

Programming Language Impacts on Learning

J. A. "Drew" Hamilton, Jr., Ph.D.
Joint Forces Program Office
Space & Naval Warfare Systems
Command
San Diego, CA 92110-3127
drew@drew-hamilton.com

Jeanne L. Murtagh
Software Professional
Development Program
Air Force Institute of Technology
WPAFB, OH 45433
Murtagh-JL@acm.org

Richard G. Zoller
Dept. of Electrical Engineering &
Computer Science
United States Military Academy
West Point, NY 10996
Richard-Zoller@usma.edu

1. ABSTRACT

What programming language should be taught in an introductory computer science course? Studies and experiences at West Point provide some insight into these questions. This paper reports the statistical results obtained from a side-by-side comparison of Pascal and Ada. The unique nature of our institution provided the opportunity to isolate and study causal factors, and allowed us to follow the experimental and control groups through graduation.

We conducted a side by side comparison of the use of Pascal, which was specifically designed for instructional purposes, and Ada in an introductory course. The experiment demonstrated that students were much more successful in Ada than in Pascal, and led to the revision of the Academy core curriculum to use Ada 95 in our introductory computer science class. This paper discusses the details of our comparison, citing specific examples to illustrate a rational basis for evaluating programming language features.

2. INTRODUCTION

The general educational goal of the United States Military Academy at West Point is to "enable its graduates to anticipate and to respond effectively to the uncertainties of a changing technological ... world." In this environment there is little disagreement over the need for the institution's graduates to understand the fundamentals of computer science. Every student who attends West Point is required to complete CS 105, Introduction to Computers and Programming. CS 105 is a CS 0 course by ACM standards. The course is one semester long, with thirty-four lessons of fifty-five minutes and six two-hour labs. CS 105 consists of three phases: ten lessons on computer concepts, twenty-three lessons on problem-solving through programming, and seven lessons on the use of office automation applications. It is important to clearly understand the goals of the second phase of the class. We focus on engineering problem solving, to help the students develop the skills to synthesize complex information and formulate concrete, workable solutions to abstract problems. CS 105 provides problem-solving experience through the use of a 3rd generation programming language (3GL). We chose a 3GL because we believe excessive use of automated tools for introductory-level projects would substitute "point and click" code generation for the design effort necessary to gain hands-on problem-solving experience. Our emphasis is problem solving, not rapid application development. It is also important for these students to understand not only a computer's capabilities, but also its limitations. These capabilities, and *particularly the limitations*, are most effectively demonstrated using a 3GL.

In a recent survey of five incoming faculty members, each was asked to name their desired characteristics of a 3GL supporting the objectives of CS105 [7]. Parenthetical numbers indicate when more than one respondent mentioned a characteristic.

- Reliability (4)
- Low cost (4)
- Strongly typed (3)
- Standards-based/Platform-independent (2)
- Basic concepts/constructs are simple (2)
- Emphasis on compile-time (versus run-time) error determination (2)
- Good debugging support (2)
- Block structured
- Provides exception handling
- Security
- Regularity

- Fully-developed professional strength
- Supports engineering design methodology
- Shallow (relatively speaking) learning curve
- Applicable to real-world problems
- Supports Object Oriented programming

These and other factors were considered in determining a language(s) to evaluate as a replacement for Pascal. The Reid Report, compiled and updated by Dr. R. J. Reid of Michigan State University [8], indicates that the primary programming languages in academic use today are, in alphabetical order, Ada, C/C++, Pascal, Modula and Scheme.

The difficulties of using C and C++ for educational purposes are well documented. The confusing syntax is detailed in [6], and some of the issues with C++ are discussed in [1]. Even with the recent successful balloting of the C++ standard, popular C++ compilers come with many proprietary features. Scheme is not used anywhere at West Point and there was little or no faculty expertise in Scheme, whatever its merits. At the time of the study, Java was not well known and the procurement of a large number of Java compilers was considered imprudent. The then recent ISO standardization of Ada 95, the dropping of the U.S. Department of Defense trademark and the successful reports of Ada 95 early adopters led us to choose Ada for our test.

3. CONDUCT AND RESULTS OF THE EXPERIMENT

We conducted a “side by side” comparison of the impact of Pascal and Ada on the success of our CS 105 students. An instructor proficient in both Pascal and Ada was selected. Then we selected one “average” CS 105 section as the experimental group, which would use the Ada programming language. The two Pascal sections taught by the experimental group’s instructor formed the control group. We also evaluated the performance of this control group against the Pascal sections taught by other instructors, to ensure that any performance differences could be attributed to the languages and not the instructor.

Every effort was made to minimize differences between the experimental and control groups. The students and lesson content were very similar in both groups. The section identified as the experimental group was “average.” Students were not specially selected for this section. These students had “average” GPAs which was consistent with their SAT scores as detailed in Tables 1 and 2. A few students had some previous programming experience, but most did not. The same concepts and programming constructs were covered, lesson-by-lesson, with each group. For example, the second programming lesson focused on basic input and output for the keyboard and monitor. The control group learned about read, readln, write and writeln, while the experimental group learned about get, get_line, put and put_line.

Based on an analysis of SAT scores and grade point averages of the control and experimental group, we conclude that the two groups were comparable to each other. Summarized data for comparing SAT math and verbal scores for the experimental and control groups is presented in tables 1 and 2. In a straightforward application of a t-test we concluded that the math and verbal mean scores were not provably different between the two groups.

	Mean	Median	Variance	Std Dev
Pascal	554.35	555	5077	71.25
Ada	530	520	2650	51.48

Table 1. SAT Verbal mean scores.

	Mean	Median	Variance	Std Dev
Pascal	641.24	630	4644	68.14
Ada	641.76	640	4152	64.44

Table 2. SAT Math mean scores.

Further evidence to support the claim that both the experimental and control groups were academically similar results from the application of a t-test to the final grade point averages of the students as shown in table 3.

	Mean	Median	Variance	Std Dev
Pascal	2.78	2.772	.215	.464
Ada	2.88	2.862	.140	.375

Table 3. Final grade point averages.

Thus we concluded that there is no statistical evidence to contraindicate the claim that student populations in both the control (Pascal) and experimental (Ada) groups had similar academic success indicators in terms of grade point averages and SAT scores.

Our research hypothesis was that the programming scores of the experimental group using Ada would be higher than the scores of the control group using Pascal. It was important for us to have sound evidence that this hypothesis was true. Although Ada has been used successfully in many computer science programs, so have many other programming languages. Most of the success stories are anecdotal in nature. The rigorously structured USMA curriculum and the large number of students (currently more than 30 sections of CS 105, with 18 students per section, each semester) led to the decision to conduct an experiment as a pilot study.

The results of the experiment were instructive. There were three full percentage points difference in overall course performance between the control group and the experimental group. A statistical analysis of forty-six sections of CS 105 in the 1995 academic year indicates that a difference of three percentage points between sections taught by the same instructor would be significant. Consequently, we concluded that students learning and using basic programming concepts in Ada would perform better than those learning those same concepts in Pascal.

The "overall" average represents the grades earned in the entire course (i.e., the average across all three phases -- computer concepts, problem-solving through programming, and applications software). The grades for Labs 1, 2, and 3 represent the major graded events from only the programming phase of the course. The graded labs consist of special two-hour class periods. Students are given a problem and are expected to use an engineering problem solving process to design and implement a solution. Note that the difference in performance between the two groups during the programming phase is greater than the overall difference. This occurred because the control group received better grades than the experimental group in the first phase of the course.

	Overall	Lab 1	Lab 2	Lab 3
Control (Pascal) Group	85.08	82.83	87.28	80.60
Experimental (Ada) Group	88.45	87.96	91.14	83.96

Table 4. Performance of control and experimental groups.

4. ANALYSIS

We studied the impacts that a programming language made on achieving our course objectives. We determined that programming language choice affected students' understanding of software engineering constructs and their ability to grasp how real-world objects are represented in a computer. We also observed the impacts of syntax and development environments on novice programmer outcomes. We are confident that the results below can be attributed to differences in the two programming languages.

4.1 Support for Software Engineering

Ada supports the early introduction of software engineering concepts [5]. We found that use of library units communicated the concepts of encapsulation, abstraction and reuse very effectively to the experimental group. This

was apparent as students wrote their first program using a simple output statement. Similar learning outcomes were not achieved in the control group. Consider the two programs below:

```
WITH Ada.Text_IO;
PROCEDURE hello is
BEGIN
    Ada.Text_IO.put_line("hello, world");
END hello;

PROGRAM hello;
BEGIN
    writeln("hello, world");
END .
```

The two programs are of approximately the same level of coding difficulty. Reading a package specification and elementary use of a library unit were not difficult for our students. Yet the Ada version better supports learning as it forces the student to address abstraction, encapsulation and reuse at an appropriate introductory level.

4.2 How Computers Work

The first course goal of CS 105 is to become familiar with the fundamental concepts of computer science. While it is desirable to program at a high level of abstraction, it is also important for the students to understand what is “going on under the hood.” We observed that students in the Ada section did markedly better on their final examinations overall, but particularly in the area of computer science fundamentals. The Ada students seemed to have a much easier time understanding the differences among types, variables of those types and the real world entities they represent.

For novice programmers, implicit type conversions (as provided in Pascal) hide the fact that different data types are represented differently inside the computer. In contrast, the explicit type conversions required in Ada reinforce this concept at the cost of only a few keystrokes.

It is also important for beginning students to know something about the internals of the machines they are working on, and programming in Ada helps students appreciate this. Consider the difference between the mathematical concept of an infinite set of integers and the constraints imposed on the finite set of integers which can be represented by a machine architecture.

In Ada, a constraint error is raised when a programmer attempts to assign a value greater than 32,767 to a 16-bit integer. This makes it easy for novice programmers to recognize that an error has occurred, and to understand, at least at a conceptual level, what caused the error.

In Pascal, no error message is provided; the program simply produces the wrong result. If this addition is embedded in intermediate calculations, it might not even occur to a novice programmer to evaluate the addition as a source of the error. "How could it be possible that the computer cannot simply add two integers together? I don't even need to check that!" thinks the student who has not yet learned to differentiate between mathematical concepts and the *implementation* of these concepts on the computer.

4.3 Error Detection and Correction

Ada can make it easier to detect and correct errors. These features were evident at both compilation time and run-time. Strong typing and concomitant type checking produce a compilation error if the problem can be detected at compile time. Otherwise, an exception is raised at run-time. The numeric overflow discussed above raises the exception, `constraint_error`.

Strong typing in Ada forces students to pay more attention to the significance of data types. As previously noted, this supports student understanding of data representation issues. This also makes it easier for students to detect problems early. Weak data typing can be confusing even for more advanced programming students, as well as for practitioners.

The power of the Ada exception mechanism manifests itself even in an introductory course. Our novices were able to reference their unhandled exceptions in the Ada Language Reference Manual and understand *why* their programs were failing. This gave them a significant advantage over the Pascal sections during debugging. We contend that the activities required to detect, isolate, and remove “the last few bugs” are critical to the problem-solving process.

4.4 Syntax Issues

Small -- but significant -- differences exist between the syntax of Pascal and Ada:

- Semicolon usage
- Inclusion of a block of statements in structured programming constructs
- Treatment of formal parameters

These differences had a critical effect on students’ success.

Semicolons are treated differently in Ada and Pascal. In Ada, semicolons are always placed at the end of every statement. In Pascal, a semicolon is placed at the end of a statement *unless* that statement is embedded in a special position in another statement. For example, the Pascal statement “`x := y * 5.0;`” would normally be followed by a semicolon. However, if that same statement is located as the THEN clause in an IF-THEN-ELSE statement, then it must *not* be followed by a semicolon:

For example:

```
if (x < y) then
  x := y * 5.0
  {note: semicolon forbidden;
   still inside if statement}

else
  x := y * 2.0;
  {note: semicolon required;
   identifies the end of BOTH the if &
   assignment statements}
```

This seemed inconsistent to the students in the control group, who often inadvertently ended an `if` statement before its `else` clause with one misplaced semicolon. This problem was exacerbated by the insufficiently specific error message (“Error 113: Error in Statement”) provided by the Turbo Pascal compiler under these circumstances.

Students in the experimental group also benefited from Ada’s clear syntax for identification of a block of statements to be included inside a structured programming construct (e.g., IF, CASE, WHILE, FOR). In Ada, reserved words delimit the different parts of a structured programming construct. For example, here is the Ada syntax for a while loop:

```
-- reserved words shown in upper case

WHILE boolean_condition LOOP
    statement;
    .
    .
    .
    statement;
END LOOP;
```

Students may include as many statements as needed between “loop” and “end loop.” The compiler provides a useful (i.e., suitably specific) error message if a student omits one of the required reserved words that serve as delimiters for the while loop.

In Pascal, the syntax for a while loop is as follows:

```
while ( boolean_condition ) do
    statement;
```

If a student needs to include more than one statement in the while loop, he or she must construct a compound statement by “bracketing” the statements to be accomplished inside the loop with a `begin` and `end`. If the student omits the `begin` and `end`, the code will still compile – but only the first statement will actually be executed inside the loop. All remaining statements will be executed exactly once, after the loop has terminated. This tends to produce output that is very different from what the programmer intended, and it can be a very difficult error for novice programmers to detect.

The use of compound statements in Pascal, compared with delimitation using reserved words in Ada, is also an issue with selection constructs (if, case). Omission of the `begin/end` pair for selection constructs causes compile-time, rather than run-time, errors. While these errors are easier to detect and correct than the run-time errors that occur with repetition constructs (while, for), they still prevented many of our control group students from completing quizzes and graded labs. This was due, in part, to the cryptic “Error 113: Error in Statement” message provided by the Turbo Pascal compiler, and we admit that a better error message might have helped the control group find this error more quickly for selection constructs. We note, however, that even a better compile-time error message would not resolve the difficulties generated when the `begin/end` pair is omitted in repetition constructs.

We observed significant differences between the abilities of the experimental and control groups to pass data between procedures using parameters, and we attribute these differences to language syntax.

In Ada, the rules governing formal parameters, including the specification of their parameter passing mode and their use inside procedures, are very clear and are enforced by the compiler. Consider the following Ada procedure specification:

```
procedure DoSomething (par1: in integer;
                      par2: in out character;
                      par3: integer);
```

Par1 is explicitly identified as an “in” mode parameter, meaning that it is intended to be read but not changed. Par3 is also an “in” mode parameter, because its mode is not explicitly specified and “in” is the default. Any attempt to write to an “in” mode parameter will cause a compile-time error. Therefore, if a student simply forgets to specify par3’s parameter passing mode and subsequently tries to change that parameter’s value inside the procedure, the compiler will notify the student of this error. Students in the experimental group readily understood this concept, and were able to quickly correct any errors in their parameter passing modes.

In Pascal, parameter modes are much more confusing to novice programmers. A parameter’s value can always be changed inside a procedure. However, that value is not passed back out of the procedure unless the parameter was declared as a “var” parameter – and novice programmers easily misunderstand the syntax required to do this. Failure to include the “var” (either because it was omitted completely, or because a student did not understand that the “var” must be repeated after each semicolon which appears in the parameter list) can lead to erroneous program behavior that is very hard to debug. Consider the following Pascal procedure header:

```
procedure DoSomething (par1: integer;
                      var par2:char; par3:integer);
```

The values of all three parameters may be changed inside procedure `DoSomething`. However, only the new value for `par2` will be returned to the calling routine. The new values of `par1` and `par3` will be discarded, *without any notification to the programmer*. We found this difference especially critical for a CS 0 course such as CS 105, in which students lack the depth of understanding needed to track down such subtle errors. This particular problem caused many difficulties for our CS 105 Pascal students.

4.5 Environment Comparison

A major concern at the outset of the experiment was that, regardless of the merits of Ada, the available environments might be too hard for novice programmers to use. If novice students are unable to successfully work in the environment, the merits of the language are immaterial. Our students had no difficulty working in the then available Ada environment. PC-based Ada environments now are even better. One environment of particular interest, AdaGIDE, was developed at the United States Air Force Academy [3].

5. CONCLUSIONS

We were pleased to see the very positive effect of the Ada programming language on engineering problem solving. Ada 95's simple, consistent syntax, an easy-to-use environment and intrinsic support for software engineering allowed students to concentrate on designing solutions, rather than arcane programming idiosyncrasies. As a direct result of this study, which demonstrated that our students could go farther, faster in Ada than they could in Pascal, the Academic Board of the U.S. Military Academy approved changing to Ada 95 as the programming language for all students in our introductory computer science class. We note that these results have been independently achieved and verified at the U.S. Air Force Academy [4]. Other programming language analyses conducted in upper division courses have produced similar results, showing the pedagogical advantages of Ada 95 [2].

Anecdotally, students reported that they found Ada much easier to use than Pascal. Currently, as students are exposed to other languages in operating systems, networking and artificial intelligence courses, the stated preference for Ada becomes more pronounced.

This experiment gave us an excellent opportunity to isolate the effect of a programming language on novice programmers. We submit that the programming language study conducted at West Point is a model approach for determining the comparative pedagogical advantages of programming languages and offers a sound basis for an unbiased decision.

6. EPILOGUE

The students in the study were members of the class of 1998. The study, which was conducted in 1995, while the members of the study groups were freshman, clearly supported the conclusions described above concerning the effects of programming language choice on students' ability to execute the problem solving process.

We reviewed the academic records of the study participants to look for additional insights. We evaluated their graduation rates, choice of minor, and choice of major.

Forty-seven of the fifty-three students involved in the study graduated. This graduation rate is consistent with the demographics of the USMA student population.

USMA requires each cadet to take a "core engineering sequence," which consists of five sequential courses in a technical area and is roughly equivalent to a minor at many other schools. Three members of the control group and two members of the experimental group completed the computer science engineering sequence. One member of the control group completed the electrical engineering sequence. A summary of the students' core engineering sequence choices appears in Table 5.

Core Engineering Seq.	Experimental (Ada)	Control (Pascal)
Civil Engineering	1	8
Computer Science	2	3
Electrical Engineering		1
Environmental Eng.		4
Mechanical Engineering	1	
Nuclear Engineering	1	
Systems Engineering	11	15

Table 5. Core engineering sequence selections.

No members of either the experimental group or the control group majored in computer science or electrical engineering. This is not unexpected because there were only 26 computer science majors and 15 electrical engineering majors among the 863 graduates from the class of 1998.

Therefore, based on our experiment, we cannot draw any conclusions on the effect of programming language beyond relative success or failure in the introductory programming course.

7. REFERENCES

- [1] Ben-Ari, M. and Henney, K., "A Critique of the Advanced Placement C++ Subset," *SIGCSE Bulletin*, Vol. 29, No. 2, September 1991, pp. 7-10.
- [2] Blair, J.R.S., Ressler, E.K., Wagner, T.D., "The Undergraduate Capstone Software Design Experience," *Tri-Ada '97 Proceedings*, Nov 9 -13, 1997, St. Louis, Mo., pp. 41 - 47.
- [3] Carlisle, M. C., and Chamillard, A.T., "AdaGIDE: A Friendly Introductory Programming Environment for a Freshman Computer Science Course", 11th Ada Software Engineering Education Team Symposium, Monmouth University, Monmouth, N.J., 12 -13 Jun 97. Also appears in *Ada Letters*, 18(2):42-52, March 1998.
- [4] Chamillard, A.T., and Hobart, Jr., W.C., "Transitioning to Ada in an Introductory Course for Non-Majors," *Tri-Ada '97 Proceedings*, Nov 9 -13, 1997, St. Louis, Mo., pp. 37 - 40.
- [5] Feldman, M. and Koffman, E., *Ada 95: Problem Solving and Program Design*, Addison-Wesley, Reading, Mass. 1996.
- [6] Mody, R.P., "C in Education and Software Engineering," *SIGCSE Bulletin*, Vol. 23, No. 3, Sept. 1991, pp. 45-56.
- [7] Ray, C.K., Jr. "Incoming Faculty Survey," e-mail correspondence with the authors, 12 July 98.
- [8] Reid, Richard J., "First-Course Language for Computer Science Majors," Internet Survey, ftp.cps.msu.edu:pub/arch/CS1_Language_List.Z.

AUTHORS

John A. (Drew) Hamilton, Jr., Ph.D., Lieutenant Colonel, U.S. Army, is the Director of the Joint Forces Program Office at the Space and Naval Warfare Systems Command. His previous assignments included tours as the Research Director and an Assistant Professor in the Department of Electrical Engineering and Computer Science at the United States Military Academy, West Point, NY and as Chief of the Ada Joint Program Office. <http://www.drew-hamilton.com>

Jeanne L. Murtagh, Major, U.S. Air Force, is the Director of the Software Professional Development Program (SPDP), at the Air Force Institute of Technology, Wright-Patterson, AFB, OH. Major Murtagh previously served as an Assistant Professor at West Point, where she directed and implemented the results of the USMA language study.

Richard G. Zoller, Major, U.S. Army, was most recently an Assistant Professor in the Department of Electrical Engineering and Computer Science at the United States Military Academy, West Point, NY. Major Zoller has previously served as a signal company commander in the 366th Signal Battalion. He is currently serving in a signal battalion on Okinawa.