The Case for Ada at the USAF Academy

Ricky E. Sward, Martin C. Carlisle, Barry S. Fagin, and David S. Gibson
Department of Computer Science
2354 Fairchild Dr, Suite 6G101
USAF Academy, CO 80840
1-719-333-3590
{ricky.sward, martin.carlisle, barry.fagin, david.gibson}@usafa.af.mil

ABSTRACT
This paper describes our experience with selecting Ada as the primary programming language for Computer Science and Computer Engineering majors at the USAF Academy. We have decided to teach Ada in the first three courses of these majors for the next few years. Our criteria for selecting Ada are based on features of the language (such as strong typing, lack of single-character errors, and case insensitivity), features of the compiler, (such as error messages and warnings), and features of the overall development experience (such as development environments, availability of textbooks, GUI development support, and industry acceptance). We compared Ada with Java, C++, and C#.

Categories and Subject Descriptors
D.3.2 [**Programming Languages**]: Language Classifications.
D.3.3 [**Programming Languages**]: Language Constructs and Features.

General Terms: Management, Languages.

Keywords: Computer Science Education, Ada 95, Java, C++, C#.

1. INTRODUCTION
The Department of Computer Science at the USAF Academy will continue using Ada 95 as the primary programming language for Computer Science and Computer Engineering Majors. There are several reasons for using Ada as our primary language. This paper will explain how Ada's language features, compiler features, and development environment help our students succeed. The paper will also discuss why we did not select Java, C++ and C#.

2. BACKGROUND
The USAF Academy is an undergraduate institution consisting of approximately 4000 students. Each student is required to take 30 core courses as part of their degree and has the option to major in any of 32 academic programs. The average course loads for students is six courses per semester with 42- lesson semesters spanning 18 weeks. As part of the core curriculum, each cadet takes one Computer Science course, Introduction to Computing, during their first year at the Academy.

Since 1996, we have used Ada as the programming language in the core Computer Science course [6]. Other departments at the Academy also chose to use Ada in their courses since we taught it in the Computer Science core course [7]. In 2001, we changed the core course to emphasize Information Technology more than computer programming. We continued to use Ada for the portion of the course that emphasized problem solving with computer programs. This year will be the first year Ada is not taught in the core course. We are now using a visual programming tool developed at USAFA that executes flow charts. This tool is described in [9].

Ada continues to be the primary programming language in our Computer Science and Computer Engineering majors. It is used in the first three Computer Science courses in these majors. This satisfies the depth of programming language experience requirement needed for ABET accreditation. Every few years, we re-examine the choice of Ada as our primary language, since new programming languages and development environments continue to emerge.

3. CRITERIA
Our decision to continue with Ada was made after considering several aspects of different programming languages and development environments. Overall, our decision was based on language features, such as strong typing and case sensitivity, compiler features, such as error messages and warnings, and the development experience, including such things as development environments, textbooks, and industry acceptance.

3.1 Language Features
There are several language features of Ada that make it the right choice for our primary language. Ada's *strong typing* is extremely beneficial to both the non-major students taking the core course and our Computer Science and Computer Engineering majors in their courses. With strong typing, programming errors are most often found at compile time instead of at run time. This means our students struggle with getting a program to compile, but once they succeed it is far more likely to do what is expected. In weakly typed languages, getting a program to compile is straightforward, but students will struggle with run time errors. These can be much harder to find and correct.

Another important language feature of Ada is *case insensitivity*. This helps our students be more productive. An error caused by case sensitivity is often hard to detect, especially for novice programmers. Although this may appear to be a small factor, Ada's case insensitivity gives our students a noticeable advantage. They do not spend time struggling with these simple errors but can spend time on the important programming issues we want them to learn.

Ada's *lack of single character errors* also gives our students an advantage. For example, in C-based languages the equality operator is ==. Since assignment in C-based languages can typically be a

SIGAda'03, December 7–11, 2003, San Diego, California, USA.
ACM 1-58113-476-2/03/0012.

statement as well as an expression, using an assignment statement in a conditional expression causes a single-character error that is very difficult for a novice programmer to detect. It is even harder for them to understand the semantics behind the cause of the error. Since assignment can not be an expression in Ada, this single-character error is avoided. Again a possibly small factor in the programming language helps our students avoid frustrating, difficult errors and allows them to be more productive in an extremely demanding academic environment. Brosgol [2] agrees that Ada is superior to Java because Ada avoids such confusing syntax. Agarwal [1] points out that C++ has deficiencies in its syntax that novice programmers may struggle with.

Ada's support for *subtypes* and *enumeration types* is far superior to Java, C++, or C#. The enumeration types in Ada can be used as indices in arrays, unlike Java, C++, and C#. This gives our students a powerful programming tool not available in the other languages. Ada also has built-in language support for *multi-tasking*, which has been used widely in our networks and operating systems courses.

As pointed out by Humphries [9], Ada includes *mixed-language pragmas*, which allow for easy interface with other programming languages. The A# language, Ada ported to .NET, is a fully interoperable language with the other languages in the .NET environment [5].

Finally, Ada's support for both imperative and Object-Oriented (OO) paradigms is essential for teaching introductory programming. Java requires students who are new to programming to learn the basics of OO syntax at the same time as they are learning the basics of computing. Students are not well-equipped intellectually to learn OO until they first understand control flow, variable assignment, and modularization. Virtually every paper in the computer science education literature over the past few years, even those friendly to Java, recognize this as a problem. (See for example [10] and [8]).

3.2 Compiler Features

The compilers available for Ada, with their *compile time error messages*, are very useful for teaching novice programmers. The error messages are most often accurate and steer the students in the right direction to solve their syntax error. Errors at run time are not as helpful to the novice programmer. With strong typing and descriptive compiler error messages, our students are able to be more productive and solve more problems on their own.

For example, suppose we have an enumeration type called Colors and we are using it in a case statement as shown in Figure 1. Ada requires that all members of the enumeration type are included in the when clauses of a case statement, or at least covered by an "others" clause. If we change the Colors enumeration type to now include the color Blue, we need to change all the case statements that refer to Colors. Compilers for languages that do not force explicit enumeration of case values will require users to identify all sites for necessary changes manually, increasing the possibility of logical errors.

By contrast, Ada compilers check this requirement and flag each of the case statements that must be updated. With a reasonable development environment, the job of updating all the affected case statements is trivial. Similarly, if we are using Colors as an index of an array, the Ada compiler will flag the initialization of the array and require that we include all members of the enumeration type in the initialization. These two particular compiler checks are not only good programming practice, but help

```
with Ada.Text_Io;
use Ada.Text_Io;
procedure Color_Prog is
 type Colors is
   (Red,
   White);
 C : Colors := White;
begin
 case C is
  when Red =>
   Put("Red");
  when White =>
   Put("White");
 end case;
end Color_Prog;
```

Figure 1 – The Colors Enumeration Example

our students avoid run time errors in their programs. Most importantly, they encourage students to think rigorously about the implications of changes they make to computer programs.

3.3 Development Experience

As we considered different programming languages to use, we evaluated newer languages such as Java and C#, the new language in Microsoft's .NET environment. We evaluated these languages based on industry acceptance, the ability to write both imperative and object-oriented code, the GUI development tools available, textbooks available, and the development environment.

Both Java and C# are much more widely *accepted by industry* than Ada. At the Academy, however, we are not affected by industry trends as much as a civilian institution might be because our students go directly into the Air Force after graduation. Our graduates will work on projects where they manage programmers, but will most likely not be programming when they serve in the Air Force. That is why we can focus on the education of computer scientists, and not the training of programmers.

We mentioned previously the importance of languages that support both imperative and object-oriented programming styles. We decided against Java as a primary language choice. With the object-oriented syntax required to write even a simple program, it is not practical to write imperative code in Java and then transition the students to an object-oriented style. However, it is practical to write imperative code segments in C#, so we examined C# in more detail.

The Visual Studio .NET *development environment* provided for C# programming is a very powerful tool. The touch and feel of the development environment including the "intelli-sense" auto-completion feature supports easy, rapid development of code. However, we believe that it is too complicated for the novice programmer to use effectively. There are many features of the environment that, while ideally suited for a professional Windows application developer, are bound to confuse a novice programmer.

By contrast, AdaGIDE [4] is an excellent tool that is easily understood by the novice programmer, yet is powerful enough for projects being done in the senior year of our majors. We also believed that if we went to Visual Studio, we would lose control of

the environment. Going from a product developed in-house to an off-the-shelf development environment built by Microsoft takes away a great deal of flexibility that, historically, has been of great benefit to us. Changes we desire in AdaGIDE are quickly integrated and fielded to our students.

We also examined the *GUI development tools* available for C# and Ada. The GUI builder in Visual Studio .NET is a powerful tool that easily allows the user to build sophisticated, professional-grade GUI applications using C#. When compared to the RAPID GUI development tool available for Ada [3], C# is a much more powerful, user-friendly tool. This is not surprising, since Visual Studio .NET is a commercial product with a multi-million dollar R&D effort behind it. However, a sophisticated development environment and GUI builder are not crucial components of introductory Computer Science courses. AdaGIDE and RAPID are sufficient for the types of projects we have our majors complete, making the superior development environments available for other languages less of a concern.

We also noticed a lack of *textbooks* available to use in our Data Structures (CS-2) course. To our knowledge, there are no textbooks available for a CS-2 course written for C#. This was a major drawback of C# and was enough to dissuade us from using C# as our primary language. The textbooks available for Ada are acceptable for our purposes.

4. CONCLUSION

The primary objections to Ada at any level, including in the classroom, are not technical but sociological, primarily concerning its lack of widespread acceptance. These are valid concerns, if taken in the appropriate context.

An analogy from natural languages may be appropriate here. No one would suggest that people stop speaking English and be made to learn Esperanto. Esperanto has simpler grammar, phonetic spelling, and by any set of technical linguistic criteria is simply a "better" language. Nonetheless, the costs of retraining and the efficiency problems with switching are too great: we live with the imperfections of English because linguistic issues aren't the only ones to consider.

Similarly, those who believe that university computer science departments serve as training grounds for industry programmers, or for whom preparation for industry is a significant factor in curriculum decisions, need to take non-technical factors into account when choosing a programming language. One author (Fagin) taught computer science in Russia for a semester. His students all were studying English as part of their computer science curriculum, because that is the rest of the world uses. Complaints to the Dean about the large number of irregular verbs in English and its atrocious spelling rules would not have progressed very far.

But education is different. If we are interested in preparing young minds for the intellectual discipline of computer science, then we have the luxury, indeed the obligation, to start from scratch and pick the best tool for that task. Currently, that tool is Ada.

We must always be open to the possibility that this may not always be the case. Despite our focus on the differences in this paper, it is clear that mainstream programming languages are becoming more and more alike. More and more features best embodied in Ada (strong typing, generics, support for vital software engineering principles) are making their way into C# and Java. That's why we'll continue to examine the question of programming language choice for our majors.

However, it is clear that, at least for now, Ada remains the right choice for introducing students to the intellectual discipline of computer science at the Air Force Academy.

5. REFERENCES

[1] Agarwal, A. and Agarwal, Krishna. Some Deficiencies of C++ Teaching CS1 and CS2. *ACM SIGPlan Notices*, v.38, June 2003.

[2] Brosgol, Benjamin M. A Comparison of Ada and Java as a Foundation Teaching Language, AdaCore Technologies, from http://www.act-europe.fr/texts/papers/ada-java-teaching-comp.pdf

[3] Carlisle, M.C. A Truly Implementation Independent GUI Design Tool. *Proceedings of SIGAda '99,* Redondo Beach CA, October 1999. Also appears in *Ada Letters*, 19(3): 47-52, September 1999.

[4] Carlisle, M.C. and A. T. Chamillard, AdaGIDE: A Friendly Introductory Programming Environment for a Freshman Computer Science Course, *Proceedings of ASEET '97*, Monmouth NJ, June 1997. Also appears in *Ada Letters*, vol. 18, no. 2 (March 1998), pp. 42-52.

[5] Carlisle, M.C., Sward, Ricky E. and Humphries, Jeffrey W. Weaving Ada 95 into the .NET Environment, *Proceedings of SIGAda 2002*, December 8-12, 2002, Houston, TX.

[6] Chamillard, A.T. and Hobart, William C. Transitioning to Ada in an Introductory Course for Non-Majors. In *Proceedings of TRI-Ada '97*, St Louis, Missouri, November 1997, pp. 37-40.

[7] Chamillard, A.T., Lisowski, Ronald J., and Young, Richard R. Using Ada in Non-CS Majors. In *Proceedings of the ACM SIGAda Annual International Conference (SIGAda '98)*, Washington, DC, November 1998, pp. 61-67.

[8] Hristova, M. et al, Identifying and Correcting Java Programming Errors for Introductory Computer Science Students. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education,* Reno, Nevada, February 2003, pp 153-156.

[9] Humphries, J. W., Carlisle, Martin C., Wilson, Terry A. Multilanguage Programming with Ada in the .NET Environment. In *Proceedings of the ACM SIGAda Annual International Conference (SIGAda '03)*, San Diego, CA, December 2003.

[10] Reges, S. Conservatively Radical Java in CS1. In *Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education, Austin, TX, March 2000, pp 85-89.*