An experiment in the use of Ada in a course in Software Engineering

Guy Lapalme, Jean-François Lamy

Département d'informatique et de recherche opérationnelle
Université de Montréal
C.P. 6128, Succ. A
Montréal, Québec
Canada, H3C 3J7

SUMMARY:

This paper describes our experience in using Ada as a vehicle for teaching Software Engineering concepts in a course for first year undergraduate students at the University of Montreal.

We first review the curriculum at our university and then give an idea of the hardware and software at our disposition. We describe the goals we had in mind in using Ada as a teaching language and then we detail the topics and the assignments chosen in our course. We conclude by describing a few lessons learned from that experience. All in all, we are very satisfied with this experiment and we intend to carry on next year.

Academic Environment

Our department has around 500 computer science undergraduate students and the curriculum is loosely patterned on the "Curriculum'78" of ACM [1]. In their first semester, our students take a course in programming using Pascal (CS1) and another where assembly language is used. After these courses, they are able to create, modify and debug small programs (200 lines or so...). They have mastered the basics of control structures, procedures and parameter passing. In the second semester, a second programming course teaches advanced topics in programming and puts emphasis on the construction of systems instead of programs. In this way we want to deepen the students' understanding and knowledge of basic programming abilities earned in their previous courses and also confront them with more advanced approaches to programming than those possible with Pascal.

The course was taught for two years using Simula 67. Last year we chose to go with Ada. Evans and Patterson [5] give many reasons to use Ada in undergraduate courses and Agrawal and Harringer [2] point out the needs for such courses to help solve the "software crisis".

Organisation

We had 120 students enrolled in our course. Each week, three one-hour lectures were given and there was a weekly 4 hour period where a group of 40 persons had exclusive access to a VAX/780 running the Telesoft Ada compiler version 1.3 [6,7]. We had three such periods booked each week. During each of these periods, teaching assistants were present to help the students run their programs; so there was a scheduled period where each student was guaranteed a "fair" share of machine power. Outside that period, and on weekends, the machine was available on a first come, first served basis. Three other courses used similar arrangements. Of course, the day before the assignments were due, there was something of a "free for all".

The compiler we used was quite slow (by about a factor of ten compared to a Pascal compiler) so we suggested that Ada compilation be done by batch submission; for a while, we even organized the priorities of the system so that a batch program had better priority than an interactive one except for the system text editor. As this resulted in batch compilations finishing earlier than "interactive" ones, we did not need to insist further on that point... With that set-up, turnaround time for the compilation of 400 lines chunks varied from about half an hour to more than 2 hours. All in all, this was a good experience, for us at least, given that it was a first course using a new compiler. Everybody had to learn on the spot, including us and the teaching assistants which never had previously programmed in Ada. This was a "partial compiler" but as we will see in the next section, the major ideas of modern software engineering were implemented.

Goals and methods used in the course

The goal of our course is to give the students a feeling of what it is to build a system instead of only one program, that is, to program in an environment where there are other programs, other programmers, users, bosses, etc... So we want them to be aware of the tools needed to build such systems reliably, namely modules and other advanced programming techniques. Another goal is to make them aware that there are many interesting programming language concepts which are not present in Pascal but which help make programs simple and efficient (e.g. dynamic arrays, block structure, exceptions, etc...)
Table 1 gives an outline of the topics covered in the course and the programming assignments

Week no	Lecture	Programming* Assignments
1	Top-Down Programming	
2	Modular Programming	Ada vs Pascal
3	Packages	Translating a Pascal algorithm to Ada
4	Types in Ada	Give the body of a package for doing (2weeks)
5	Pointers	date manipulation
6	Variant records	Give the body of a (2weeks) DeQue Manipulation package and
7	Mid-term Exam	use it to simulate a card game
8	Dynamic arrays	Use the ideas in (3weeks) previous packages to build
9	String manipulation	a bank account managing system where daily interest is to be computed
10	Exceptions	
11	An example of a complete package;TEXT_IO	Implement a robust (3weeks) user-interface using exceptions for the bank account
12	Run-Time organisation	system of the previous assign-
13	of computer programs	ment. Tests had to be designed by the students and were graded.

Table 1. Outline of the course

* The programming assignments cannot start on the first week because of administrative delays in assigning user accounts on the computers.

In this plan we had the programming assignments follow very closely the subjects of the lectures. Also we tried to have the students build on their previous assignments for their next one; i.e. the fourth assignment uses the package developed in the second and third, and the fifth is a refinement of the fourth.

The set of topics were selected because of their interest (e.g. packages, exception), or because they were extensions of Pascal concepts that had been introduced in the previous course but could not be well assimilated because of time constraints ("advanced" types in Ada, pointers and dynamic arrays). Here we took the opportunity to reintroduce these concepts in Ada comparing with those in Pascal and afterwards to cover the Ada extensions. Another interesting aspect, generic packages and subprograms, was only shown briefly because they were not implemented in our compiler so the students could not use them in their programming assignments. Tasking was partially implemented but was not covered because of time limitations.

Experience

The course was well received by the students who felt that they were at the forefront of computer science techniques; in fact our department was the first to use Pascal in courses outside ETH in Zurich in 1971. Telling the students the facts in this way helps sell a "partially bugged" partial Ada compiler; so every one "gets around" features that are implemented either badly or differently from the standard or not at all.

The biggest advantage we felt were the packages. Private, limited private and hidden

types are very useful for keeping modularity principles intact. The compiler ENFORCES this discipline; in Pascal for instance, it is too easy to put more global variables than necessary just to patch things up; the only tangible incentive the students have in this case is that when their assignment is graded they might lose a few points because of a lack of modularity. In Ada, in this case, the program will not even compile unless we explicitly put the variables in the specification where they are more evident. Moreover, on two assignments the package specification where given and could not be changed. This gave a modularity reflex which helped very much: the students often asked themselves and us appropriate questions as to where each variable or procedure should go, what should be their status, etc.

The package feature is very useful because once a specification is given, there is no way to change its interface with the outside world (this could not really be implemented in our case because our compiler required compiling both specification and body at the same time). Given the specification, a test program can be devised to test the body given by each student. It is also very easy to build on top of previously defined packages because of their well defined interface.

Of course, these advantages have to be paid for in some way: the compiler we used was slow but usable (much better than the next, validated one, (version 2.2), which seems to be another 10 times slower...). But the main limitation was that it was a partial compiler, so quite often we had to give two versions of a program, one in good Ada and another which could run under our compiler. As the limitations are well described, it was usually easy to find acceptable ways of expression. But one drawback was that if the students were told that some feature of some construct was not implemented, they too often concluded that the whole construct was not implemented (e.g. even if dynamic bounds for arrays were not implemented in variant records, variant records were usable). Students were also more inclined to blame the partial compiler than really look at their syntactically incorrect programs.

However in retrospect, we are convinced that using even a partial compiler for a good tool is much better than having a full compiler for a partial tool, in our case Pascal that lacks advanced features like packages, exceptions, dynamic arrays, etc...

A more global criticism of Ada arises from our previous use of SIMULA-67 [4] in this course; Ada has a much more static approach to object oriented programming [3] so in some situations it can lead to more obscure or less natural solutions. For example, generic instantiation is done at compilation time instead of having dynamic objects created at run-time. Coroutines which were taught in the previous years were not even alluded to this year. We have found Ada to be a very cumbersome vehicle to express hierarchies of data types. Inheritance of properties and procedures from existing data structures is a fundamental property of object oriented programming that is easy to teach with SIMULA-67 but is more convoluted in Ada (especially as we lacked generics...).

Even with these caveats, we are convinced that Ada is a wonderful teaching and programming vehicle that covers a broad spectrum of advanced software engineering techniques. We intend to continue our efforts next year.

References

[1] ACM Curriculum Committee on Computer Science, Curriculum '78: Recommendations for the undergraduate program in computer science, CACM 22, 3 (March 1978), p. 147-166.

[2] Agrawal, J.C., Harriger, A.R.: Undergraduate Courses Needed in Ada and Software Engineering. Sixteenth SIGCSE Technical Symposium on Computer Science Education, New Orleans, March 1985, SIGCSE Bulletin, Vol. 17, no. 1, p. 266-281.

[3] Buzzard, G.D., Madge, T.N.: Object Based Computing and the Ada Language, Computer, Vol. 18, no. 3, p. 11-19, 1985.

[4] Dahl, O.-J., Myhrang, B., Nygaard, K.: Simula-67 Common Base. Report 725, Norwegean Computing Center, 1982.

[5] Evans, H., Patterson, W.: Implementing Ada as the Primary Programming Language. Sixteenth SIGCSE Technical Symposium on Computer Science Education, New Orleans, March 1985, SIGCSE Bulletin, Vol. 17, no. 1, p. 255-265.

[6] Rudd, D.: Software Review of the Telesoft Ada Version 1.3. IEEE Software, p. 99-100, May 1985.

[7] Telesoft Ada Compiler User's Manual. Telesoft, San Diego, May 1983.