EVALUATING AND COMPARING SOFTWARE METRICS IN THE SOFTWARE ENGINEERING LABORATORY*

Victor R. Basili and Tsai-Yun Phillips
Department of Computer Science
University of Maryland
College Park, MD. 20742

I. Introduction

There has appeared in the literature a great number of metrics that attempt to measure the effort or complexity in developing and understanding software(1). There have also been several attempts to independently validate these measures on data from different organizations gathered by different people(2). These metrics have many purposes. They can be used to evaluate the software development process or the software product. They can be used to estimate the cost and quality of the product. They can also be used during development and evolution of the software to monitor the stability and quality of the product.

Among the most popular metrics have been the software science metrics of Halstead, and the cyclomatic complexity metric of McCabe. One question is whether these metrics actually measure such things as effort and complexity. One measure of effort may be the time required to produce a product. One measure of complexity might be the number of errors made during the development of a product. A second question is how these metrics compare with standard size measures, such as the number of source lines or the number of executable statements, i.e., do they do a better job of predicting the effort or the number of errors? Lastly, how do these metrics relate to each other?

One simple way of checking the relationship between errors or effort and the various metrics is to examine the plots of variables against one another and correlations between the various variables. This provides us with a first look at attempting to shed some light on the questions posed and the relationships that may hold.

One of the goals of the Software Engineering Laboratory(3) has been to provide an experimental data base to be used for examining such relationships and providing insights into attempting to answer such questions. The Software Engineering Laboratory is a joint venture between the University of Maryland, NASA/Goddard Space Flight Center, and Computer Sciences Corporation.

The software being analyzed is ground support software for satellites. The systems in this paper consist of 50,000 to 110,000 lines of source code. The source code is predominantly FORTRAN. Anywhere from 10 to 60 percent of the code is reused from previous systems. There are between 200 and 500 modules in each system where a module is defined as a FORTRAN subroutine. The average staff size ranges from 5 to 8 people, including the support personnel.

II. The Data

Data is collected in the Software Engineering Laboratory that deals with many aspects of the development process and product. Among the data collected is the effort to design, code and test the various components of the systems as well as the errors committed during development. The data collected is analyzed to provide insights into software development and to study the effect of various factors on the process and product.

One standard problem in data of this kind is its validity. Unlike the typical controlled experiments where the projects tend to be smaller and the data collection process dominates the development process, the major effort here is the software development process, and the data collectors must effect minimal interference to the developers.

This creates potential problems with the validity of the data. For example, suppose we are interested in the effort expended on a particular module and one programmer forgets to turn in his weekly effort report. This can cause erroneous data for all modules the programmer may have worked on that week. Another problem

*Research supported in part by National Aeronautics and Space Administration grant NSG-5123 to the University of Maryland. Computer time supported in part through the facilities of the Computer Science Center of the University of Maryland.

is how does a programmer report time on the integration testing of three modules? Does he charge the time to the parent module of all three, even though that module may be a small driver module? Clearly that is easier for him to do than to divide the time between all three modules he has worked on.

How does one count errors? An error that is limited to one module is easy to credit. But what about an error that required the analysis of ten modules to determine that it effects changes in three modules? Does one associate the error with all ten modules or the three modules? Does one associate one third of the error with each of the three modules or a full error with all three? It is clear that the larger the system the more complicated the association. All this assumes that all the errors were reported. It is common for programmers not to report clerical errors because the time to fill out the error report form might take longer than the time to fix the error.

In a commercial program development environment, the errors are not seeded so they are not known to the analysis beforehand. The programmers are not watched with respect to the time they put in and report; the full development process may take a year. A class of problems not expected in the controlled development environment is common here and can create problems with obtaining valid results.

The data discussed in this paper is extracted from several sources. First, there is effort data which is taken from a form called the Components Status Report. This form is filled out each week by the programmers on the project. They report the time they spend on each component in the system broken down into the basic phases of design, code and test, as well as any other time they spend on work related to the project, e.g., documentation, meetings, etc.

A component is defined as any named object in the system. A component could be a FORTRAN subroutine, a COMMON block or set of subroutines that makes up a subsystem. The effort data analyzed in this paper is extracted from the Component Status Reports.

Another form, filled out weekly by the project management, is the Resource Form. This form represents accounting data and records all time charged to the project for various project personnel. It is not broken down by activity. This data is used in section IV of this paper to validate the effort data on the components.

The various metrics computed on the source data are calculated automatically by a program called SAP(9) which was developed specifically for the Software Engineering Laboratory by Computer Sciences Corporation. Data collected by the SAP program consists of various software science metrics, such as the Halstead E metric used here, the number of decisions (similar to the McCabe cyclomatic complexity metric), the number of source statements, the number of executable statements and the number of call statements. These metrics are computed at the component level. The number of source lines consists of the total number of lines in the source text, including comments and data statements. The number of executable statements consists of only the executable FORTRAN statements, excluding comments and data statements such as COMMON declarations. Typically, the number of executable statements is about 50 percent to 60 percent of the total number of source lines.

The error count discussed here is collected from a form called the Change Report Form which is filled out each time a change is made to the system. These reports are normally not filled out until testing has begun. The error count consists of only those changes which have been classified as errors. Nonerror changes are not discussed in this paper.

III. A First Pass

We began by examining four projects which we shall call A,I,P and S. For each of these projects we considered the aspects of the development separately and in combinations. These phases are design, coding, and testing phases. In considering all available components, A had 111 components, I had 55 components, P had 229 components, and S had 118 components for which we had some effort data and a software science E measure. It turned out that the union of coding and testing, as well as total effort, gave us the best results:

Project	Design	Code	Test	Design & Code	Design & Test	Code & Test	Total
A	.4563	.4700	.4212	.4775	.5444	.6380	.6599
I	-.0503	.0322	.0094	.0931	.0942	.0977	.0500
P	.3817	.4316	.3946	.4301	.4296	.4296	.4660
S	.3658	.3957	.4015	.4157	.4688	.4688	.5459

The lowest correlation between effort and the E metric were in project I. As it turns out, project I had the most reused code from previous projects. That is, modules from previous projects were taken wholly or slightly modified for the use in SystemI. Since this factor was obviously affecting the relationship, we classified all the modules studied as either newly developed, modified, or old. We then recalculated the relationship between effort and the E metric using only newly developed components. The results are given below for total effort only.

Project	# of Components	Total
A	101	.6774
I	31	.4162
P	178	.6230
S	106	.4580

The correlations here are higher, as expected, because of the better data.

We were interested in whether other measures, such as lines of source code and executable statements, provided better relationships as well as the relationship between metric E and these other measures. We were also interested in whether other factors affected the correlations. For example, what effect would a division of the modules by such factors as size, complexity and testing level have?

First, a study of all 416 components across the four systems yielded the following correlations:

	E	Source Lines	Executable Statements
		.7497	.8031
tual Effort	.6384	.5795	.4949

Next, division of the components by the amount of the time spent in development effort showed better correlations for those projects in which more time was spent. The division by the number of lines, however, did not show a clear trend. This provided us with the idea that some of the effort data might be missing at the component level and, therefore, we should eliminate those components for which the effort data was not good enough. The results of this validation are reported in the next section.

The separation of components by complexity was based upon an evaluation of the complexity of the particular component by the programmer. Components were rated as hard, moderate or easy. In general, higher correlations between effort and all other variables grew as the subjective complexity rating grew.

The results of separating the components by various subsystems that were common across the projects, as well as by various testing levels (such as tree chart subsystem levels), seemed inconclusive. These variations will be examined again in light of the data validation discussed in the next section.

IV. A Second Pass

Because the correlations between effort and the various size metrics were better for those components with greater effort, we became concerned that the results might be due to poor reporting of effort data. To check this, we proposed a validation check on the data, providing each component with a validity rating. For each programmer on a project, we examined both the total time reported on the Component Status Report, as well as the total time charged to the project. We then placed components into categories depending upon the percent of time reported by the programmer on the Component Status Report compared to the percent of time charged to the project, and gave the components an accuracy rating. For example, if all the programmers working on component X reported at least 90 percent of their total resource time on the Component Status Report, then X is in the $\geq$ 90 percent category.

Besides examining the E metric, the source lines and executable statement counts, we also analyzed the cyclomatic metric and the number of calls contained within a component. The correlation

between actual effort and these complexity metrics is given in table 1(a) and (b) for those projects with greater than 90 percent accuracy and greater than 84 percent accuracy. Figures 1,2,3, and 4 provide plots of the data points at the 84 percent and 90 percent accuracy levels for source lines and the E metric with effort.

The correlations between the various factors appear to be better on the validity rated data than on the full data and appears to do better as the 90 percent validity rated level than at the 84 percent validity rated data. For this reason, we believe that the validity rated data is more reliable than the earlier data.

Since complexity is also meant to measure the number of errors associated with the development of a project, we compared the various complexity measures, including the total effort required for development with the number of errors. An error was associated with a component if it was isolated to that component or the component was one of several involved in the error. Table 1 also gives the correlations between the error count and the various complexity metrics. Figures 5,6, and 7 provide plots of the data points at the 84 percent accuracy level for the actual effort, source lines and Halstead's E metric compared with the error count.

Another question is whether we can predict or account for the actual effort or error count using the metrics discussed so far. In an attempt to study this problem we applied a forward multiple regression analysis using the other metrics to account for effort. Using the data for effort at the 84 percent validity level, we came up with the following order: executable statements (XQT), number of errors (ERR), E metric (E), cyclomatic complexity (CC) and source lines. The number of calls was never included. Table 2 shows the amount of variation explained (R^2) as each new factor is included in the equation. Based on a .05 level of significance in using the last variable included, the regression equation generated was

Effort = 19.9*XQT + 107.5*ERR - 1.2*E
-24.7 CC + 250.5

Dependent Variable . . . Effort	
variables	R^2
Executable Statements	.6358
Error Count	.6792
E	.7110
Cyclomatic Complexity	.74571
Source Lines	.74966

Table 2

Pearson Correlation Coefficients

≥ 90% Reported Programmers

	Effort	Error	Halstead	XQT	Source	McCabe -1	Calls
Effort	1.0000*	.6346*	.6612*	.7974*	.7583*	.7399*	.6033*
Error	.6346*	1.0000*	.5432*	.5837*	.5576*	.5592*	.4861*
Halstead	.6612*	.5432*	1.0000*	.9160*	.8706*	.8906*	.8818*
XQT	.7974*	.5837*	.9160*	1.0000*	.9513*	.9777*	.8258*
Source	.7583*	.5576*	.8706*	.9513*	1.0000*	.9519*	.8726*
McCabe -1	.7399*	.5592*	.8906*	.9777*	.9519*	1.0000*	.8110*
Calls	.6033*	.4861*	.8818*	.8258*	.8727*	.8110*	1.0000*

cases = 37 (data points)

* - significance ≤ .001

Table 1(a)

Pearson Correlation Coefficients
Across Projects (≥ 84% Reported Programmers)

	Effort	Error	Halstead	XQT	Source	McCabe -1	Calls
Effort	1.0000*	.6227*	.6719*	.5094*	.6025*	.3261*	.6666*
Error	.6227*	1.0000*	.5028*	.4289*	.4891*	.3045*	.6431*
Halstead	.6719*	.5028*	1.0000*	.8301*	.7565*	.6540*	.8044*
XQT	.5094*	.4289*	.8301*	1.0000*	.8061*	.9116*	.7703*
Source	.6025*	.4891*	.7565*	.8061*	1.0000*	.6533*	.7759*
McCabe -1	.3261*	.3045*	.6540*	.9116*	.6533*	1.0000*	.5990*
Calls	.6666*	.6431*	.8044*	.7703*	.7759*	.5990*	1.0000*

cases = 116 (data points)

* - significance ≈ .001

Table 1(b)

There has been some work done in isolating the individual programmers. There is some evidence that a better correlation exists between the effort or error count of an individual programmer and a particular complexity metric. Some work will also be done in examining specific error classes and complexity

Further validation of the data needs to be done in examining some of the outlying points. For example, a point with a high number of source lines but low effort rating might be a COMMON block and therefore eliminated from the study of control flow components.

V. Conclusion

There is hope in using commercially-obtained data rather than experimentally-obtained data to validate complexity metrics. It is possible to systematically clean up the data and study the relationships and accountability do exist between various complexity metrics, effort and error counts. The results tend to get better as the data used appears to be more reliable.

References

(1) Halstead, M., Elements of Software Science, Elsevier North-Holland, New York, 1977.

(2) McCabe, T.J., "A Complexity Measure," IEEE Transactions on Software Engineering, 1976, 2, 308-320.

(3) Gaffney, John, "Program Control Complexity"; Proceedings of the "Workshop on Quantitative Software Models for Reliability, Complexity, and Cost, IEEE Computer Society, Oct. 1979.

(4) Chen, E.T., "Program Complexity and Programmer Productivity," IEEE Transactions on Software Engineering, May 1978, Vol. SE-4, No. 3, pp. 187-194.

(5) Curtis, Sheppard, & Milliman, "Third Time Charm: Stronger Prediction of Programmer Performance by Software Complexity Metrics," Proceedings of the Fourth International Conference on Software Engineering, 1979, pp. 356-360.

(6) Feuer and Fowlkes, "Some Results from an Empirical Study of Computer Software," Proceedings of the Fourth International Conference on Software Engineering, 1979, pp. 351-355.

(7) Basili, V., "Tutorial on Models and Metrics for Software Management and Engineering," IEEE Computer Society, IEEE Catalog No. EHO-167-7, 1980.

(8) Basili and Zelkowitz, "Analyzing Medium Scale Software Developments," Third International Conference on Software Engineering, Atlanta, Georgia, May 1978.

(9) O'Neil, E., "The Static Source Code Analyzer's Users Guide," CSC TM-78/6045, 1978.

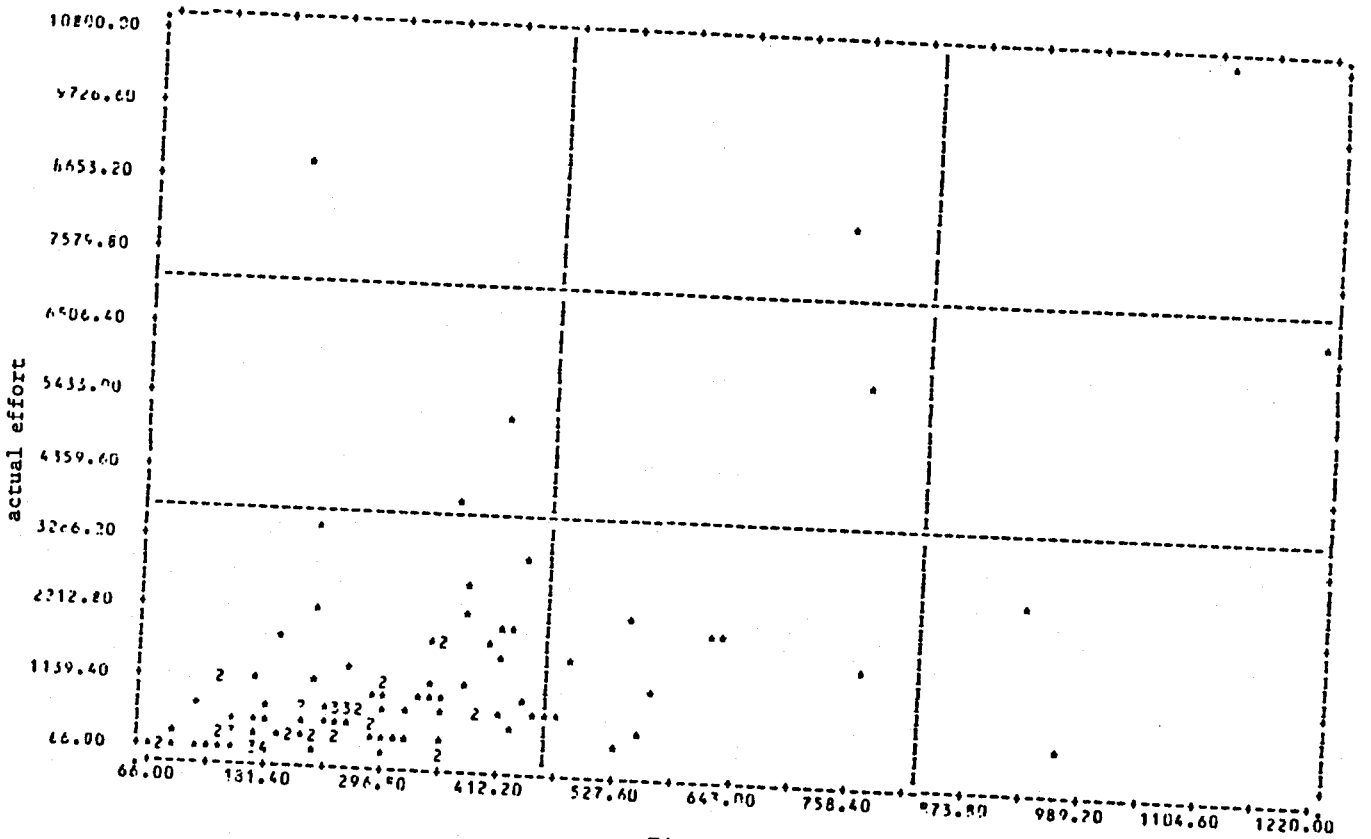

Figure 1

84% accuracy level
Correlation (R): .6025
R square (R^2): .3630
Significance: .00001

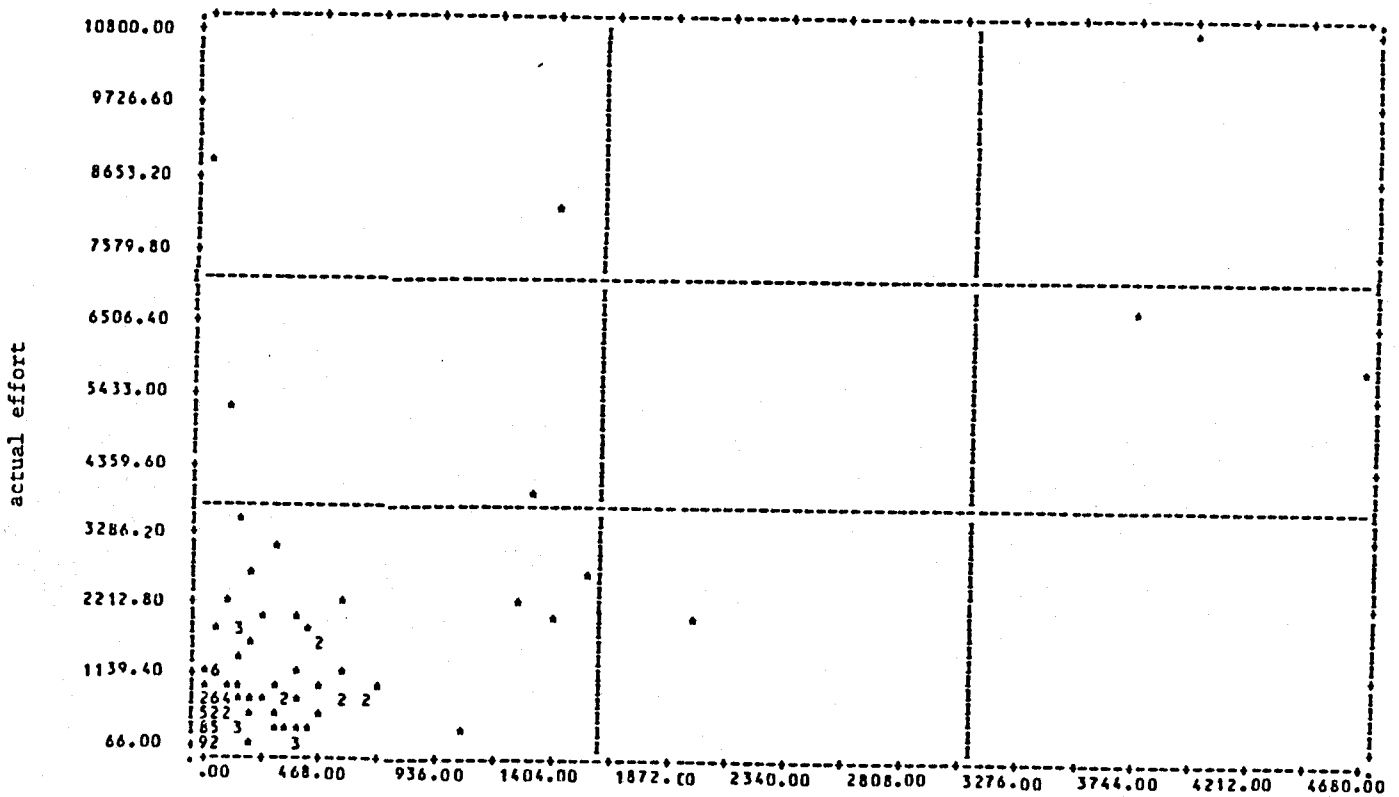

Figure 2

84% accuracy level

Correlation (R): .6719

R square (R^2): .4515

Significance: .00001

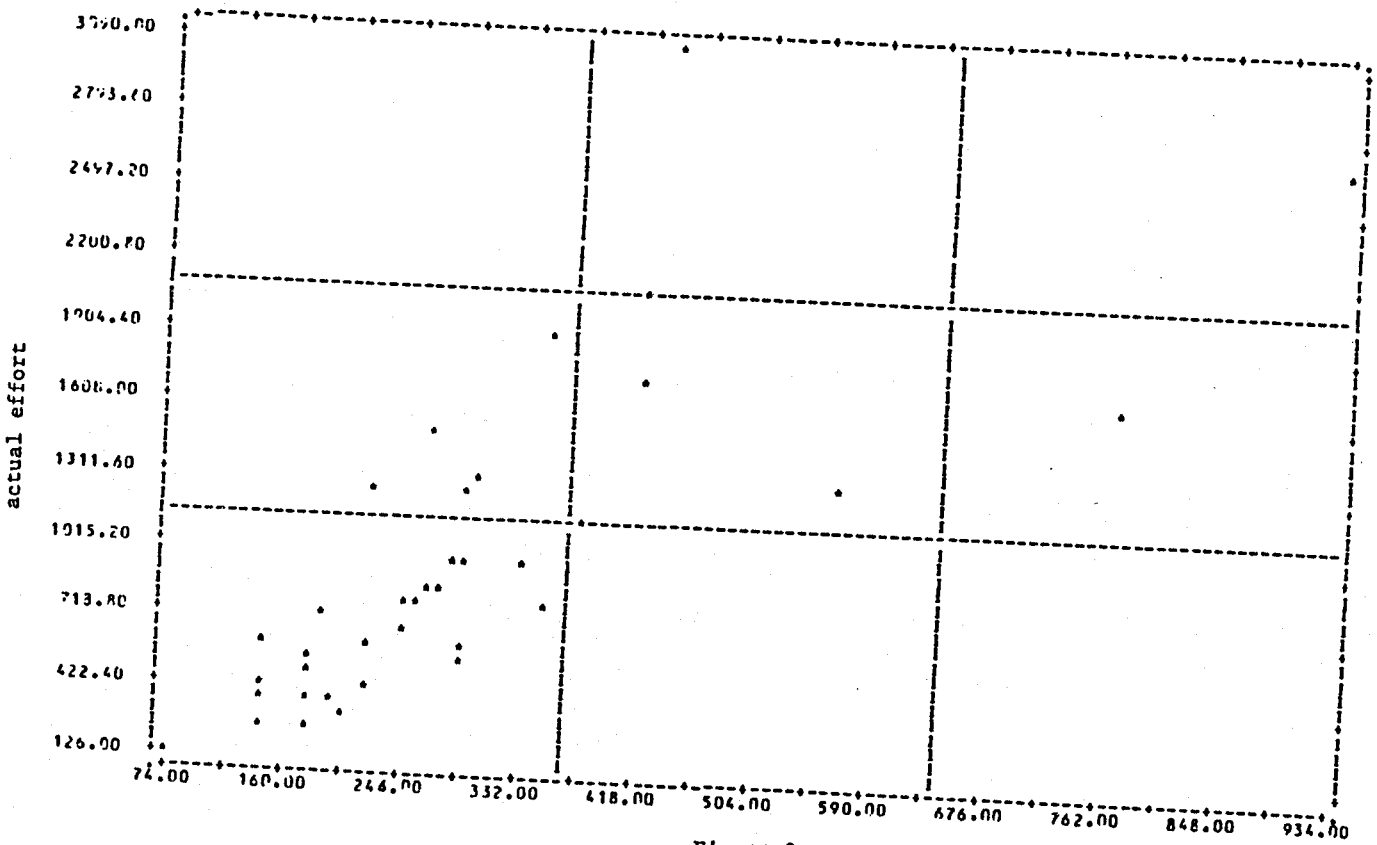

Figure 3

90% accuracy level

Correlation (R): .7583

R square (R^2): .5751

Significance: .00001

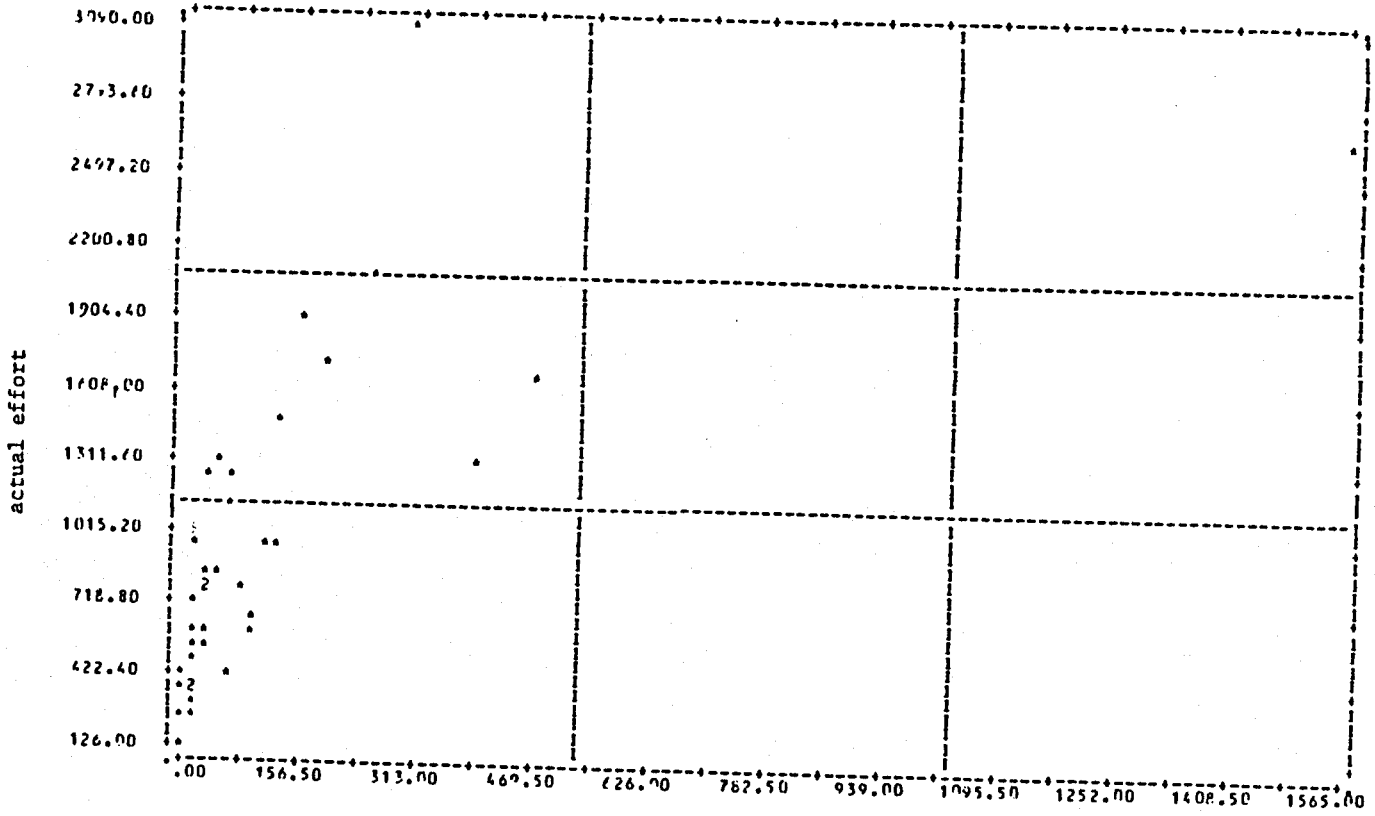

Figure 4

90% accuracy level

Correlation (R): .6612

R square (R^2): .4371

Significance: .00001

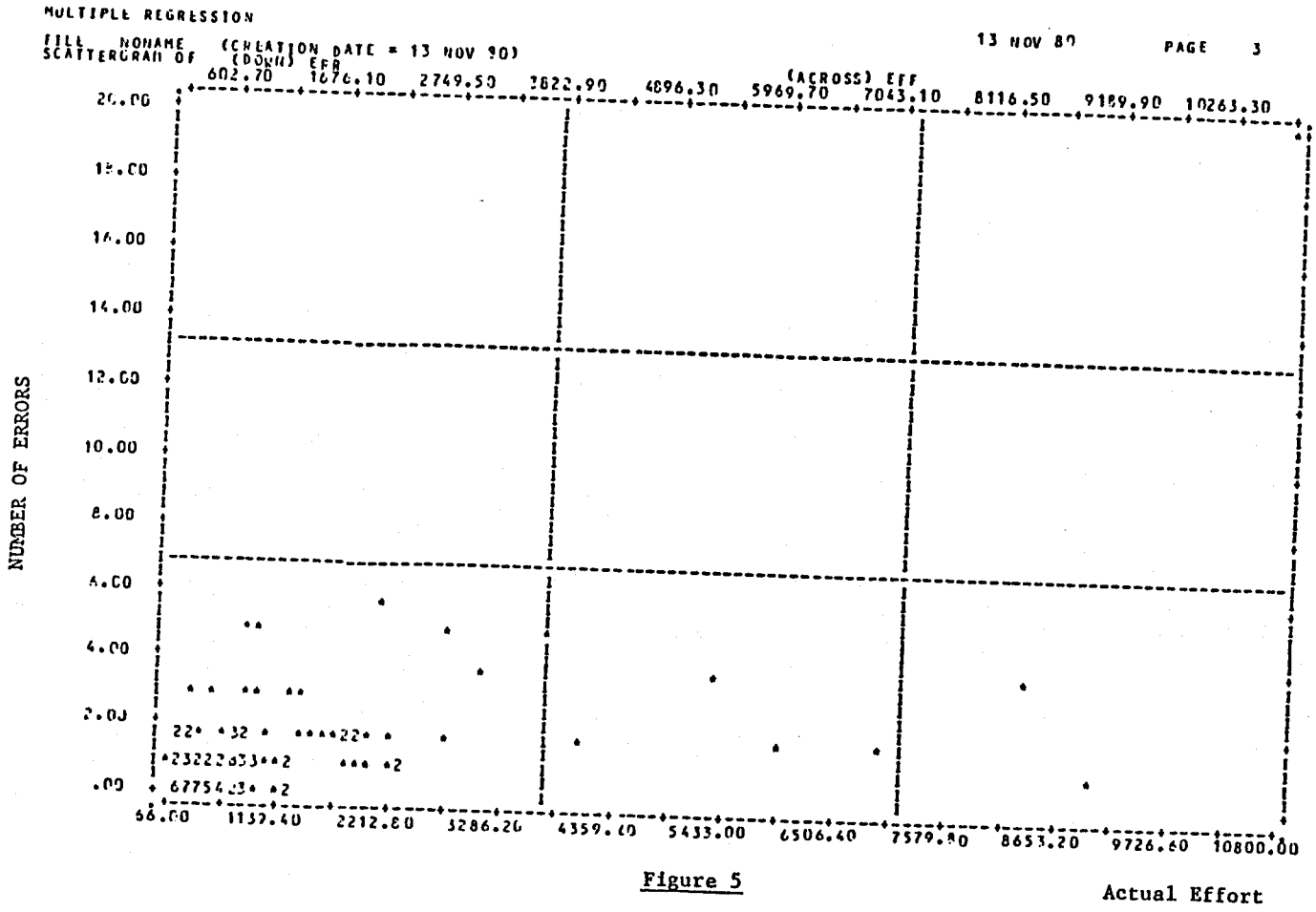

Figure 5

84% accuracy level

Correlation (R):	.6227
R square (R^2):	.3877
Significance:	.00001

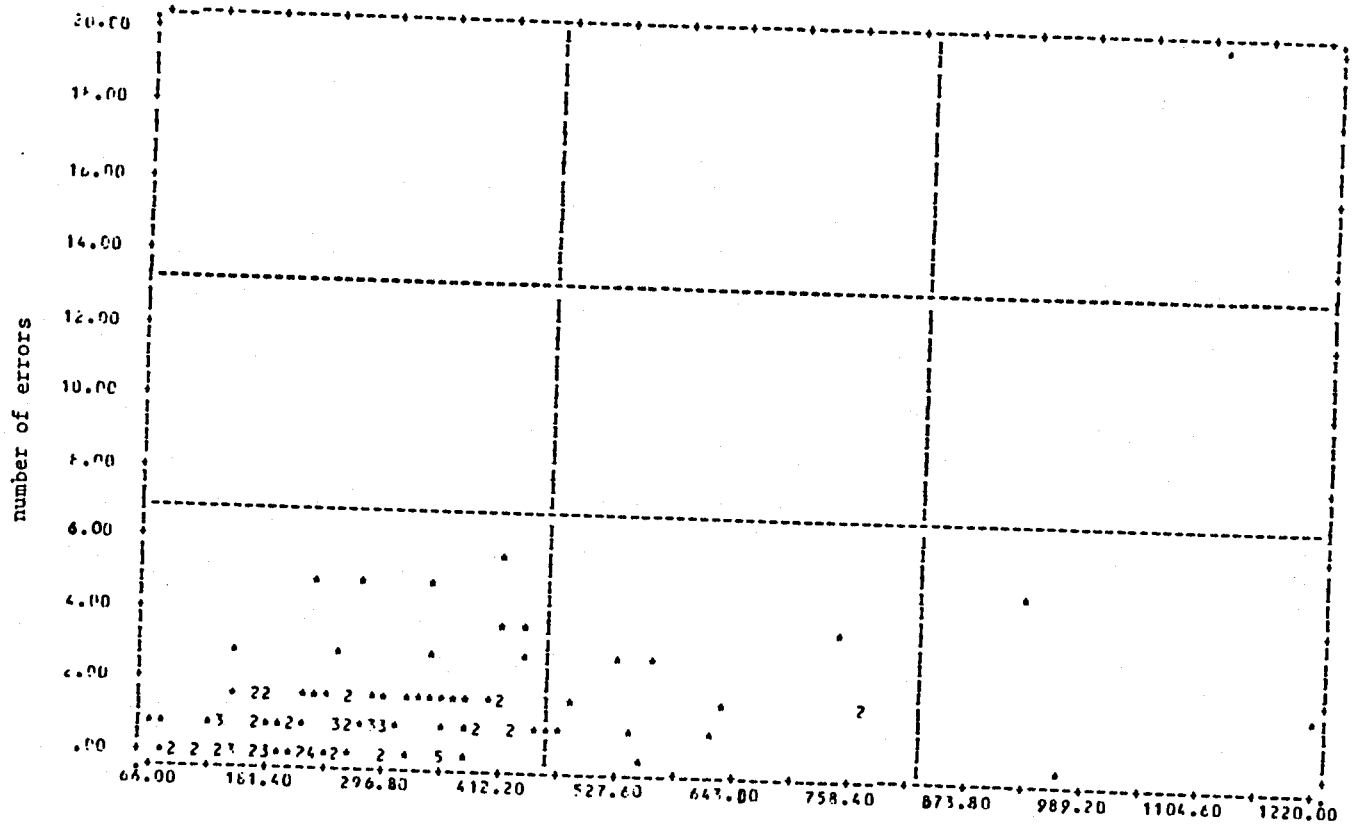

Figure 6

84% accuracy level

Correlation (R): .4890

R square (R^2): .2392

Significance: .00001

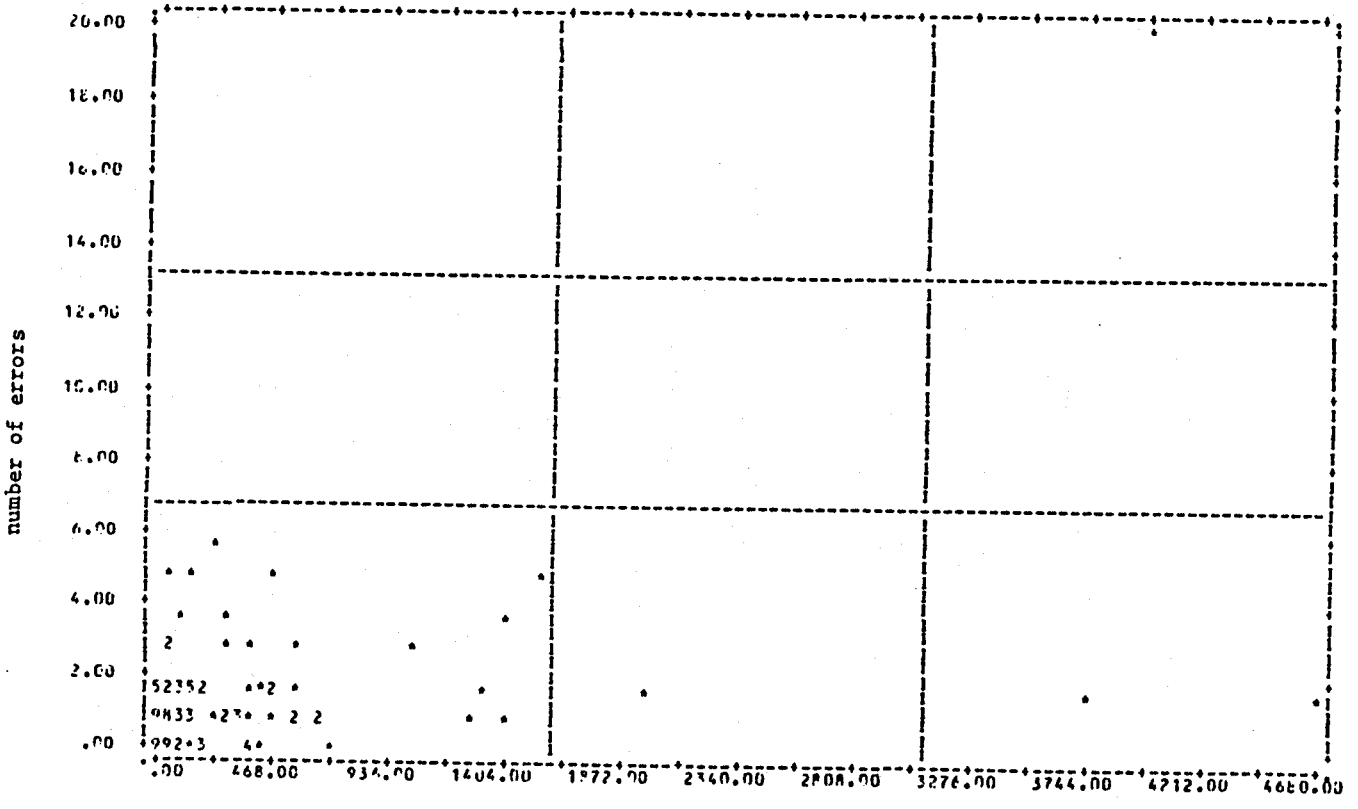

Figure 7

84% accuracy level

Correlation (R):	.5028
R square (R^2):	.2528
Significance:	.00001