

"PROGRAMMING EFFORT" ANALYSIS OF THE ELLPACK LANGUAGE

John R. Rice

Division of Mathematical Sciences

Purdue University

CSD-TR 288

September 12, 1978.

ABSTRACT

We study the programming effort for problem solving in the context of partial differential equations. Three alternatives are considered: (1) Using ELLPACK, a problem statement language. (2) Writing a control program for a set of powerful library routines. (3) Programming the entire solution in Fortran. The measures of programming effort used are (1) lines of code needed, (2) number of operators and operands used and (3) elementary mental discriminations required. The latter two measures are from Halstead's software science methodology. The conclusions reached are:

- A. Using a powerful library increases programming productivity by a factor of 10 compared to programming the solution in Fortran.
- B. Using a problem statement language increases programming productivity by a factor of 100 to 500 compared to programming the solution in Fortran.

To Appear: SIGNUM NEWSLETTER, Vol. 13, No. 4, December, 1978.

"PROGRAMMING EFFORT" ANALYSIS OF THE ELLPACK LANGUAGE

John R. Rice

Division of Mathematical Sciences

Purdue University

ELLPACK is a problem statement language and system for elliptic partial differential equations (PDEs) which is implemented by a Fortran preprocessor. ELLPACK's principal purpose is as a tool for the performance evaluation of software. However, we use it here as an example with which to study the "programming effort" required for problem solving. It is obvious that problem statement languages can reduce programming effort tremendously; our goal is to quantify this somewhat. We do this by analyzing the length and effort (as measured by Halstead's "software science" technique) of various approaches to solving these problems.

A simple ELLPACK program is shown below to illustrate the nature of the ELLPACK language. Space does not allow a description of the language but it is somewhat self explanatory. See [2] and [3] for further details.

```
*      ELLPACK 77 - EXAMPLE 4 FOR SIGNUM CONFERENCE
EQUATION.  2 DIMENSIONS
          UXX$ +6.UYY$ -4.UY$ +(DUBS(X)-3.)U = EXP(X+Y)*DUBS(X)*(2./(1.+X)-1.)
BOUND.     X = 0.0      , U = TRUE(0.0,Y)
          Y = 1.0      , UY= EXP(1.+X)*SQRT(DUBS(X)/2.)
          X = EXP(1.)  , U = TRUE(2.71828182846,Y)
          Y = 0.0      , MIXED = (1.+X)U (1.+X)UY = 2.*EXP(X)
GRID.      UNIFORM X = 5      $      UNIFORM Y = 7
*
DISCRETIZATION(1).  5-POINT STAR
DIS(2).           P3-C1 COLLOCATION
INDEX(1).         NATURAL
INDEX(2).         COLLOCATE BAND
SOL.              BAND SOLVE
OUTPUT(B).        MAX-ERROR $      MAX-RESIDUAL
OUTPUT(99).       TABLE(5,5)-U
SEQUENCE.         DIS(1) $ INDEX(1) $ SOLUTION $ OUTPUT(B)
                  DIS(2) $ INDEX(2) $ SOLUTION $ OUTPUT(B)
                  OUTPUT(99)
OPTIONS.          MEMORY $ LEVEL=2
FORTRAN.
      FUNCTION TRUE(X,Y)
      TRUE = EXP(X+Y)/(1.0+X)
      RETURN
      END
      FUNCTION DUBS(T)
      DUBS = 2./(1.+T)**2
      RETURN
      END
END.
```

A problem solution with the ELLPACK system goes through three principal stages: (1) the ELLPACK language input is read by a Fortran preprocessor which writes a Fortran Control Program. (2) The Control Program is compiled. (3) The Control Program object deck is loaded along with modules from the ELLPACK library which implement steps in the solution of the PDE. We compare the programming effort for each of these steps, i.e. (1) An ELLPACK statement of the PDE problem to be solved and method to be used. (2) Preparation of the Control Program, assuming familiarity with the module library and (3) Programming the entire solution in Fortran.

Three measures of programming effort are used: lines of code, total number of operators and operands and "effort" measured by thousands of elementary mental discriminations. The latter two measures are part of Halstead's "software science" presented in [1]. This is an empirical method to define and relate various program parameters to the effort required to write the programs. While we do not attempt to explain this method, it is very plausible that the total number of operators and operands in a program is more directly related to the complexity of a program than the number of lines of Fortran. Two shortcomings of the method for this application is that it ignores declarations and and I/O statements and the mechanism for measuring the effort estimates for a set of tightly integrated subroutines is underestimated. However, the measurements are good enough for the present purposes where only rough accuracy is needed.

We consider 10 example problems and present the data N =total number of operators and operands, L =total lines of code (including comments in the Fortran modules, most of which are well commented) C =code complexity measured by number of operators and operands per line, and E =programming effort in 1000's of elementary mental discriminations as defined by Halstead. For each problem we have data for the ELLPACK language (labeled ELPK), the Control Program (labeled Control) and the set of library subroutines used (labeled Modules).

	PROBLEM 1			PROBLEM 2			PROBLEM 3			PROBLEM 4		
	ELPK	Control	Modules	ELPK	Control	Modules	ELPK	Control	Modules	ELPK	Control	Modules
N	187	1793	14,349	103	1331	6632	147	1552	14,203	134	1354	12,671
L	33	381	3,852	22	295	1330	27	353	5,348	29	314	3,402
C	5.7	4.7	3.7	4.7	4.5	5.0	5.4	4.4	2.7	4.6	4.3	3.7
E	27	1076	6,425	5	371	4804	14	852	4,232	12	614	5,881

PROBLEM 5			PROBLEM 6			PROBLEM 7			PROBLEM 8			
	ELPK Control Modules			ELPK Control Modules			ELPK Control Modules			ELPK Control Modules		
N	113	1231	11,198	125	1358	15,113	125	1366	8500	102	1238	7,261
L	40	303	2,918	42	336	5,435	51	311	2561	29	303	2,145
C	2.8	4.1	3.8	3.0	4.0	2.6	2.5	4.4	3.3	3.5	4.1	3.4
E	8	385	5,306	12	587	3,784	11	444	2771	6	394	2,211

PROBLEM 9			PROBLEM 10			
	ELPK Control Modules			ELPK Control Modules		
N	112	1283	14,134	87	1716	7997
L	38	315	3,937	110	365	2517
C	2.9	4.1	3.6	.8	4.7	3.2
E	6	503	6,739	4	390	3243

There are considerable variations among these examples but there is also an obvious trend of greatly increased "length" from stage to stage, no matter how it is measured. The programming effort E should increase faster than the number of lines, but it does not always do so because of the inability of the software science method to completely account for the use of modularity in implementing an algorithm.

Comparing the Control and Modules data should be representative of the comparison of using or not using a library of powerful subroutines. We see that the ratios of effort range from 6 to 15 with 10 as an average, the ratios of lines range from 6 to 17 with 11 as an average. Thus we conclude that, at least in the context of solving PDEs, the use of a library increases programming productivity by a factor of 10. It may well increase it more and the quality of the results will be improved if the library is good.

Comparing the ELPK and Control data should measure the value of a problem statement language compared to using a library. The ratios of effort range from 40 to 100 with 60 as an average and the ratios of lines range from 3 to 13 with 9 as an average. We thus conclude that using an ELLPACK type preprocessor increases programming productivity by a factor of 10 to 50.

We also conclude that using this preprocessor instead of writing the programs from scratch reduces programming effort by a factor of between 100 and 500.

This work is partially supported by NSF Grant MCS76-10225.

- [1] M. H. Halstead, Elements of Software Science, Elsevier North-Holland, New York, 1977.
- [2] J. R. Rice, ELLPACK: A Research Tool for Elliptic Partial Differential Equations Software in Mathematical Software III (J. R. Rice, ed.) Academic Press, 1977, pp. 319-341.
- [3] J. R. Rice, ELLPACK 77 User's Guide, CSD-TR 226, Computer Science Dept., Purdue University, September 1978.