Empirical Software Engineering, 1, 45–59 (1996)

Comparing Ada and FORTRAN Lines of Code: Some Experimental Results

THOMAS P. FRAZIER
Institute for Defense Analyses, 1801 N. Beauregard St., Alexandria, VA 22311

JOHN W. BAILEY
Institute for Defense Analyses, 1801 N. Beauregard St., Alexandria, VA 22311

MELISSA L. CORSO
Institute for Defense Analyses, 1801 N. Beauregard St., Alexandria, VA 22311

Abstract. This paper presents the results of a study comparing pairs of functionally equivalent programs written in the FORTRAN and Ada languages. We found the Ada programs to require more lines of code than their functionally equivalent FORTRAN counterparts. However, we also observed that the overhead for Ada diminishes as program size increases. Our limited data suggested that there may be a cross-over point beyond which the size of an Ada program would be smaller than a functionally equivalent FORTRAN program. We explore some of the reasons for these economies of scale when using Ada. The implications of our findings on software cost estimating are also discussed.

Keywords: measurement theory, cost estimation, Ada, software size, size estimation

1. Introduction

A. Background

The introduction of Ada in the early 1980s has been cited as one of the major weapons in the fight to reduce the proliferation of computer languages and to control the cost of software in the DoD. The features of the Ada language were carefully chosen to enable good engineering practices and structure to be imposed during the development and maintenance of computer software. However, the use of these structural and engineering features presents new problems for the cost analysts responsible for estimating either the size or the cost of software systems to be developed in Ada.

Most software cost estimating models in use today assume that the cost of developing a computer program is a function of the size of the program plus other representations or measures of the complexity of the program, the skills and experience of the programmers, as well as other factors which affect cost. Typically, these cost models use lines of code as a representation of the size of a software program or project. For example, (Boehm, 1981) models the relationship between effort to develop software and the size of the code by using the following general form

$$E = \alpha(KLOC)^{\beta} \prod^{n} m_i, \tag{1}$$

where E is defined to be the staff months of development effort, $KLOC$ is defined as thousands of source lines of delivered code, α and β are the parameters that define the baseline relationship between effort and size, and m_i are multipliers or cost drivers that account for differences in software product attributes, computer attributes, personnel attributes and project attributes.

The use of a line of code as a unit of measure is appropriate and effective when dealing with line-oriented languages such as FORTRAN or assembly languages. However, several problems arise when applying a FORTRAN-specific or line-oriented cost model to software being developed in Ada.

First, instead of being line-oriented, Ada is block-oriented, which means its statements and declarations can span several lines or be nested within one another. This implies that, instead of simply counting carriage returns, a special Ada-specific way of counting the effective number of lines in an Ada program is needed. Further, even given a way of measuring the size of an Ada program by some method for line counting, there is no assurance that a line of Ada by this definition will capture the same amount of function as a line of FORTRAN. This means that two functionally equivalent programs in the two languages might be considerably different in size, as measured by lines of code. Finally, there is no assurance that the development cost for a line of Ada by this definition will be the same as the cost to develop a line of FORTRAN.

B. Objective

This project addresses the functional size issues but not the programming effort issues raised when comparing the sizes of Ada and FORTRAN programs. The later question can be addressed by observing the cost required to develop Ada programs of various sizes[1]. In theory, functional size issues could be handled by using function points to provide a language-independent measure of functionality (Jones, 1991). However, there are problems with using function points for this purpose. First, the ratio of function points to lines of code is assumed to be constant irrespective of the size of a program. We believe this may not be the case. Second, defining and counting function points is a matter of some debate and the focus of a great deal of current research. We believe our research may contribute understanding in both of these areas.

Information about the relative sizes of functionally equivalent programs is needed by any organization that is considering the use of Ada for application areas in which they have previous experience in FORTRAN. The reasoning is that such an organization would be able to estimate the size of a programming job if it were developed in FORTRAN. However, it would have no way of knowing whether an Ada solution would be more or fewer lines of code. What is needed is the added knowledge about how large an Ada solution to a problem will be, given an estimate of size for a FORTRAN solution. This knowledge will allow FORTRAN organizations to "bootstrap" their software cost estimating capabilities to include developments in the Ada language. Eventually, the need for this stop-gap technique will be eliminated by first-hand experience with Ada.

The focus of this study can be expressed in algebraic terms. The relationship between effort and size in line-oriented languages such as FORTRAN has been studied extensively

by software engineers and cost analysts (Keremer, 1991) and can be represented by equation (2), which is a FORTRAN-specific version of equation (1),

$$E_F = \alpha(KLOC_F)^{\beta} \prod_{i}^{n} m_i \tag{2}$$

(where the subscript F denotes FORTRAN). Similarly, researchers are learning about the relationship between effort and size in block-oriented languages such as Ada using an Ada-specific version of equation (1) as shown in equation (3),

$$E_A = \delta(KLOC_A)^{\gamma} \prod_{i}^{n} m_i. \tag{3}$$

Far less is known about the relationship between the size of an Ada program and the size of a functionally equivalent FORTRAN program, or

$$KLOC_A = f(KLOC_F). \tag{4}$$

We focus on FORTRAN because the impetus for this research stems from our work at the Institute for Defense Analyses for the Strategic Defense Initiative (SDI). Space systems have historically employed FORTRAN for both the ground segment software and the software embedded in the space craft or satellite itself. Cost estimating relationships using FORTRAN lines of code have been the rule. However the SDI plans to field space systems software that will be predominately written in Ada. By determining the differences in size between functionally equivalent FORTRAN and Ada programs, this study will further our understanding of how traditional FORTRAN cost and size estimating models will have to be adjusted to handle the Ada language. In addition, by understanding the differences in size between functionally equivalent FORTRAN and Ada programs, we can estimate the error incurred by cost analysts when they simply use Ada and FORTRAN lines of code counts interchangeably.

2. Approach

In order to compare the sizes of functionally equivalent Ada and FORTRAN programs, we devised a simple experimental procedure. The procedure involved taking standard programs and routines written in FORTRAN and rewriting them in the Ada language. First, we developed an Ada solution for each program using the features of Ada as appropriate, such as packages and user-defined types. Because programming style can affect program size, we also wrote both terse and verbose versions of each FORTRAN program and of each of the Ada programs. This yielded six functionally equivalent versions of each algorithm studied, three in FORTRAN and three in Ada. We then selected two established definitions for an Ada line of code and compared the number of Ada lines of code in these new programs to the number of lines in the original FORTRAN programs.

A. Test Programs

A total of four FORTRAN routines were used in the experiment. Three FORTRAN routines and their drivers were taken from *Numerical Recipes in FORTRAN* (Press, 1992). The fourth FORTRAN program was supplied by NASA/SEL (NASA, 1994) along with an Ada translation which we adapted for our basic Ada version of the program. Terse versions of the FORTRAN routines were devised by taking shortcuts such as allowing implicit declarations and eliminating certain unnecessary statements, such as format statements and continue statements. Verbose versions were devised by separately declaring variables, adding explicit format statements, and adding other optional statements to improve clarity. The terse versions of the Ada routines were devised by allowing multiple variables to appear in a single declaration and by using only positional parameter associations. The verbose versions were derived by separately declaring all variables and by using named parameter associations. By having a terse, normal, and verbose version of each algorithm in each language we were able to obtain a useful picture of how the range of possible program sizes for a given function would differ in the two languages.

The three routines selected from *Numerical Recipes* were:

- Moments of a Distribution—a statistical routine that computes the moments (e.g., mean, variance, kurtosis) of a given distribution.
- Quicksort—a sorting routine that uses a "partition-exchange" sorting method.
- Fast Fourier Transform (FFT)—a computational algorithm that relates physical processes defined either in the time domain or frequency domain.

A fourth routine, an orbit propagator provided by the NASA/SEL, computes the orbital position of an earth satellite. These four were selected because they cover a range of computational applications likely to be used in space systems, and the algorithms involved are well known and widely used.

B. Ada and FORTRAN Formatting and Style

For comparisons we used two methods to measure the size of each of the Ada and the FORTRAN subprograms and their drivers. Method 1 requires the adoption of a specific style for the formatting of the code and then simply counts the number of non-comment, non-blank lines in the file containing the code. The most complete definition we found for Ada formatting and style is *Ada Quality and Style: Guidelines for Professional Programmers*, published by the Software Productivity Consortium (SPC, 1992). Except when deliberately employing either a verbose or a terse format, we have adopted these rules of style for the examples of Ada used in this report. For the style of the FORTRAN examples we followed the conventions detailed in *American National Standard Programming Language FORTRAN* (ANSI, 1978). This standard was adopted by the DoD in 1978.

Method 2 is a count of the number of source statements which appear in the code. Because this method measures the number of logical statements it is not sensitive to the number of

physical lines a statement occupies. It is therefore not sensitive to formatting, comments or blank lines. Because of the multiple declaration option in Ada, this method is still somewhat sensitive to programming style, however. The specific declaration and statement counting rules we followed for both methods are described in *Code Counting Rules and Category Definition/Relationships* (SPC, 1991). Consistent with the terms used in that report, we call Method 1 the physical source statement count, or the PSS count, and Method 2 the logical source statement count, or the LSS count[2].

Although the SPC report also discusses how to count comments in each language, we have chosen for this study to ignore all comments and blank lines when measuring the sizes of our examples. Also, in this study we adopt the definition for a source statement to mean any programming instruction. In other words, all Ada declarations, statements, and pragmas are counted as source statements. In FORTRAN a source statement can be an executable statement, a data declaration, or a compiler directive.

C. Code-Counting Conventions: Sizing Issues

The two selected approaches to measuring program size are each compromises between the amount of the information captured by a size measure and the complexity of taking the measurement. The PSS method requires the program to be first formatted according to a set of rules and then simply counts the number of carriage returns in the code, excluding blank lines and comments. This approach either requires that a particular style be followed by the code developers or that a formatter (or "pretty printer") be used before any line counting is done. The LSS method defines a method for counting syntactic units rather than counting lines at all, so that the formatting of the code is immaterial. This general approach can be useful when reporting size outside of an individual development organization where styles and formatting rules may differ. However, it requires the additional complexity of processing or parsing the code in order to obtain the size count automatically.

Each statement, declaration, or pragma in Ada terminates with a semicolon (";"). Semicolons are also used to separate formal subprogram and entry parameters. To compute the LSS, count the semicolons except when they appear in (1) comments, (2) character literals, and (3) string literals. We decided to count the semicolons in formal parameter lists since formal parameters are, in effect, declarations. Although this count always misses the last parameter, we felt that correcting for this small effect was not worth the added complexity. A logical source statement in FORTRAN can be computed by counting only those lines which have the blank character in column 1 and either a blank or a zero in column 6. This follows from the convention that comments in FORTRAN are identified as those lines with the character "C" or "*" in column 1, while continuation lines have any character except a blank or a zero in column 6. This rule, therefore, counts only non-comment, non-continuation lines. In structured FORTRAN (such as that used in our examples) the statement "end if" is not counted as a logical source statement but it is included in the count of physical source statements.

The difference between the PSS and LSS methods and how they apply to counting code in Ada and FORTRAN can best be illustrated using a simple example. Table 1 shows a portion of the Ada and FORTRAN code found in the Fourier analysis subroutine.

Table 1. Comparison of Ada and FORTRAN "if ... then" statements.

Ada	LSS	FORTRAN	LSS
if J > I then		if (j.gt.i) then	√
Temp:= Data (J);	√	tempr=data(j)	√
Data (J):= Data(I);	√	tempi=data(j+1)	√
Data (I):= Temp;	√	data(j)=data(i)	√
end if;	√	data(j+1)=data(i+1)	√
		data(i)=tempr	√
		data(i+1)=tempi	√
		end if	

The portion of the subroutine is an "if... then" statement written in the styles according to the references noted above. (Capitalization is not significant in either language. The lower case convention used in the FORTRAN example is adopted from (Press, 1992)). The Ada PSS count is five and the FORTRAN PSS count is eight. The Ada LSS count is four and the FORTRAN LSS count is seven. There are four semicolons in the Ada code. The "end if" in the FORTRAN code is not counted as a logical statement since it is required by and part of the "if" statement.

3. Results

In this section we examine the results of applying the code and style conventions discussed in the preceding chapter to the four test programs. We examine some of the differences between the Ada and FORTRAN that might explain the results and we discuss the notion that the Ada language exhibits scale economies (i.e., as the size of the program grows the number of Ada lines grows slower than the size of an equivalent FORTRAN program). Finally we discuss the impact of our results on the practice of software cost estimating.

A. PSS and LSS Counts

The results of applying the code counting methods to the FORTRAN and Ada examples using the conventional programming style examples are summarized in Table 2.

There are several interesting aspects to the results. The PSS count is always greater than the LSS count. The Ada code count is in every case greater than the FORTRAN code count. The Ada code count is, on average, 50 percent greater than the FORTRAN count when measured by PSS. The Ada code count is, on average, 40 percent greater than the FORTRAN count when measured by LSS. McGarry and Agresti (McGarry, 1989), in an experiment of parallel development of flight dynamics systems by two teams of programmers, one team using FORTRAN and the other team using Ada, reported the Ada product was significantly larger (measured by PSS) than the FORTRAN product by a factor of almost three. McGarry and Agresti posit three reasons for the large difference in the counts. First, the characteristics of the Ada language itself (about which more will be said

Table 2. Lines-of-code count for four programs.

	FORTRAN		Ada		Ada/FORTRAN	
Program	PSS	LSS	PSS	LSS	PSS	LSS
Moments	68	61	124	109	1.82	1.78
Quicksort	92	79	141	106	1.53	1.34
Fourier	133	115	189	147	1.42	1.27
Orbit	1101	803	1382	1065	1.25	1.32
Mean:					1.51	1.43

in the next sections). Second, additional functionality was built into the Ada version (the Ada team developed a more contemporary screen-oriented user interface). Third, the Ada version was not driven by tight schedules and funds as was the FORTRAN version thus there was a tendency to continually add capability to the Ada version. Our experiment controlled for the latter two factors. Our results also suggest that as the size of the program grows, the difference between the FORTRAN and Ada counts falls. This suggests that Ada exhibits economies of scale relative to FORTRAN.

In order to determine if the observed differences between the Ada and FORTRAN code counts are statistically significant we conducted a nonparametric test. We would expect Ada to be greater than FORTRAN half of the time and less than FORTRAN half the time. As noted, the Ada program code counts were always greater than the FORTRAN counts. Obviously we would reject the null hypothesis that the Ada and FORTRAN counts were the same. One might wonder if the programs were decomposed into their smaller constituencies whether the same results would be observed. We decomposed the four programs into 17 corresponding modules. In only one out the 17 components was the Ada not larger than the equivalent FORTRAN component. Here again we would reject the null hypothesis that the Ada and FORTRAN counts were the same.

In carrying out this experiment, we observed that FORTRAN, like Ada, has optional variations in style which can change the number of lines in a subroutine depending on the formatting used. We also observed that certain kinds of statements were more verbose in Ada than in FORTRAN, such as input and output statements, while other kinds of statements, such as assignments to structured data, could be expressed more efficiently in Ada. The impact that these variations in style are discussed in detail in the next two sections.

B. Declarations

FORTRAN allows the implicit declaration of variables, where the data type is implied by the first letter of the name (beginning a symbolic name with the letters "I" through "N" implies an integer while any other letter implies a real number). In spite of this allowance, most programming practices now dictate explicit declaration as a way of avoiding certain kinds of errors. Nevertheless, it is common in FORTRAN to use a single statement to declare all the variables of a certain type rather than to place each declaration on a separate line. Conversely, most of the guides about Ada style recommend using a separate line for

each declaration. This allows the initialization of variables during the elaboration of their declarations, and also improves the maintainability of the code, though it tends to inflate both the LSS and the PSS for Ada when compared with FORTRAN. As discussed earlier, in order to understand the variability in program size due to the observance of these and other conventions, we wrote and compared both terse and verbose versions of each routine in each language.

Another stylistic issue which tends to increase the size of a program written in Ada over a similar one written in FORTRAN is the use of descriptive names. Since FORTRAN symbolic names are limited in length to six characters (ANSI, 1978, section 2.2) it is often easier to fit a long expression which contains several names on a single line. In several of our examples, multiple editor lines were required to write an expression in Ada which only took one line in FORTRAN. One might argue that the descriptive choice of names in Ada might reduce the need for in-line commentary as compared with a corresponding FORTRAN program, meaning that the effect on the size of a fully commented program may be counterbalanced. However, since we did not study the effects of commenting on program size, we did not attempt to investigate this possibility. Further, this issue only affects the physical line count (PSS) and not the logical count of statements and declarations (LSS).

In Ada, formal parameters are declared along with the name of a subprogram, rather than in a subsequent declarative area, as is the case with FORTRAN. Depending on the counting method used, the effect of this on size is often canceled out since this makes the program unit declaration longer in Ada but it eliminates the need to repeat the parameter names in a later declaration.

Ada allows, but does not require, the declaration of a library-level subprogram (i.e., a procedure or function) to be compiled separately from its executable body. If this separation is not done there will be a reduction in the number of lines needed to write a given program. However, most style guides recommend the addition of these lines since it can greatly reduce the recompilation effort required if a subprogram body is modified. A separate subprogram declaration can also be used within a declarative area, usually to allow mutual visibility between two locally-declared subprograms. In the case of subprograms exported from a package declaration, the subprogram declarations are always separated from their bodies, which appear in the package body. In all these cases, however, the extra programming effort required to provide a separate subprogram declaration is negligible since it is simply a verbatim repetition of the specification part of the subprogram body. In fact, some Ada development environments automatically complete the repeated syntax so that no additional typing or editing is required of the programmer. (It might be argued that maintenance is made more complicated by this syntactic duplication in the language since both copies have to be modified in the case of a change. However, not only will a compiler immediately detect an inconsistency, but the more likely maintenance situation is a change to the unique code in a subprogram body rather than the redundant interface code in the specification.) For these reasons, one might argue that separate subprogram declarations should not be included in the size of a program. However, we felt that it was not worth the added complexity of defining counting rules to compensate for this (see appendix A for further discussion of counting methods).

Since FORTRAN does not allow the definition of structured data types, arrays are often

used for various logical data structures. This simplifies the declaration of such structures, since it only requires a dimension statement, but is at the possible expense of more elaborate processing later in the program. To assign an array value to an array object in FORTRAN it is necessary to use a loop which explicitly assigns each component. In comparison, Ada array objects contain implicit information about their own size and bounds which allows them to be assigned to one another with single assignment statements.

The manipulation of arrays which represent nested data structures can require even more complexity. For example, the FORTRAN version of the fast Fourier transform used in one of our examples uses an array of real numbers to represent an array of complex numbers, with the odd-indexed values being the real parts and the even-indexed values being the imaginary parts. This required the "do" loops to use an increment of two rather than one each time the complex numbers were processed. When we initially translated these into Ada, extra statements were required to implement these loops since Ada does not allow "for" loops which skip values in the loop range. When the algorithms were written in a more appropriate Ada style using structured data, however, the loops were reduced to half the number of statements required by their FORTRAN equivalents.

C. Statements

One of the most noticeable differences in program size between Ada and FORTRAN programs was in the statement areas used for input and output. Since Ada only allows a single value to be either input or output with each statement, the translation into Ada of a formatted FORTRAN I/O statement often resulted in considerable expansion. This effect can be clearly observed by comparing the driver routines in Ada and FORTRAN for the example engineering algorithms. For example, the three "write" statements in the driver for the Fourier transform routine required five statements while the same output in Ada required 27 statements. In actual practice, however, such verbosity would typically be eliminated by the use of predefined I/O utilities which serve to simplify the process of performing I/O from within application code. (We tested the extent of the I/O overhead by substituting calls to hypothetical I/O utilities in each example. The Ada examples were reduced over 16% by this enhancement whereas the FORTRAN examples were reduced less than 7%. Nevertheless, these changes did not alter the rank ordering of program size when comparing the resulting Ada and FORTRAN examples.)

One of the stylistic issues that allows a single FORTRAN program to be written with different numbers of lines is the use of separate "format" statements when specifying input and output columns rather than including this information directly in the "read" or "write" statements. Our FORTRAN program examples, which were originally written without "format" statements have been re-styled to conform to the conventions found in (ANSI, 1978) in order to make them representative of industry programming standards. The verbose and terse versions of the FORTRAN programs explore the different ways to effect input and output formatting.

There were several minor differences between the syntactic conventions used in the two languages which were noticed when applying the counting rules chosen. One minor differ-

ence between the languages is that Ada always implicitly declares loop variables. This is to ensure that the availability of that variable is limited to the scope of its loop, but this also has the effect of reducing the size of the program by one declaration. Another minor difference is the implicit "return" statement at the end of an Ada subprogram. A "return" statement is still required if processing is to stop at any other point but most Ada subprograms are written to return after their last statement. In a FORTRAN routine, the last line must be an "end" statement. It has the same effect as a "return" statement, which is to return control to the referencing program unit. Nevertheless, it is common to see both a "return" and an "end" statement in a FORTRAN subprogram. A third minor difference is the lack of a need for a "continue" statement in Ada. Although a "continue" statement is rarely required in FORTRAN, it is common practice to use one at the end of a loop to avoid confusing the last statement in the loop with the statements which follow the loop. In comparison, Ada loops require an "end loop," however this increases only the number of physical lines (PSS) and not the number o logical statements (LSS).

It should be noted than an inconsistency between the LSS methods for Ada and FORTRAN existed. In FORTRAN the "else" statement is counted as a logical source statement, but in Ada it considered part of the same logical statement as its containing "if" statement. Thus it not counted as an additional logical source statement in Ada (SPC, 1991).

D. Economies of Scale in the Ada Language

As noted above, as the size of the program grows for the four programs in the experiment, the difference between the FORTRAN and Ada count falls. An interesting question concerning this observed scale effect is at what program size would the Ada code count fall below the FORTRAN code count? A graphical representation of this cross over or break even point is presented below. The Ada and FORTRAN lines of code (measured in thousands) are represented on the y and x axis respectively. The ray that passes through the origin at 45 degrees represents points where the number of Ada and FORTRAN lines of code are equal. The curved line represents a hypothetical relationship between Ada and FORTRAN. If economies of scale exist, we would expect this relationship to exhibit a curvilinear form similar to that depicted in Figure 1. This form suggests that as the size of the program grows the number of Ada lines required to perform the function grows more slowly than does the corresponding FORTRAN program. The point where the two lines intersect represents the break-even point. The interpretation of this point is that any program of size greater than B could be written with fewer lines of Ada than equivalent FORTRAN lines.

We were hoping to be able to extrapolate the actual shape of the curvilinear line relating the sizes of functionally equivalent Ada and FORTRAN programs. As mentioned, the trend seen from our samples implies that the overhead for Ada programs drops as the program size grows. However, a regression analysis indicated that the break-even point would be several times larger than our largest example and we therefore felt that our data was insufficient to make this extrapolation. However, there is anecdotal evidence from NASA/SEL and others that agrees with our impression that Ada programs of sufficient size become smaller than equivalent FORTRAN programs (NASA, 1994).

The question remains as to why we should observe this scale effect with the Ada lan-

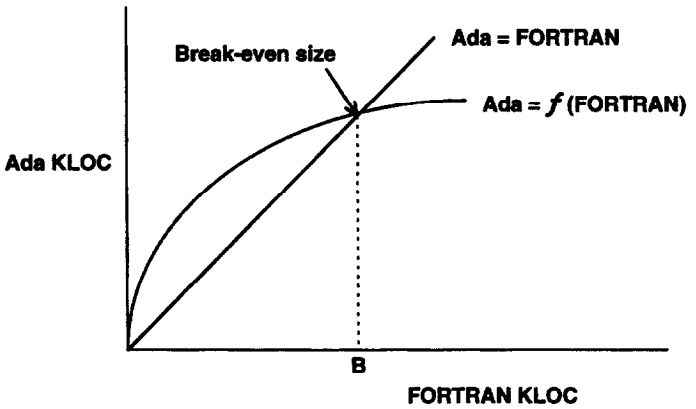


Figure 1. Break-even size.

Table 3. Executable and declarative code count.

	FORTRAN LOC		Ada LOC	
Routine	Executable	Declarative	Executable	Declarative
Moment	52	9	71	38
Quicksort	72	7	81	25
FFT	104	11	110	37
Orbit	738	65	701	364

guage. Several differences between the two languages that were especially noticeable may contribute to the effect. One difference was the fact that as program size increased, the executable portions increased slower in Ada than in FORTRAN. Although the declarative portions increased more in Ada than in FORTRAN, they contributed less to overall size. In our largest example, the executable portion was smaller in Ada than in FORTRAN even though the overall size was greater in Ada. Table 3 presents a view of the four test programs separated into their executable and declarative portions.

An example from the FFT test program illustrates this tradeoff between the number of executable and declarative statements. Table 4 presents functionally equivalent Ada and FORTRAN code taken from the FFT program.

If our results that indicate there are significant differences in size between functionally equivalent FORTRAN and Ada programs are correct, then the practice of cost analysts to simply use Ada and FORTRAN lines of code counts interchangeably will induce errors in the subsequent cost estimates.

How large these potential cost estimation errors can be is not only a function of the different sizes of equivalent programs in the two languages, but is also a function of the relative effort required to program a line of code in each language. Several cost models which attempt to represent the relationship between size and effort for each language have

Table 4. Declarations vs. statements.

FORTRAN	Ada
REAL Data (2*nn)	type Complex is record
REAL tempi, tempr	Real : Float;
REAL wi, wr	Imaginary : Float;
	end record;
	type Complex_Array is array (Natural range<>)of Complex;
	function "+" (Left,Right:Complex) return Complex;
	function "-" (Left,Right:Complex) return Complex;
	function "*" (Left, Right:Complex) return Complex;
	Data : Complex_Array (1..N);
	W, Temp : Complex;
.	.
.	.
.	.
tempr=wr*Data(j) - wi*Data(j+1)	Temp :=W*Data (J);
tempr=wr*Data(j+1) - wi*Data(j)	Data (J) := Data (I) - Temp;
Data(j)=Data(i) - tempr	Data (I) := Data (I) - Temp;
Data(j+1)=Data(i+1) - tempi	
Data(i)=Data(i) + tempr	
Data(i+1)=Data(i+1) + tempi	

been reported. When we compared one model which is frequently used for FORTRAN cost estimation (Boehm, 1981) with a more recent report of the cost of writing Ada code (Giallombardo, 1992) we found a difference of 25% when estimating 10,000-line programs and a difference of 15% for 100,000-line programs. Obviously the results are sensitive to the effort estimating equation used. However, the point is that significant error can result from this practice of indiscriminately interchanging code counting units.

4. Conclusions and Future Research

The main objective of this research was to fill a gap in the knowledge needed by experienced FORTRAN size and cost estimators when estimating Ada developments for the first time. Although there are published models for the cost of developing Ada programs based on their expected size, there has been no standard way of knowing what the size of an Ada development is likely to be based on the expected size of an equivalent FORTRAN development. By showing with this work that the sizes of functionally equivalent programs in Ada and FORTRAN differ, we have demonstrated that it is wrong to assume that simply using a FORTRAN effort estimate is sufficient, nor is it sufficient to use the expected number of FORTRAN lines of code in an Ada estimating equation. With the added knowledge of how the sizes of functionally equivalent programs in Ada and FORTRAN compare, a cost estimator can first adjust the expected number of lines of FORTRAN code to complete a job to a more accurate estimate of the expected number of lines of Ada code. Then, an Ada effort estimating equation may be properly applied.

This study should be viewed mainly as a model for further investigation, although we

feel our limited results are still of interest. In particular, we suspect that the tendency we observed for small Ada programs to be larger than their functionally equivalent FORTRAN counterparts is reasonable, as is our further observation that the overhead for Ada diminishes as the program size is increased. Our limited data suggested that there may even be a crossover point beyond which the size of an Ada program is smaller than a functionally equivalent FORTRAN program. Although our number of observations was small and entirely below this projected crossover point, one of the strongest pieces of evidence that such a point exists is that the number of executable lines of Ada in our largest example was smaller than the equivalent number of executable lines in FORTRAN. Through an inspection of language features, we felt that this is a reasonable occurrence, since Ada has richer declarative power and, in return, can take advantage of simpler algorithmic processing.

Since the relationship between the sizes of functionally equivalent Ada and FORTRAN programs is probably not linear, more observations are needed, and in particular, observations are needed which are at least an order of magnitude greater than the largest of our examples. The only published comparison of the sizes of a pair of large Ada and FORTRAN programs developed from the same set of requirements was too confounded to be useful for this purpose (McGarry, 1989). The only other evidence we found about the way larger Ada and FORTRAN programs compare was anecdotal, although the feelings being reported tended to agree with our observations (NASA, 1994).

A final observation from this study, which was not previously discussed, stems from our interest in examining the possible variations of program size due to programming style. Although we always used what we considered to be a conventional style of formatting in the programs used in our analyses, we additionally wrote both terse and verbose styles for each example. The most interesting result we observed was that the possible variation in size for an Ada program is much greater than the possible size variation for a FORTRAN program. This means that the comparison of Ada size, effort and productivity results across organizations which may not be observing the same style standards is more prone to error than are similar comparisons using FORTRAN results. Although we used well-defined counting rules for both languages to maximize the portability of our results, we were not able to similarly well-define a programming style. In order to assure comparability of Ada size, effort, and productivity results across organizations, more study is needed into how the size of an Ada program might be normalized for any implemented functionality.

Acknowledgments

This work was supported under the Institute for Defense Analyses' Independent Research Program. The views expressed here are solely those of the authors, not the Department of Defense nor the Institute for Defense Analyses. The authors acknowledge helpful comments from Bruce Angier, Stephen Balut, Linda Brown, and two anonymous referees.

Notes

1. There are several data bases containing observations of productivity on Ada projects. One of the best examples is the work done at the MITRE Corporation and reported in (Giallombardo, 1992).
2. The definitions of PSS and LSS are identical to the definitions in (IEEE, 1993).

References

ANSI, 1978. *American National Standard Programming Language FORTRAN 1978.* New York: American National Standards Institute, Inc.

Boehm, B. 1981. *Software Engineering Economics.* New Jersey: Prentice-Hall, Inc.

Giallombardo, R. 1992. *Effort and Schedule Models for Ada Software Development.* MTR 11303, Bedford, MA: MITRE Corporation.

IEEE 1993. *Standard for Software Productivity Metrics.* IEEE Std 1045-1992, New York: Institute of Electrical and Electronics Engineers, Inc.

Jones, C. 1991. *Applied Software Measurement.* New York: McGraw-Hill, p. 76.

Kemerer, C. 1991. Software cost estimation models. *Software Engineering Reference Handbook.* Surrey, UK: Butterworth's Scientific Limited.

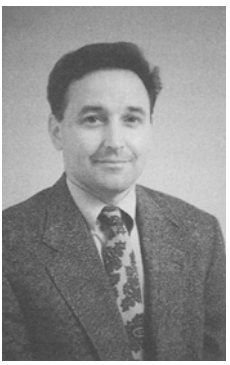
McGarry, F., and Agresti, W. 1989. Measuring Ada for software development in the software engineering laboratory. *The Journal of Systems and Software* 9: 149–159.

NASA, 1994. Conversations with SEL personnel at Computer Sciences Corporation in Greenbelt, Maryland, and estimates from Kaman Sciences personnel in Colorado Springs, Colorado.

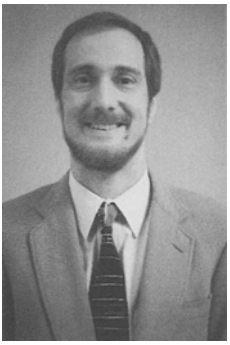
Press, W., Teukolsky, S., Vetterling, W., and Flannery, B. 1992. *Numerical Recipes in FORTRAN, The Art of Scientific Computing.* Second Edition, Cambridge, MA: Cambridge University Press.

SPC 1991. *Code Counting Rules and Category Definitions/Relationships.* CODE_COUNT_RULES-90010-N, Version 02.00.04. Hendron, VA: Software Productivity Consortium Inc.

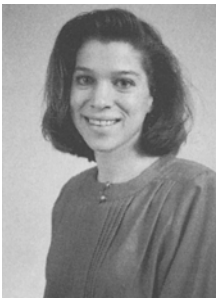
SPC 1992. *Ada Quality and Style: Guidelines for Professional Programmers.* SPC-91061-CM, Version 02.01.01, Herdon, VA: Software Productivity Consortium Inc.



Thomas Frazier is a member of the Research Staff at the Institute for Defense Analyses in the Cost Analysis and Research Division. He holds B.S. and M.A. degrees from Ohio University and a Ph.D. in Economics from The American University. Dr. Frazier's research interests include the economics of software development and maintenance and the application of statistical methods as a tool to evaluating DoD policy questions. He has published articles in variety of journals including the *Review of Economics and Statistics* and *Naval Research Logistics.*



John Bailey is an Adjunct Research Staff Member at the Institute for Defense Analyses in the Cost Analysis and Research Division. After earning bachelor's and master's degrees in music performance, Dr. Bailey completed M.S. and Ph.D. degrees in Computer Science at the University of Maryland. In addition to playing string quartets, Dr. Bailey's interests include research into software measurement and methods which assist in the development of high quality software products.



Melissa Corso is a Project Director at Health Management Systems in New York City. Before joining Health Management Systems, Ms. Corso was a Research Assistant at the Institute for Defense Analyses. She holds a B.A. degree in Mathematics, summa cum laude, from Hartwick College and a M.S. degree in Operations Research from The College of William and Mary.