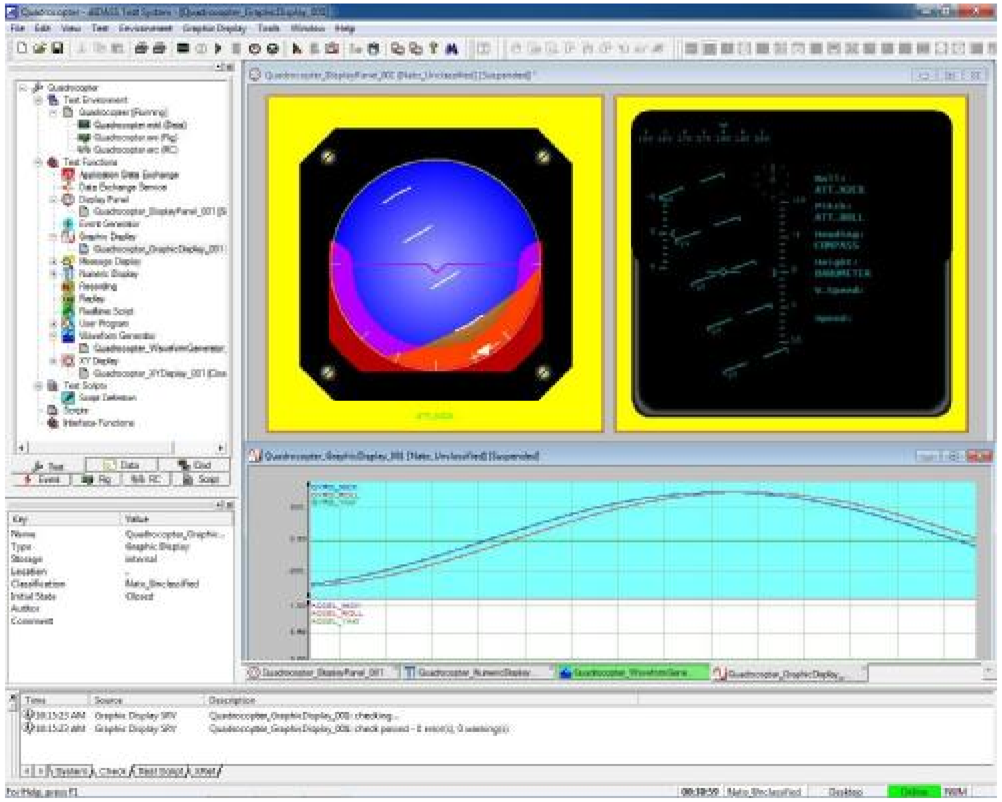
Students Using a Test Tool of Aerospace Industry


File Edit View Test Environment Graphic Display Tools Window Help
Quadrocopter
Test Environment
Test Functions
Test Scripts
Key
Value

What is important?

What causes trouble?

Students Using a Test Tool of Aerospace Industry



What is important?

What causes trouble?

Get the data types right!

Binary and decimal fixed point data types,

Integer and float data types,

Size and layout of data types, etc.

Comparison to Other Languages

- Course as described so far is only an Ada course at first glance, but …
- All Ada concepts are always compared to other programming languages
- Many asides, discussions, etc.
- Self-directed learning (see next chapter)
- Cross-references to other courses (see chapter after the next)

3 Course Segment “Questions and Discussions”

(è Self-directed learning)

3 Course Segment “Questions and Discussions”

- Started a number of years ago with only a few items
- Now about fifty topics
- May now well be deemed the most important part of the course
- Idea behind it:
  Guide the students to think about (and discuss)
  these topics for themselves
- Examples: see next slides

Examples: easy level

A student’s error in the first semester practical on programming - what is (probably) wrong?

```
for (i=1;i<max;i++);
   {
   ...
   }
```

Examples: medium level

Which style guide is right?

- ***“A switch statement must always contain a default branch which handles unexpected cases.”***
- ***“Never use an others choice in a case statement.”***

What are the “best” strings of the three categories discussed (fixed-length, bounded-length, unbounded-length)?
Hint: Where are strings to be stored?

What should a pacemaker do when new raises storage error (or bad_alloc)?

Examples: medium level

What happens in `x:=x+1.23;` (Ada)

or in `x=x+1.23;` (C, C++, Java) respectively?

(x be of type float)

How should `x+i` be computed (x be float, `i` be integer)?

Should the required type as in `variable_of_some_type := x+i;`

be considered?

How about an overloaded function call in the expression?

Examples: difficult level

In Ada the short-circuit operations `or else` ***and*** `and then` ***formally are no operators, and they cannot be overloaded – why?***

What does the following code do?

How can it be that no code at all is generated with optimizations turned on?

```
inline unsigned64 Swap_64(unsigned64 x) {
  unsigned64 tmp;
  (*(unsigned32*)&tmp)= Swap_32(*(((unsigned32*)&x)+1));
  (*(((unsigned32*)&tmp)+1)) = Swap_32(*(unsigned32*)&x);
  return tmp;
}
```

Examples: difficult level

Why does Ada have two dots in a range `(1..10)`,
VHDL on the other side uses the reserved words `to` ***and*** `downto`,
e.g. `(1 to 10)` ***or*** `(10 downto 1)`?

Why does Ada use `in out`, ***while VHDL uses*** `inout` ***(without blank)?***

Programs for Trying out and Discussing

- Execution of a loop

```
N := 4;
for I in 1..N loop
   put(I);
   N := 10*N;
end loop;
```

- “Evil Pointers” adapted from the book of John Barnes, also in a C version (see proceedings)

- …

4 Cross-References to other Courses

4 Cross-References to other Courses

- **From the Section “Questions and Discussions”**

Given a parameter of type

`access function (l,r : integer) return boolean`***,***

why is it not possible to use `">"'access` ***as an actual?***

(Error message is "prefix of access attribute cannot be intrinsic")

- **References to the course on safety-critical systems**

- **Quicksort in functional style in Scala – done in Ada**

5 Evaluation of the Course

5 Evaluation of the Course

- **How to Evaluate a University Course?**
  - **feedback from peers**
  - **feedback from industry**
  - **feedback from the students**
  - **self-reflection of the lecturer**

- **Comparing the success of the course to other courses on an objective scale: hard or impossible**

5 Evaluation of the Course

- **How to Evaluate a University Course?**
  - **feedback from peers (+)**
  - **feedback from industry (+)**
  - **feedback from the students (+)**
  - **self-reflection of the lecturer**

- **Comparing the success of the course to other courses on an objective scale: hard or impossible**

Self-Reflection of the Lecturer

+ Course is specifically tailored to programmers of technical, embedded, and safety-critical systems, giving practical help in everyday programming, even including parts of VHDL

○ Functional and logic programming languages not treated
Mitigated by a separate course on Functional Programming

− Omission of the synchronous programming paradigm

6 Conclusion

6 Conclusion

- **Using Ada as the central theme: successful, even with a focus on immediate practical applicability as demanded at Universities of Applied Sciences**
- **The presented course has a certain bias towards technical, embedded, real-time, safety-critical systems**
- **Good idea: complement such a course with a course on functional programming languages**

6 Conclusion

Thank you!

Questions?

Comments?

7 Supplements

7 Supplements

```
-- Quicksort in functional style in Scala [25]


-- def sort(xs: Array[Int]): Array[Int] = {
-- if (xs.length <= 1) xs
-- else {
-- val pivot = xs(xs.length / 2)
-- Array.concat(
-- sort(xs filter (x => pivot > x)),
--      xs filter (x => pivot == x),
-- sort(xs filter (x => pivot < x)))
-- }
-- }
```

```
with predicates; use predicates; -- not shown
with filters; use filters;       -- not shown
-- ArrayInt is defined with index type natural

function sort (xs : ArrayInt) return ArrayInt is
begin
  if xs'length <= 1 then  return xs;
  else
    declare
      pivot : constant integer := xs(xs'length/2);
    begin
      return sort(filter(xs,  pivot,  greater)) &
                  filter(xs,  pivot,  equal)    &
             sort(filter(xs,  pivot,  less));
    end;
  end if;
end sort;
```

```
procedure Evil_Pointers is
  type P_Object_T is access all Integer;
  Evil_Obj_P : P_Object_T;
  procedure P (Objptr : access Integer) is
  begin
    Evil_Obj_P := Objptr;
  end P;
begin
  Put_Line ("Let's start!");
  declare   ---------------------- nested block
    An_Obj : aliased Integer;   -- |
  begin                         -- |
    P (An_Obj'access);          -- |
  end;      ---------------------- end of nested block
  Evil_Obj_P.All := 123;
end Evil_Pointers;
```

```
-- How to compile without errors?
-- Maybe "p_objec_t" instead of "access integer"  ??
-- Maybe a type conversion ... := p_object_t(objptr) ??
```

```
typedef int* object_p_t;
object_p_t  evil_obj_p ;
void p(int* objptr){
  evil_obj_p = objptr;
}
int main (){
  void x (){        // local function instead of
    int an_obj;     // the nested block
    p(&an_obj);     // (possible in GNU C)
  }
   printf("Let’s start!\n");
   x();
   *evil_obj_p = 123;
   printf("Result: %i\n",*evil_obj_p);
   return 0;
}
```

7 Supplements