1978

The Relationship Between Student Grades and Software Science Parameters

V. Y. Shen

Report Number:
78-294

THE RELATIONSHIP BETWEEN STUDENT GRADES AND SOFTWARE SCIENCE PARAMETERS

TR-294

V.Y. Shen
Computer Sciences Department
Purdue University
W. Lafayette, IN 47907

Abstract

The evaluation of programmers' performance has been an important subject in recent years. Making the reasonable assumption that a good programmer would consistently write programs of high quality, the problem of evaluation would be simplified if an objective measure of quality were available which could be applied to a sample program. This paper describes an experiment to study the relationship between grades in a programming course, which are the accepted evaluations of student programmers, and certain objectively computed software science parameters on program samples. The data shows that two of the parameters are correlated to the grades.

Keywords and Phrases: Software science, programmer evaluation, software measurement, grading of student programs

CR Categories: 4.0, 4.6

I. Introduction

The demand for computer programmers has made their training and evaluation an important subject. A good programmer should not only write programs that produce correct results, but also write them in a form that is easy to maintain. The latter requirement refers to programs being written in "good style", so that there are few "bugs" to begin with, and these bugs are easy to detect and fix. This is the main emphasis of the second course on computer programming offered by the Computer Sciences Department at Purdue University (CS 320). Our approach is to teach the concept of structured program development using D-charts [1,2], reinforced by examples taken from a book on programming style [3]. Students are asked to work out problems in FORTRAN and PASCAL which are then graded based on both correctness and style. We hope that the students receiving better grades in CS 320 are "better" programmers.

The grading of program correctness is relatively easy. It is done by generating a test file containing a large number of test cases for a given problem. A program is considered correct if it produces correct results for all of these cases. Grading of style, on the other hand, is highly subjective. The graders are instructed to skim through the program, mark any items of apparent poor style and deduct points. There is no assurance that all such items are found, and there is considerable inconsistency among the graders on how the points are deducted. Thus an effective and objective method to grade the programs is urgently needed.

One of the discoveries of software science [4,5] is that if we consider a computer program as a sequence of operators and operands, together with the following definitions:

N_1: total usage of operators

N_2: total usage of operands

η_1: unique operator count

η_2: unique operand count

$N = N_1 + N_2$: measured program length

$\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$: predicted program length

then there is a simple relationship between the measured length N and its vocabulary η_1 and η_2. This relationship, called the "length" equation, has been validated by several researchers and is stated as

$$N = \hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$$

It should be noted, however, that any computer program can be taken and altered in such a way that the length equation will not hold, even though the result of executing the program remains the same. For example, a statement in a program that references an operand A and uses the two operators "+" and "-" may be modified by adding and then subtracting A one-thousand times, thus increasing N_1 and N_2 without changing η_1 or η_2. Several such methods that would affect the length equation are identified and called "impurity classes". * A hypothesis is that the length equation will hold if a program does not have any impurities, or is "pure". Since all of the impurity classes are examples of poor programming style, it is of interest to investigate whether an objective measure of style for a given program exists in the deviation from the length equation. This paper reports the findings of one experiment.

II. The Environment of the Experiment

The students in CS 320 (fall semester, 1977) were required to write six computer programs, take three tests and complete a programming project of approximately 1500 statements. The six programs were graded on an equal basis and constituted 20% of the final course grade. The three tests and the project were worth 80% at 20% each. A decision was made to collect the programs for the second assignment (LEX2), a FORTRAN subroutine to count the number of arithmetic operators in FORTRAN assignment statements. This assignment was given after the students received a brief review of FORTRAN, the subject covered in the prerequisite course. They were told to finish the assignment quickly, receiving neither instruction nor hints on program structure and style. The programs were saved on tape in machine readable form.

Unlike the second assignment the programming project was given after the instruction on program structure and style was completed. The large size of

* The impurity classes include complementary operations, ambiguous operands, synonomous operands, common subexpressions, unwarranted assignment, and unfactored expressions [4].

the project prompted the decision to analyze just one subroutine in the project (LEXP), which was the expanded version of the second program. The subroutine was specified to perform a complete lexical analysis of FORTRAN statements, including most of the FORTRAN keywords. The two programs together contributed less than ten percent of the course grade, and were analyzed after grades had been assigned using traditional methods.

An analyzer program, which was written for software science research and used also for detecting program plagiarism [6], processed the collected programs, giving the results shown in Table 1. Every student is identified by a three-character identifier assigned by the computing center. The GRADE column contains the course grades of the students. The column LEX2N represents the measured lengths of the second program N, and the column LEX2H the corresponding predicted lengths $\hat{N}$. The column LEXPN represents the measured lengths of the selected subroutine in the project, and the column LEXPH the corresponding predicted lengths. Since the subroutine in the project is of special interest, additional data is recorded. The columns ETA1 and ETA2 are the unique operator and operand counts, respectively. The columns N1 and N2 refer to the total number of operators and operands in the project subroutine.

Of the 68 students who did the second program correctly, seven did not complete the course. Twenty-four of those who did finish had trouble saving the subroutine from the project according to our instructions. We were unable to access their files and, when the problem was discovered, the students had already left the University for Christmas vacation. Six of those who did save the programs had errors in the program and thus were rejected. The complete analysis was done using the 31 remaining students who had grades ranging from 48 to 99, the average being 79.8. The average grade for the 61 students who did complete the course was 74.5. The missing data points are represented by "-0" in Table 1.

III. The Analysis

The first step was to see how well the students' programs fit the length equation. Figure 1 shows the data for the first set of programs (LEX2) for the 31 students whose programs were correct and available to us. Many points lie near the 45° line, showing that they follow the length equation. However,

a significant number of data points lie below the line, possibly indicating the unnecessary and redundant use of operators and operands. Since the students wrote this program before they were taught "good programming practices", we hoped that the data for the project (LEXP) would fit better. Figure 2 shows this data for the same students. One can not say that there is a marked improvement in fitting the length equation, however.

Since the purpose of the experiment was to find a way to rate the students, it is of interest to see if the better students tended to write programs that deviated less from the length equation. Figure 3 shows the absolute error for LEXP, defined as $\left|\frac{N-\hat{N}}{\hat{N}}\right|$, plotted against the students' grades. It is surprising to find that students NTS, who had the highest course grade, also had the smallest absolute error. Statistical analysis shows that the correlation coefficient between the absolute error and grades is -0.46, with F = 7.64 and Significance = 0.01. There is a less than 1% chance that the absolute error and grades are not correlated. Figure 4 shows the absolute error for LEX2 of the 31 students, with correlation coefficient = -0.25. One possible explanation is that the students were taught to write programs with smaller deviation from the length equation, and the better students were more successful in accomplishing this goal.

The impurity classes identified in [4] include both cases that tend to make N larger than $\hat{N}$ and cases that tend to have the reverse effect. It is certainly possible to add impurities to any given program for the purpose of reducing the absolute error from the length equation. Thus we do not expect that to be the only measure for good programs. Another possible measure is the language level λ, defined as

$$\lambda = L^2 V = \left(\frac{2}{\eta_1}\ \frac{\eta_2}{N_2}\right)^2 N \log_2(\eta_1+\eta_2)$$

Although the language level is intended to compare programs written in different languages, the large variances reported in [4] show that programmers frequently use just a subset of the features of a given language. For example, the language PL/1 is generally considered to be at a higher level than FORTRAN. But a novice PL/1 programmer who was previously trained in FORTRAN may only use the PL/1 features that are available in FORTRAN, thus

not showing the effect of the higher language level. One would expect that given the same language, a better programmer would use the more advanced features and write programs at a higher level. Figure 5 shows the language level for LEXP. The correlation coefficient between λ and grades is 0.4, with F = 5.55 and Significance = 0.025.

Writing a program at different language levels may cause significant changes in the deviation from the length equation. For example, the statement A = B + C * 1000 could be replaced by an equivalent statement using only the addition operation, which will be a lot longer. For short programs, the introduction of a new language feature may also increase the predicted length. We decided to compute the multiple correlation coefficient for the grades and both the absolute error and the language level. The coefficient is determined to be 0.56, with F = 6.37 and Significance = 0.005! There is only a ½ percent chance that these are not correlated.

IV. Concluding Remarks

The experiment shows that certain parameters in software science are correlated with students' grades. We are currently collecting additional data to see if the results can be reproduced. A positive result would show that at least two objective measures are available for programmer evaluation.

It should be noted that there are other parameters one can measure for a given program. For example, the "effort" E of programs was studied and reported elsewhere [7]. But the value of E for a program depends on the problem that program solves. A higher value normally indicates a more complicated problem. This is also true for the parameters "volume" V, "program level" L, and "programming time" T. The parameters $\left|\frac{N-\hat{N}}{\hat{N}}\right|$ and λ, however, have accepted reference values which are independent of the problem a program solves. Thus they may tell us more about the programmer, which is ideal for programmer evaluation.

References

1. Wirth, N., "Program Development by Stepwise Refinement," Comm. ACM (14,4), April 1971, 221-227.

2. Bruno, J. and Steiglitz, K., "The Expression of Algorithms by Charts," J. ACM (19,3), July 1972, 517-525.

3. Kernighan, B. W. and Plauger, P. J., The Elements of Programming Style, McGraw-Hill, New York, 1974, 147 pages.

4. Halstead, M. H., Elements of Software Science, Elsevier North-Holland, New York, 1977, 127 pages.

5. Fitzsimmons, A. and Love, T., "A Review and Evaluation of Software Science," C. Surveys (10,1), March 1978, 3-18.

6. Ottenstein, K. J., "An Algorithm Approach to the Detection and Prevention of Plagiarism," ACM SIGCSE Bulletin (8,4), Dec. 1976, 30-41.

7. Gordon, R. D., "A Measure of Mental Effort Related to Program Clarity," Ph.D. Thesis, Purdue Univ., August 1977, 129 pages.

ID	GRADE	LEX2N	LEX2H	LEXPN	LEXPH	ETA1	ETA2	N1	N2
NQD	81.	182.	145.	840.	776.	59.	70.	504.	336.
NQE	76.	282.	145.	-0	-0	-0	-0	-0	-0
NQH	43.	164.	167.	-0	-0	-0	-0	-0	-0
NQL	91.	201.	151.	463.	399.	39.	37.	275.	188.
NQM	48.	224.	156.	598.	314.	28.	35.	342.	256.
NQP	94.	74.	109.	-0	-0	-0	-0	-0	-0
NQQ	95.	141.	146.	439.	401.	31.	45.	253.	186.
NQY	60.	188.	179.	-0	-0	-0	-0	-0	-0
NQZ	35.	179.	185.	-0	-0	-0	-0	-0	-0
NQ0	89.	127.	135.	-0	-0	-0	-0	-0	-0
NQ1	92.	251.	185.	-0	-0	-0	-0	-0	-0
NQ3	64.	140.	133.	-0	-0	-0	-0	-0	-0
NQ4	90.	84.	102.	167.	226.	27.	22.	110.	57.
NQ5	94.	103.	119.	247.	239.	29.	22.	148.	99.
NRB	-0	234.	179.	-0	-0	-0	-0	-0	-0
NRC	74.	149.	133.	463.	373.	33.	39.	269.	194.
NRE	62.	277.	294.	704.	446.	40.	43.	424.	280.
NRG	83.	161.	128.	519.	421.	32.	47.	295.	224.
NRL	41.	153.	162.	-0	-0	-0	-0	-0	-0
NRM	77.	140.	157.	-0	-0	-0	-0	-0	-0
NRN	82.	171.	157.	-0	-0	-0	-0	-0	-0
NRO	-0	250.	226.	-0	-0	-0	-0	-0	-0
NRP	82.	206.	203.	521.	442.	32.	50.	302.	219.
NRS	93.	81.	103.	247.	322.	26.	38.	144.	103.
NRU	77.	246.	227.	-0	-0	-0	-0	-0	-0
NRW	93.	150.	156.	-0	-0	-0	-0	-0	-0
NSC	55.	127.	152.	-0	-0	-0	-0	-0	-0
NSF	79.	181.	167.	580.	402.	29.	47.	327.	253.
NSG	-0	146.	140.	-0	-0	-0	-0	-0	-0
NSI	35.	124.	123.	-0	-0	-0	-0	-0	-0
NSM	56.	122.	112.	-0	-0	-0	-0	-0	-0
NSO	-0	144.	134.	-0	-0	-0	-0	-0	-0
NSP	78.	241.	163.	324.	309.	37.	25.	203.	121.
NSR	87.	162.	168.	209.	263.	27.	28.	126.	83.
NSS	87.	161.	179.	-0	-0	-0	-0	-0	-0
NSU	73.	142.	145.	320.	314.	30.	33.	190.	130.
NS2	56.	220.	139.	203.	227.	29.	20.	129.	74.
NS1	84.	157.	156.	163.	174.	23.	17.	104.	59.
NS3	84.	186.	157.	-0	-0	-0	-0	-0	-0
NS4	69.	174.	139.	336.	314.	31.	32.	197.	139.
NTB	77.	151.	139.	446.	352.	35.	34.	263.	183.
NTC	82.	225.	209.	274.	257.	29.	25.	159.	115.
NTI	89.	176.	167.	-0	-0	-0	-0	-0	-0
NTK	88.	157.	152.	-0	-0	-0	-0	-0	-0
NTO	60.	198.	145.	-0	-0	-0	-0	-0	-0
NTP	79.	228.	162.	-0	-0	-0	-0	-0	-0
NTR	-0	170.	150.	-0	-0	-0	-0	-0	-0
NTS	99.	171.	132.	195.	196.	22.	22.	114.	81.
NTT	47.	127.	139.	-0	-0	-0	-0	-0	-0
NTU	-0	250.	139.	-0	-0	-0	-0	-0	-0
NTV	-0	155.	168.	-0	-0	-0	-0	-0	-0
NTX	81.	174.	190.	-0	-0	-0	-0	-0	-0
NTZ	60.	123.	128.	-0	-0	-0	-0	-0	-0
NT1	65.	202.	202.	-0	-0	-0	-0	-0	-0
NT4	61.	248.	185.	-0	-0	-0	-0	-0	-0
NT5	87.	189.	169.	354.	276.	27.	30.	211.	143.
NUA	79.	174.	146.	-0	-0	-0	-0	-0	-0
NUH	78.	129.	128.	404.	288.	30.	29.	235.	169.
NUI	77.	149.	145.	969.	754.	59.	67.	572.	397.
NUR	85.	201.	220.	337.	269.	29.	27.	203.	134.
NUZ	82.	185.	168.	842.	639.	44.	66.	485.	357.
NU3	34.	280.	161.	-0	-0	-0	-0	-0	-0
NU4	78.	125.	123.	362.	288.	27.	32.	213.	149.
NVA	82.	187.	173.	226.	220.	24.	24.	136.	90.
NVC	84.	213.	167.	874.	533.	37.	58.	493.	381.
NVE	62.	145.	139.	257.	208.	23.	23.	152.	105.
NVI	82.	226.	167.	226.	167.	21.	18.	131.	95.
NVN	86.	177.	123.	-0	-0	-0	-0	-0	-0

Table 1. CS 320 students' grades and software science parameters.

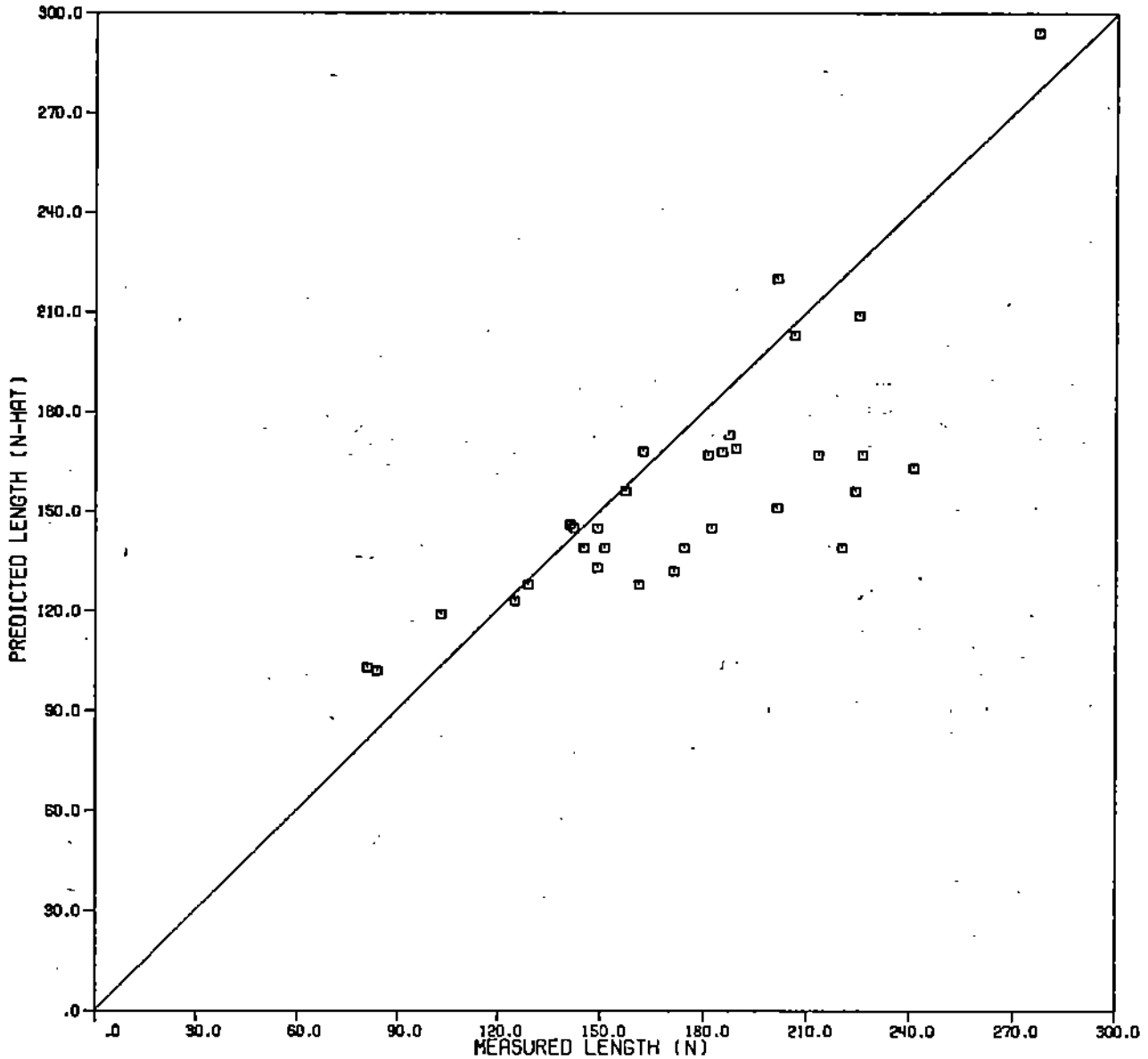

Figure 1. Length relationship for Program 2 (LEX2H vs. LEX2N).

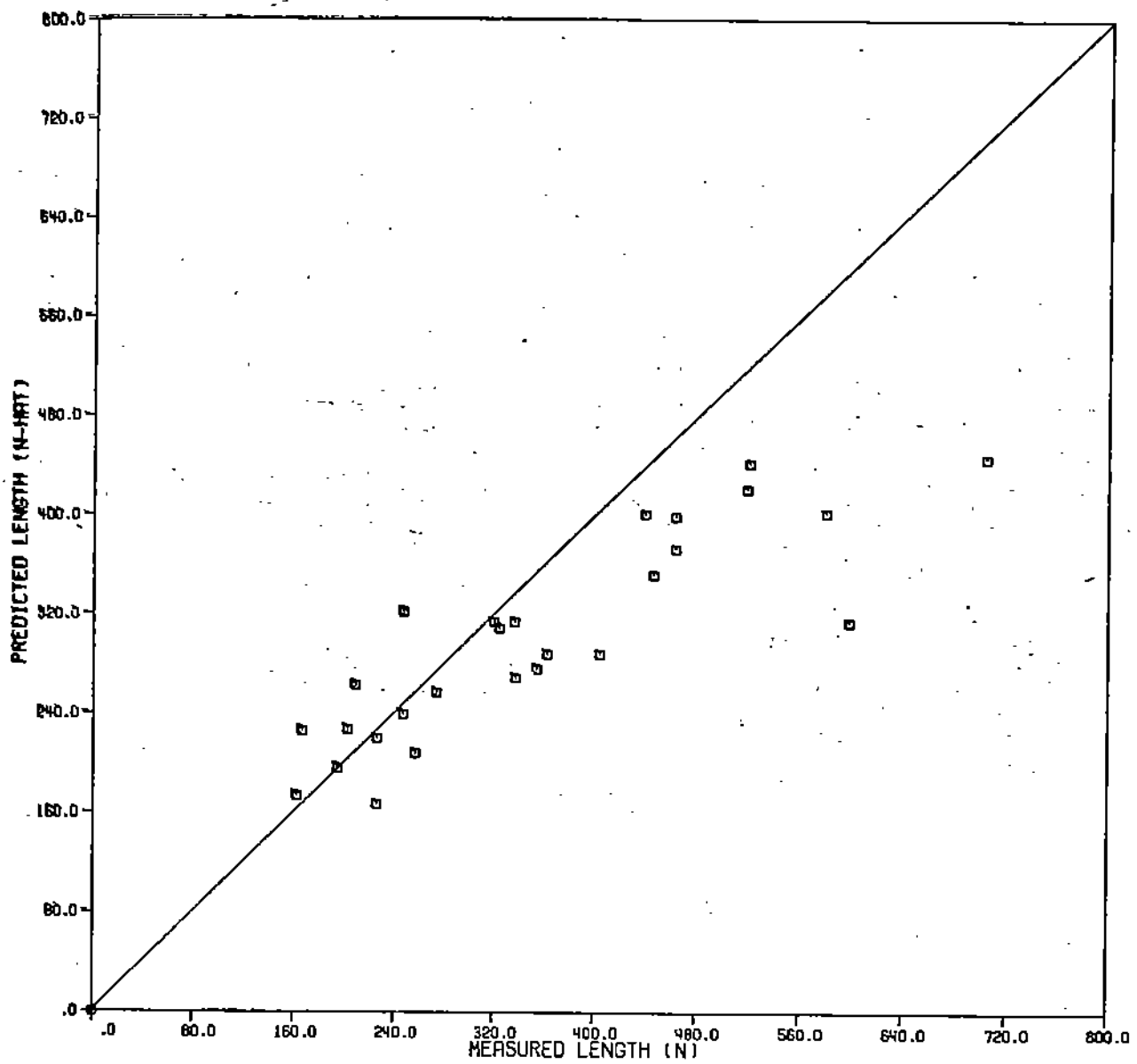

Figure 2. Length relationship for Project subroutine (LEXPH vs. LEXPN).

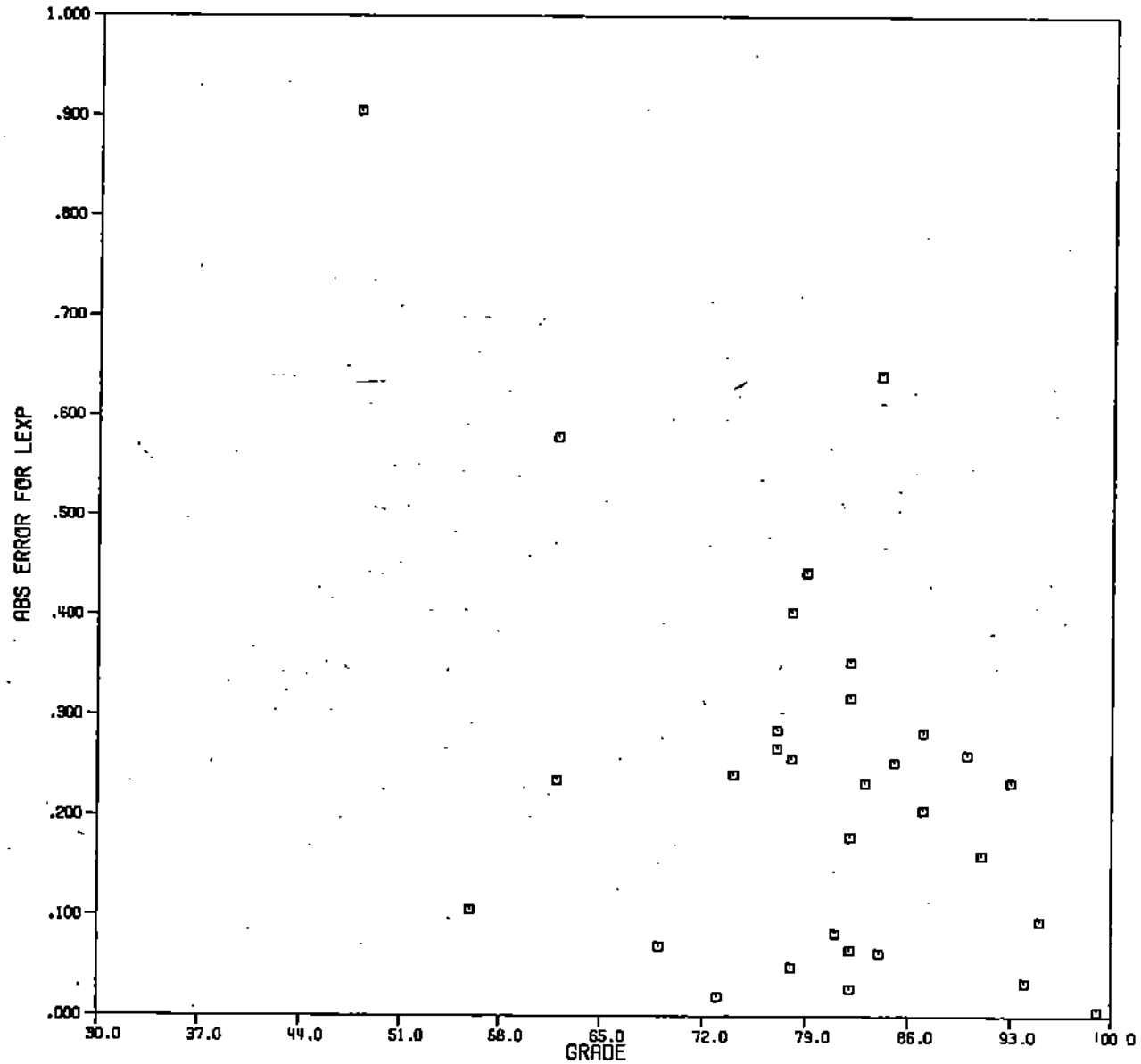

Figure 3. Absolute errors for Project subroutine.

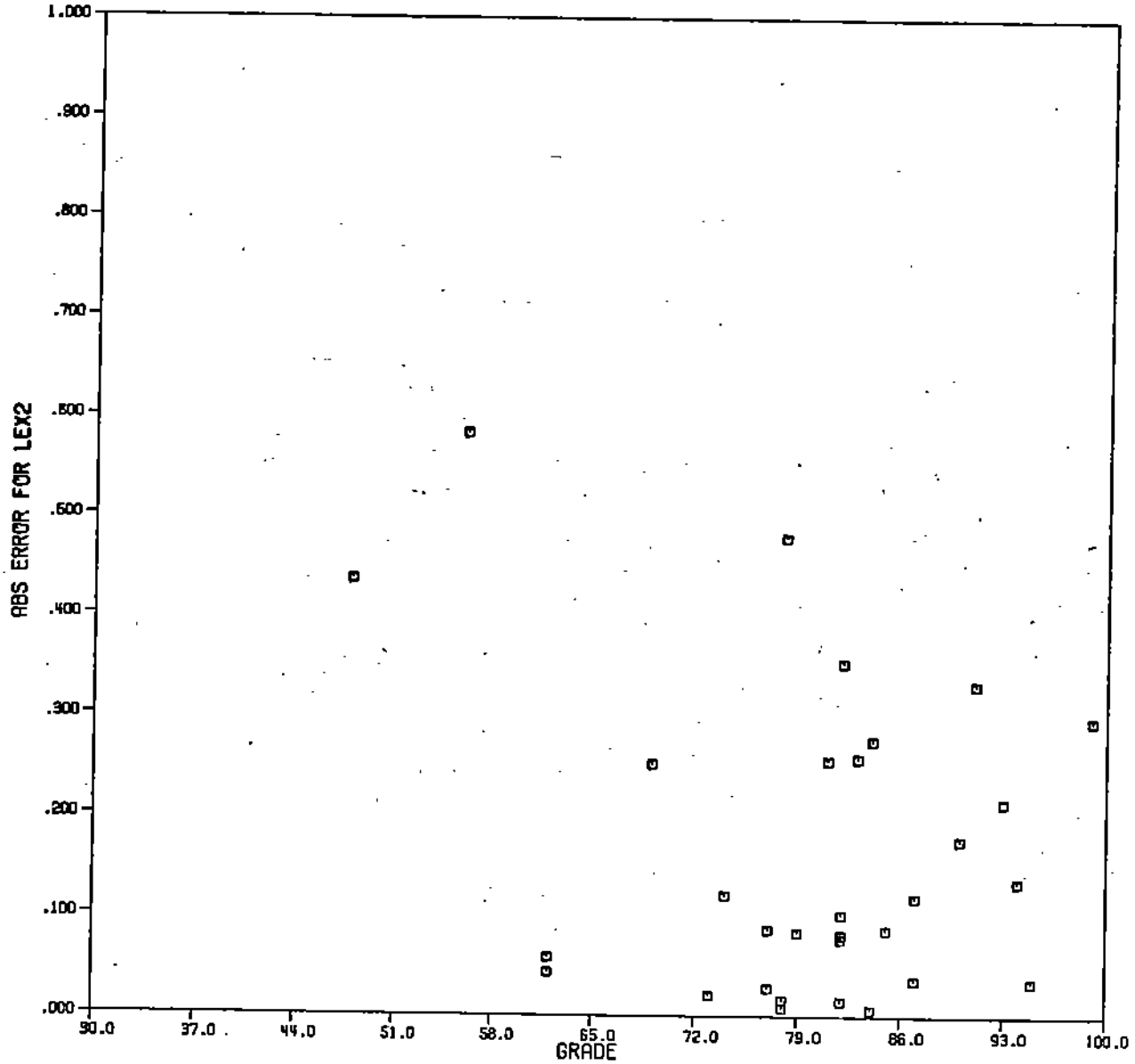

Figure 4. Absolute errors for Program 2.

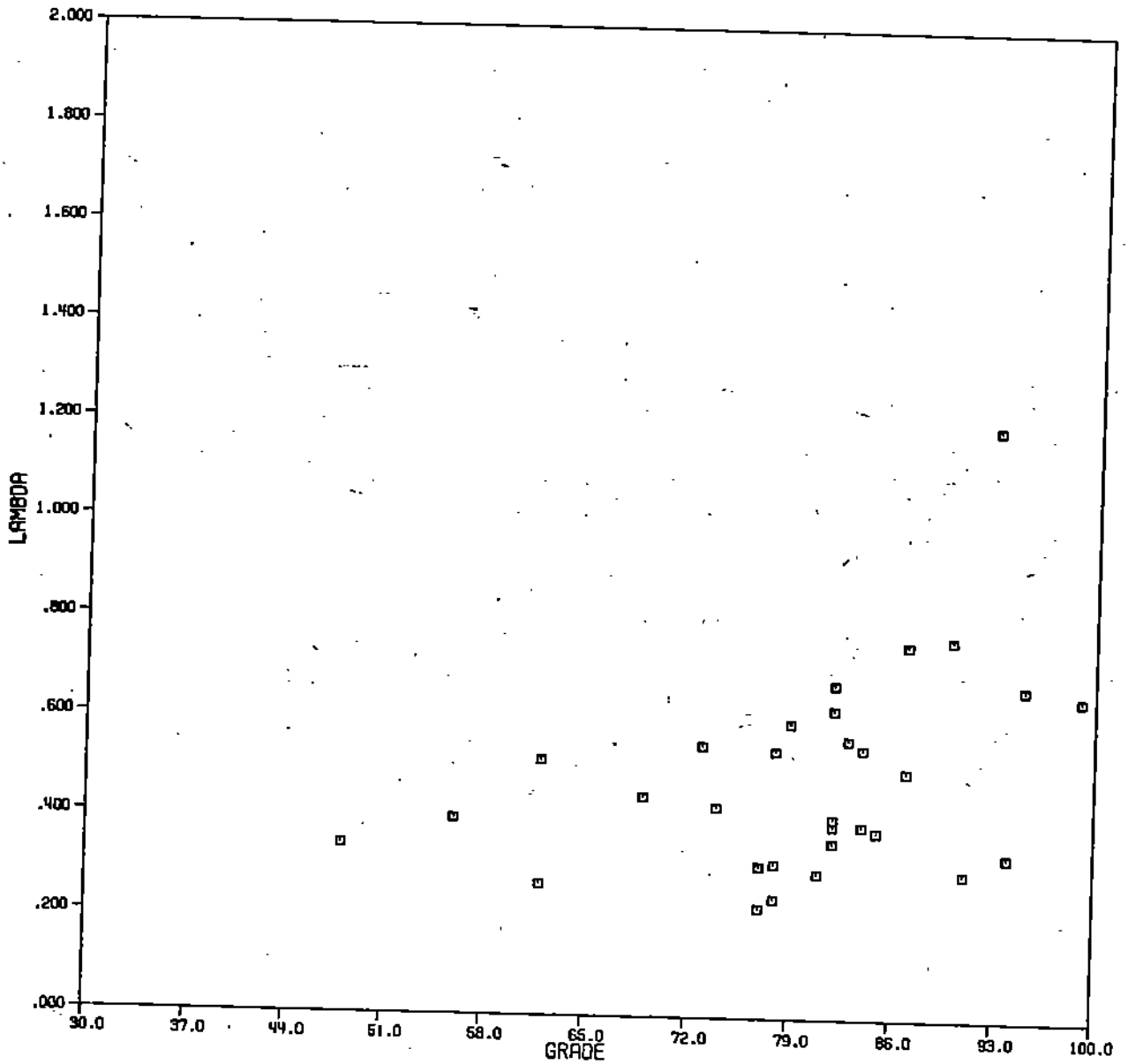

Figure 5. Language levels (λ) for Project subroutine.